

PDFASSIST: REQUIREMENTS DOCUMENT

By

STEPHANIE BELOW
JONATHAN MERCER

SPRING 2017
WRIGHT STATE UNIVERSITY

PDFASSIST: Requirements Document (version 1.0)

Project: CS 7140 PDF Renaming Project

Date(s): February 8, 2017

Prepared by: Jonathan Mercer & Stephanie Below

Document status: ☒ Draft ☐ Proposed ☐ Validated ☐ Approved

1. Introduction

This document contains the system requirements for **PDFAssist**. These requirements have been derived from several sources.

1.1 Purpose of This Document

This document is intended to guide development of **PDFAssist**. It will go through several stages during the course of the project:

1. **Draft:** The first version, or draft version, is compiled after requirements have been discovered, recorded, classified, and prioritized.
2. **Proposed:** The draft document is then proposed as a potential requirements specification for the project. The proposed document should be reviewed by several parties, who may comment on any requirements and any priorities, either to agree, to disagree, or to identify missing requirements. Readers include end-users, developers, project managers, and any other stakeholders. The document may be amended and repropose several times before moving to the next stage.
3. **Validated:** Once the various stakeholders have agreed to the requirements in the document, it is considered validated.
4. **Approved:** The validated document is accepted by representatives of each party of stakeholders as an appropriate statement of requirements for the project. The developers then use the requirements document as a guide to implementation and to check the progress of the project as it develops.

1.2 Scope of the Product

PDFAssist will be used to rename large numbers of research paper PDF files. The names of the new files will be based off certain semantics of the paper.

1.3 Business Case for the Product

It is possible that a user may wish to search a large number of PDFs for articles by specific authors, with specific titles, or some other important attribute. A useful addition to PDFAssist would be a data retrieval system, able to query and return PDFs that meet specific criteria. This system could make use of either the parsing logic and/or the files after they have been renamed to fulfill this requirement.

2. Specific Requirements

This section of the document lists specific requirements for PDFAssist. Requirements are divided into the following sections:

1. User requirements. These are requirements written from the point of view of end users, usually expressed in narrative form.
2. System requirements. These are detailed specifications describing the functions the system must be capable of doing.
3. Interface requirements. These are requirements about the user interface, which may be expressed as a list, as a narrative, or as images of screen mock-ups.

4. Maintainability requirements. These are requirements about the source code that will be used to implement a design based on these requirements.
5. Acceptance Tests. These show examples of prospective documents the program would be expected to take as input, and the expected output for each document.

2.1 User Requirements

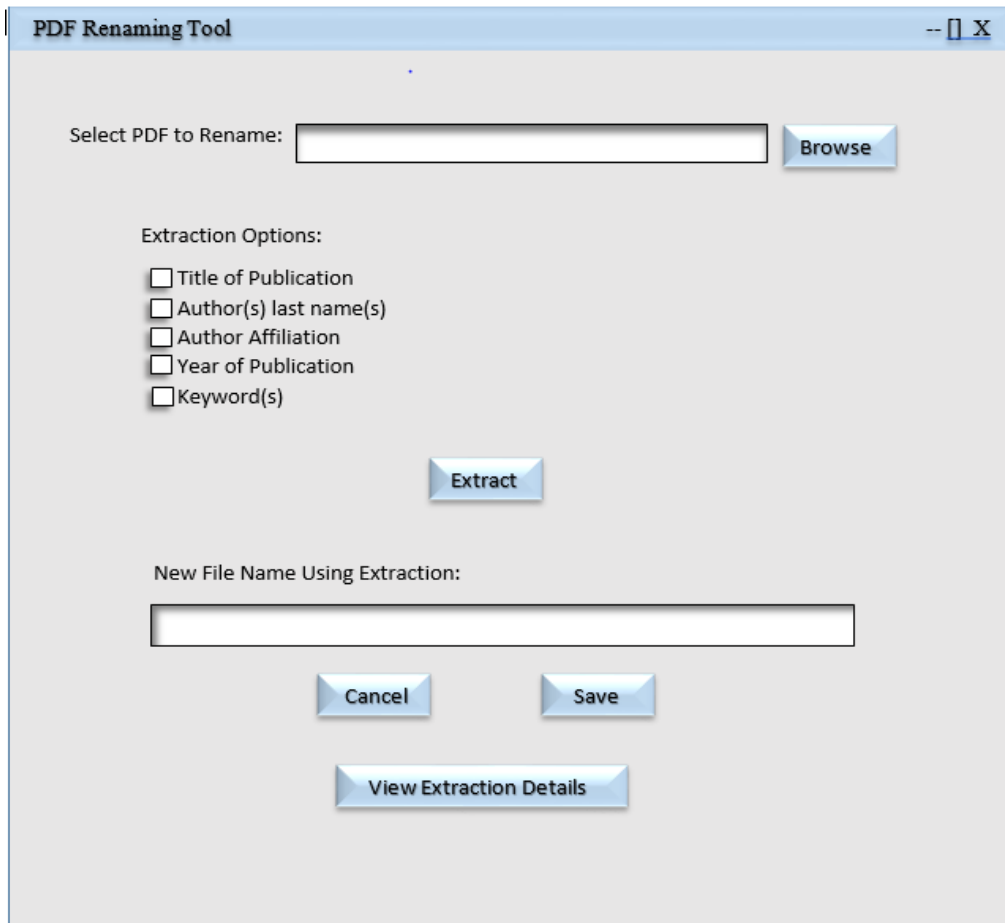
1. The program should accept any scholarly article in PDF format as an input.
2. The program should parse an input and determine its title.
3. The program should parse an input and determine its year of publication.
4. The program should parse an input and determine at least one keyword that summarizes its subject matter.
5. The program should parse an input and determine any and all last names of its authors.
6. The program should parse an input and determine any and all affiliations of its authors.
7. The program should rename an input with a name that includes its title.
8. The program should rename an input with a name that includes its year of publication.
9. The program should rename an input with a name that includes its keyword(s).
10. The program should rename an input with a name that includes its author names.
11. The program should rename an input with a name that includes its author affiliations.

2.2 Functional Requirements

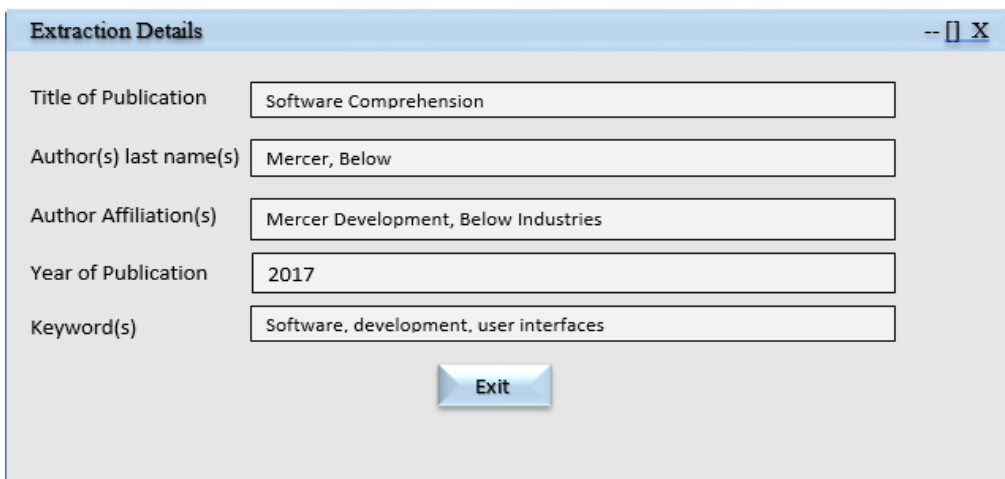
12. The program should throw an error upon determining that an input is not in PDF format.
13. If the program throws an error as stated in requirement 12, it should not rename the input.
14. The program should throw an error if it is unable to extract the input's title.
15. The program should throw an error if it is unable to extract the input's year of publication.
16. The program should throw an error if it is unable to extract the input's keyword(s).
17. The program should throw an error if it is unable to extract the input's author names.
18. The program should throw an error if it is unable to extract the input's author affiliations.
19. If the program throws an error for any reason as described in requirements 14 through 18, it should rename the input file as detailed in requirements 7 through 11 with what information it was able to successfully extract as detailed in requirements 2 through 6.
20. If the program is forced to terminate before extracting all the relevant data from the file as described in requirements 2 through 6, the program should not rename the input.
21. The program should be able to rename or throw an error in response to 100,000 inputs in no more than one minute.
22. The program should not alter any part of an input except for its name.
23. The program's methods should minimize cyclomatic complexity whenever possible.

2.3 Interface Requirements

The following images show a basic mockup of a possible user interface for PDFAssist. Any implementation of these requirements should include options to select a PDF file and specify what information to extract or not extract. The interface will output the resulting file name and display the information extracted in a separate window.



The mockup shows a window titled "PDF Renaming Tool" with a standard Windows-style title bar. Inside, there is a label "Select PDF to Rename:" followed by a text input field and a "Browse" button. Below this is a section titled "Extraction Options:" containing five unchecked checkboxes: "Title of Publication", "Author(s) last name(s)", "Author Affiliation", "Year of Publication", and "Keyword(s)". A central "Extract" button is positioned below the checkboxes. Underneath the "Extract" button is the label "New File Name Using Extraction:" followed by another text input field. At the bottom of the window are three buttons: "Cancel", "Save", and "View Extraction Details".



The mockup shows a window titled "Extraction Details" with a standard Windows-style title bar. It contains five rows of data, each with a label and a text input field containing specific text: "Title of Publication" with "Software Comprehension", "Author(s) last name(s)" with "Mercer, Below", "Author Affiliation(s)" with "Mercer Development, Below Industries", "Year of Publication" with "2017", and "Keyword(s)" with "Software, development, user interfaces". An "Exit" button is located at the bottom center of the window.

2.4 Maintenance Requirements

1. Each method in the program should include documentation describing the role of any parameter passed into the method.
2. Each method in the program should include documentation describing the weakest precondition of that method.
3. Each method in the program should include documentation describing the strongest postcondition of that method.
4. The program should include unit tests to ensure that the program meets all functional requirements.

2.5 Acceptance Tests

The program should, given the following three research papers as cited in the References section as input, parse the following information:

2.5.1 Document 1

MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS

Dr. Winston W. Royce

INTRODUCTION

I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.

COMPUTER PROGRAM DEVELOPMENT FUNCTIONS

There are two essential steps common to all computer program developments, regardless of size or complexity. There is first an analysis step, followed second by a coding step as depicted in Figure 1. This sort of very simple implementation concept is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it — as is typically done with computer programs for internal use. It is also the kind of development effort for which most customers are happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product. An implementation plan to manufacture larger software systems, and keyed only to these steps, however, is doomed to failure. Many additional development steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development costs. Customer personnel typically would rather not pay for them, and development personnel would rather not implement them. The prime function of management is to sell these concepts to both groups and then enforce compliance on the part of development personnel.

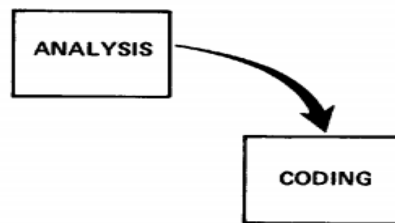


Figure 1. Implementation steps to deliver a small computer program for internal operations.

A more grandiose approach to software development is illustrated in Figure 2. The analysis and coding steps are still in the picture, but they are preceded by two levels of requirements analysis, are separated by a program design step, and followed by a testing step. These additions are treated separately from analysis and coding because they are distinctly different in the way they are executed. They must be planned and staffed differently for best utilization of program resources.

Figure 3 portrays the iterative relationship between successive development phases for this scheme. The ordering of steps is based on the following concept: that as each step progresses and the design is further detailed, there is an iteration with the preceding and succeeding steps but rarely with the more remote steps in the sequence. The virtue of all of this is that as the design proceeds the change process is scoped down to manageable limits. At any point in the design process after the requirements analysis is completed there exists a firm and closeup, moving baseline to which to return in the event of unforeseen design difficulties. What we have is an effective fallback position that tends to maximize the extent of early work that is salvageable and preserved.

Reprinted from *Proceedings, IEEE WESCON*, August 1970, pages 1-9.
Copyright © 1970 by The Institute of Electrical and Electronics Engineers, Inc. Originally published by TRW. 328

Given Document 1, the title should be “Managing the Development of Large Software Systems”, the author(s) should be “Royce”, the year of publication should be “1970”, and the keywords and author affiliation should be unparsable.

2.5.2 Document 2

A Spiral Model of Software Development and Enhancement

Barry W. Boehm, TRW Defense Systems Group

“Stop the life cycle—I want to get off!”
“Life-cycle Concept Considered Harmful.”
“The waterfall model is dead.”
“No, it isn’t, but it should be.”

These statements exemplify the current debate about software life-cycle process models. The topic has recently received a great deal of attention.

*The Defense Science Board Task Force Report on Military Software*¹ issued in 1987 highlighted the concern that traditional software process models were discouraging more effective approaches to software development such as prototyping and software reuse. The Computer Society has sponsored tutorials and workshops on software process models that have helped clarify many of the issues and stimulated advances in the field (see “Further reading”).

The spiral model presented in this article is one candidate for improving the software process model situation. The major distinguishing feature of the spiral model is that it creates a *risk-driven* approach to the software process rather than a primarily *document-driven* or *code-driven* process. It incorporates many of the strengths of other models and resolves many of their difficulties.

This article opens with a short description of software process models and the issues they address. Subsequent sections outline the process steps involved in the

This evolving risk-driven approach provides a new framework for guiding the software process.

spiral model; illustrate the application of the spiral model to a software project, using the TRW Software Productivity Project as an example; summarize the primary advantages and implications involved in using the spiral model and the primary difficulties in using it at its current incomplete level of elaboration; and present resulting conclusions.

Background on software process models

The primary functions of a software process model are to determine the *order of the stages* involved in software development and evolution and to establish the *transition criteria* for progressing from one stage to the next. These include completion

criteria for the current stage plus choice criteria and entrance criteria for the next stage. Thus, a process model addresses the following software project questions:

- (1) What shall we do next?
- (2) How long shall we continue to do it?

Consequently, a process model differs from a software method (often called a methodology) in that a method’s primary focus is on how to navigate through each phase (determining data, control, or “uses” hierarchies; partitioning functions; allocating requirements) and how to represent phase products (structure charts; stimulus-response threads; state transition diagrams).

Why are software process models important? Primarily because they provide guidance on the order (phases, increments, prototypes, validation tasks, etc.) in which a project should carry out its major tasks. Many software projects, as the next section shows, have come to grief because they pursued their various development and evolution phases in the wrong order.

Evolution of process models. Before concentrating in depth on the spiral model, we should take a look at a number of others: the code-and-fix model, the stage-wise model and the waterfall model, the evolutionary development model, and the transform model.

The code-and-fix model. The basic model used in the earliest days of software

Given Document 2, the title should be “A Spiral Model of Software Development and Enhancement”, the author(s) should be “Boehm”, the year of publication should be “1988”, the author affiliation should be “TRW Defense Systems Group”, and the keywords should be unparsable.

2.5.3 Document 3

CHI 99 15-20 MAY 1999	ACM ISBN: 1-58113-158-5	Tutorials
<h2>The Usability Engineering Lifecycle</h2>		
<p>Deborah J. Mayhew Deborah J. Mayhew & Associates 88 Panhandle Road Post Office Box 248 West Tisbury, MA 02575-0248 USA +1 508-693-7149 drdeb@vineyard.net http://vineyard.net/biz/drdeb/index.html</p>		
<p>ABSTRACT The purpose of this tutorial is to provide a lifecycle of practical usability tasks and techniques for structuring the process of designing good user interfaces to either traditional software applications or Web pages and applets. The tutorial presents techniques which can be applied at different points in a typical product development lifecycle. Techniques presented include not only requirements analysis, design and testing techniques, but also organizational and managerial strategies.</p>	<p>Over the past 15 years, however, there has been a growing recognition that the success of software products in the marketplace, and as tools in business environments, depends to a significant degree on the usability of the product, that is, on the quality of the user interface. Web developers are just beginning to recognize usability as a critical success factor.</p>	
<p>Keywords User interface design, user profile, task analysis, usability goals, style guide, conceptual model, usability testing, usability evaluation, usability organization, cost-benefit analysis</p>	<p>Ad hoc design based on intuition and limited experience is not enough to insure the usability of a software application. The literature offers many principles for good interface design [1]. These principles can be helpful guidelines for designers. However, even if every designer in a software development organization was well versed in these design principles, this would not be enough to ensure good</p>	
<p>TUTORIAL CONTENT</p>		

Given Document 3, the title should be “The Usability Engineering Lifecycle”, the author(s) should be “Mayhew”, the year of publication should be “1999”, the author affiliation should be “Mayhew and Associates”, and the keywords should be “user interface design, user profile, task analysis, usability goals, style guide, conceptual model, usability testing, usability evaluation, usability organization, cost-benefit analysis”.

2.5.4 Document 4

Understanding Software Development Through Networks

Christopher Roach
Apple, Inc.
19333 Vallco Parkway, Building A
Cupertino, CA 95014
croach@apple.com

ABSTRACT

Software engineering, being a relatively new field, has struggled to find ways of gauging the success/failure of development projects. The ability to determine which developers are most crucial to the success of a project, which areas in the project contain the most risk, etc. has remained elusive thus far. Metrics such as SLOC (Source Lines of Code) continues to be used to determine the efficacy of individual developers on a project despite many well-documented deficiencies. In this work, I propose a new way to look at software development using network science. I use net-

metrics such as SLOC count were first used and yet the use of these metrics to evaluate individual developers' worth within an organization still pervades the industry. However, software development projects are no longer small with few developers; most are complex, involving sometimes hundreds of developers and millions of lines of code. Despite the fact that it is now common knowledge that traditional metrics are poor indicators of a developer's value in today's world, they can still be found in every corner of the industry [3]. In this study, I argue that individual-based, static statistics such as SLOC count, and other similar size count metrics,

Given Document 4, the title should be "Understanding Software Development Through Networks", the author(s) should be "Roach", the year of publication should be unparsable, the author affiliation should be "Apple Inc.", and the keywords should be unparsable.

Document 5.5.5

Proceedings of the 38th Hawaii International Conference on System Sciences - 2005

A TOPOLOGICAL ANALYSIS OF THE OPEN SOURCE SOFTWARE DEVELOPMENT COMMUNITY

Jin Xu Yongqin Gao Scott Christley Gregory Madey

Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556

Email: {jxu1, ygao1, schrist1, gmadey}@nd.edu
Tel: (574)631-7596 Fax: (574)631-9260

ABSTRACT

The fast growth of OSS has increased the interest in studying the composition of the OSS community and its collaboration mechanisms. Moreover, the success of a project may be related to the underlying social structure of the OSS development community. In this paper, we perform a quanti-

the OSS development process still need to be investigated. For example, unlike closed software, the OSS development process involves developers who irregularly participate in projects. Moreover, the roles of users in OSS may be different from those in closed software because they have more closer contact with developers. Understanding the collabo-

Given Document 5, the title should be "A Topological Analysis of the Open Source Software Development Community", the author(s) should be "Xu, Gao, Christley, Madey", the year of publication should be "2005", the author affiliation should be "Univeristy of Notre Dame", and the keywords should be unparsable.

2.5.6 Document 6

THE OPEN SOURCE SOFTWARE DEVELOPMENT PHENOMENON: AN ANALYSIS BASED ON SOCIAL NETWORK THEORY

Greg Madey
Computer Science & Engineering
University of Notre Dame
gmadey@nd.edu

Vincent Freeh
Computer Science & Engineering
University of Notre Dame
vin@nd.edu

Renee Tynan
Department of Management
University of Notre Dame
rtynan@nd.edu

Abstract

The OSS movement is a phenomenon that challenges many traditional theories in economics, software engineering, business strategy, and IT management. Thousands of software programmers are spending tremendous amounts of time and effort writing and debugging software, most often with no direct monetary compensation. The programs, some of which are extremely large and complex, are written without the benefit of traditional project management, change tracking, or error checking techniques. Since the programmers are working outside of a traditional organizational reward structure, accountability is an issue as well. A significant portion of internet e-commerce runs on OSS, and thus many firms have little choice but to trust mission-critical e-commerce systems to run on such software, requiring IT management to deal with new types of socio-technical problems. A better understanding of how the OSS community functions may help IT planners make more informed decisions and develop more effective strategies for using OSS software. We hypothesize that open source software development can be modeled as self-organizing, collaboration, social networks. We analyze structural data on over 39,000 open source projects hosted at SourceForge.net involving over 33,000 developers. We define two software developers to be connected — part of a collaboration social network — if they are members of the same project, or are connected by a chain of connected developers. Project sizes, developer project participation, and clusters of connected developers are analyzed. We find evidence to support our hypothesis, primarily in the presence of power-law relationships on project sizes (number of developers per project), project membership (number of projects joined by a developer), and cluster sizes. Potential implications for IT researchers, IT managers, and governmental policy makers are discussed.

Keywords: Open source software, social networks

Introduction

The global E-Commerce infrastructure relies heavily on open source software (OSS). Examples include web-servers (Apache, iPlanet/Netscape), e-mail servers (Sendmail), languages (Perl, Java, Python, GCC, Tk/TCL), and operating systems (Linux, BSD Unix). Corporate IT managers and decision makers have traditionally depended on either internally developed systems or commercially purchased systems that have been mostly closed source and proprietary. Open source software is written and supported by programmers, many coming from the “hacker culture”. This development culture includes hundreds of thousands of distributed programmers voluntarily producing, sharing, and supporting their software with no monetary compensation for their efforts. This presents those corporate IT managers and decision makers with a new development process and culture to learn and

1806 2002 — Eighth Americas Conference on Information Systems

Given Document 6, the title should be “Open Source Software Development Phenomenon: Analysis Based on Social Network Theory”, the author(s) should be “Madey, Freeh, Tynan”, the year of publication should be “2002”, the author affiliation should be “University of Notre Dame”, and the keywords should be “Open source software, social networks”.

2.5.7.1 Document 7.1

The social structure of Open Source Software development teams

Kevin Crowston and James Howison

School of Information Studies
Syracuse University

4-206 Centre for Science and Technology
Syracuse, NY 13244-4100

+1 (315) 443-1676

Fax: +1 (866) 265-7407

Email: crowston@syr.edu, james@freelancepropaganda.com

Submission to the *Individuals, Teams, and Virtual Communities* Track
of the 2003 *International Conference on Information Systems*

Draft of 5 May 2003

2.5.7.2 Document 7.2

The social structure of Open Source Software development teams

Abstract

Open Source Software development teams provide an interesting and convenient setting for studying distributed work. We begin by answering perhaps the most basic question: what is the social structure of these teams? Based on a social network analysis of interactions represented in 62,110 bug reports from 122 large and active projects, we find that some OSS teams are highly centralized, but contrary to expectation, others are not. Furthermore, we find that the level of centralization is negatively correlated with project size, suggesting that larger projects become more modular. The paper makes a further methodological contribution by identifying appropriate analysis approaches for interaction data. We conclude by sketching directions for future research.

Completed Research Paper submission
to the *Individuals, Teams, and Virtual Communities Track*
of the *International Conference on Information Systems*

5813 words, 22 pages (1 title/abstract + 20 text)

Keywords: Open Source Software, software development, bug fixing, distributed work, social network analysis

Given Documents 7.1-2, the title should be “The social structure of Open Source Software development teams”, the author(s) should be “Crowston, Howison”, the year of publication should be “2003”, the author affiliation should be “Syracuse University, School of Information Studies”, and the keywords should be “Open Source Software, software development, bug fixing, distributed work, social network analysis”.

focus guest editors' introduction

Global Software Development

James D. Herbsleb and Deependra Moitra, *Lucent Technologies*

The last several decades have witnessed a steady, irreversible trend toward the globalization of business, and of software-intensive high-technology businesses in particular. Economic forces are relentlessly turning national markets into global markets and spawning new forms of competition and cooperation that reach across national boundaries. This change is having a profound impact not only on marketing and distribution but also on the way products are conceived, designed, constructed, tested, and delivered to customers.¹

Software engineers have recognized the profound influence of business globalization for some time, generating alarmist reactions

in some quarters and moving others to try to capture and emulate models that have met with success. More recently, attention has turned toward trying to understand the factors that enable multinationals and virtual corporations to operate successfully across geographic and cultural boundaries.

Globalization and software

Over the last few years, software has become a vital component of almost every business. Success increasingly depends on using software as a competitive weapon. More than a decade ago, seeking lower costs and access to skilled resources, many organizations began to experiment with remotely located software development facilities and with outsourcing. Several factors have accelerated this trend:

- the need to capitalize on the global resource pool to successfully and cost-competitively use scarce resources, wherever located;



Given Document 8, the title should be “Global Software Development”, the author(s) should be “Herbsleb, Moitra”, the year of publication should be “2001”, the author affiliation should be “Lucent Technologies”, and the keywords should be unparsable.

2.5.9 Document 9



WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC

S.Balaji

Computer Science Dept., Gulf College
Muscat, Sultanate of Oman.

Dr.M.Sundararajan Murugaiyan

Computer Science Dept., Government Arts College
Chennai, TN, India.

Email: sundramoorthybalaji@gmail.com

Abstract: Organizations that are developing software solution are faced with the difficult choice of picking the right software development life cycle (SDLC). The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The V-model represents a software development process which may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape Agile Modeling is a practice-based methodology for modelling and documentation of software-based systems. It is intended to be a collection of values, principles, and practices for modelling software that can be applied on a software development project in a more flexible manner than traditional Modelling methods. This comparative summarizes the steps an organization would have to go through in order to make the best possible choice.

Keywords: SDLC, Waterfall, V-Model, Agile

1. INTRODUCTION

A Software Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, and are explained in the section below. A number of software development life cycle (SDLC) models have been created: waterfall, spiral, V-Model, rapid prototyping, incremental, and

2. Q&A BEFORE DECIDING WHICH MODEL TO BE USED

Before deciding the model to be used, we should get answer for some questions.

1. How stable are the requirements?

Given Document 9, the title should be “Waterfall vs V-Model vs Agile: A Comparative Study on SDLC”, the author(s) should be “Balaji, Murugaiyan”, the year of publication should be “2012”, the author affiliation should be “Gulf College, Government Arts College”, and the keywords should be “SDLC, Waterfall, V-Model, Agile”.

Agile Software Development: The People Factor

Alistair Cockburn, Humans and Technology
Jim Highsmith, Cutter Consortium

In a previous article (“Agile Software Development: The Business of Innovation,” *Computer*, Sept. 2001, pp. 120-122), we introduced agile software development through the problem it addresses and the way in which it addresses the problem. Here, we describe the effects of working in an agile style.

AGILE RECAPPED

Over recent decades, while market forces, systems requirements, implementation technology, and project staff were changing at a steadily increasing rate, a different development style showed its advantages over the traditional one. This *agile* style of development directly addresses the problems of rapid change.

A dominant idea in agile development is that the team can be more effective in responding to change if it can

- reduce the cost of moving information between people, and
- reduce the elapsed time between making a decision to seeing the consequences of that decision.

To reduce the cost of moving information between people, the agile team works to

- place people physically closer,
- replace documents with talking in person and at whiteboards, and
- improve the team’s amicability—its sense of community and morale—so that people are more inclined to relay valuable information quickly.



Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams.

To reduce the time from decision to feedback, the agile team

- makes user experts available to the team or, even better, part of the team and
- works incrementally.

Making user experts available as part of the team gives developers rapid feedback on the implications to the user of their design choices. The user experts, seeing the growing software in its earliest stages, learn both what the developers misunderstood and also which of their requests do not work as well in practice as they had thought.

The term *agile*, coined by a group of people experienced in developing software this way, has two distinct connotations.

The first is the idea that the business and technology worlds have become turbulent, high speed, and uncertain, requiring a process to both create change and respond rapidly to change.

The first connotation implies the second one: An agile process requires responsive people and organizations. Agile development focuses on the talents

and skills of individuals and molds process to specific people and teams, not the other way around.

AGILE IS FOR PEOPLE

The most important implication to managers working in the agile manner is that it places more emphasis on people factors in the project: amicability, talent, skill, and communication.

These qualities become a primary concern for the would-be agile team. Skill development is important, so that each person can deliver more value with time.

The attention to the human issues gives agile projects a particular feel. It is

not simply a matter of reducing paperwork and hoping that everything else will fall into place. To paraphrase one developer, “A small, rigorous methodology may *look* the same as an agile methodology, but it won’t *feel* the same.”

The agile group concept grows to span teams, organizations, and other working relationships. It is very difficult for agile people to function well in a rigid organization—and vice versa.

INDIVIDUAL COMPETENCE

Agile development teams focus on individual competency as a critical factor in project success.

If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy—“people trump process” is one way to say this.

However, lack of user and executive support can kill a project—“politics trump people.” Inadequate support can keep even good people from accomplishing the job.

The “CHAOS Chronicles Version II,” a recent update of the Chaos Report

November 2001

131

Given Document 10, the title should be “Agile Software Development: The People Factor”, the author(s) should be “Cockburn, Highsmith”, the year of publication should be “2001”, the author affiliation should be “Humans and Technology, Cutter Consortium”, and the keywords should be unparsable.

3. Glossary

Method - Equivalent to a "function" for the purposes of this document

Scholarly Articles – Informative documents intended specifically for an academic audience.

Cyclomatic Complexity – A measure of how the running time of a program scales in response to arbitrarily large input size.

4. References

- Balaji, S., and M. Sundararajan Murugaiyan. "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC." *International Journal of Information Technology and Business Management* 2.1 (2012): 26-30.
- Boehm, Barry W. "A spiral model of software development and enhancement." *Computer* 21.5 (1988): 61-72.
- Cockburn, Alistair, and Jim Highsmith. "Agile software development, the people factor." *Computer* 34.11 (2001): 131-133.
- Crowston, Kevin. *The social structure of open source software development teams*. Diss. Syracuse University, 2003.
- Herbsleb, James D., and Deependra Moitra. "Global software development." *IEEE software* 18.2 (2001): 16-20.
- Madey, Gregory, Vincent Freeh, and Renee Tynan. "The open source software development phenomenon: An analysis based on social network theory." *AMCIS 2002 Proceedings* (2002): 247.
- Mayhew, Deborah J. "The usability engineering lifecycle." CHI'99 Extended Abstracts on Human Factors in Computing Systems. ACM, 1999.
- Roach, Christopher. "Understanding Software Development Through Networks."
- Royce, Winston W. "Managing the development of large software systems." proceedings of IEEE WESCON. Vol. 26. No. 8. 1970.
- Xu, Jin, et al. "A topological analysis of the open source software development community." *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*. IEEE, 2005.

5. Journal Entries

Jon Mercer

2/6/2017

Requirements Journal

1/23/2017:

-First introduction given to project contents. Need to parse and extract title, list of authors, etc. from large number of PDF files. Expected to rename the files based on the data parsed. Mendeley and Zotero both stated as useful tools in this regard.

2/1/2017:

-Unable to verify Zotero has the "PDF parsing" functionality, and Mendeley costs money. Met with Stephanie before class to attempt to find a working application that can parse PDF data as described a week ago, could not find a working application, but a few libraries.

Stated in class that our requirements document does not need to be based off a specific working application.

2/6/2017:

-Worked with Stephanie on Requirements document before class, requirements document due date pushed back to Wednesday, just need to append journals and clean up some unused sections that were part of the original document template.

2/8/2017:

-Removed several members of the maintenance requirements, expanded glossary, removed blank appendices section from requirements document, corrected several small errors in the document, turned in the document

Stephanie Below

Entries to date: 2-13-17

DATE	Time	Description
Unknown	Unknown	Previous preliminary work done before the start of my journal includes: drafting up requirements for the PDF Renaming Tool and speaking with Jon Mercer via email to compare results and combine them to form a uniform set of requirements.
2-1-17	6:30pm to 7:35pm	Began downloading and looking through various source code from both iText and PDFbox
2-3-17	7:10pm to 8:00pm	Looked through various GUI Interface examples such as the AD3 Mp3 Renaming tool and began to think of ways to create a GUI Interface for the PDF Renaming tool.
2-4-17	3:30pm to 4:15pm	Began creating a GUI Interface for the PDF Renaming tool. Although the interface did not look at all like I wanted it to so I deleted it and decided to start over.
2-6-17	12:00pm to 1:30pm	Went back and looked at the Mp3 GUI interface and created an interface example through Microsoft Word. Awaiting approval or revision by Jon Mercer.
2-6-17	4:30pm to 6:30pm	Met with Jon Mercer and worked on the requirements document. We combined our journals, documented what should be accomplished at the end of the testing phase and cited each PDF document we used. The GUI interface was also revised.
2-6-17	6:45pm to 7:00pm	Revised the Requirements Document and changed the formatting. Printed of the document as well as my current journal entries for class.
2-9-17	1:45pm to 3:45pm	Added a cover page to the requirements document and revised the document to fit a constant format. Added the journal entries to the document and save it off as a PDF to turn in during class.
2-11-17	10:15pm to 11:45pm	Jon Mercer emailed me 4 more PDF links and I went through and copied the pages into the requirements document and cited them and included what information should be extracted.
2-13-17	10:00am to 10:50am	Found 3 more PDF documents and added them to the Requirements Document.

