Software Technology Maturation and Software Security

by

Trina Kay Locklear

July 2019

Director of Thesis: Dr. Nasseh Tabrizi, PhD

Major Department: Software Engineering

Abstract: Software technology maturation, also referred to as technology transfer, is as difficult as it is rare, mostly because of the time scale involved. Software maturation is defined as the process of taking a piece of technology from conception to popularization. Frequently, software engineers and developers tend to oversimplify the problems of technology transfer. They attribute problems to management pressures that complicate the use of software-engineering practices. However, a good understanding of the processes and problems is necessary to effectively tackle the technology-transfer problem. Without that understanding, the transfer of inappropriate technology to an organization without the maturity to understand and absorb it is likely to do harm, rather than to bring benefits. This research aims to answer two research questions regarding the technology maturation. Namely, is Redwine and Riddle's *"Software Technology Maturation"* study the accepted and gold standard within the software engineering discipline for assessing the maturation of software technology? Secondly, can the software technology maturation study be applied to other areas of software technology? The purpose of this research is to answer these

questions of interest which will serve as the basis for the second implementation; applying the Redwine and Riddle criteria to the comparatively young discipline of software security. The primary goal for the second implementation is to explore and extend the second research question and demonstrate the maturity phases for the field of software security.

Software Technology Maturation and Software Security

A Thesis

Presented to The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Trina Kay Locklear

July 2019

Software Technology Maturation and Software Security

by

Trina Kay Locklear

APPROVED BY:

DIRECTOR OF THESIS: ______________________________

Nasseh Tabrizi, PhD

COMMITTEE MEMBER: ______________________________

Mark Hills, PhD

COMMITTEE MEMBER: ______________________________

Sergey Vilkomir, PhD

CHAIR OF DEPARTMENT ______________________________

OF COMPUTER SCIENCE Venkat Gudivada, PhD

DEAN OF THE ______________________________

GRADUATE SCHOOL Paul J. Gemperline, PhD

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Software engineering was first introduced at a NATO conference in 1968. Since that time, it has come a long way [1]. The same can be said for software technology maturation. Software technology maturation, also referred to as technology transfer, is as difficult as it is rare, mostly because of the time scale involved [6]. Defined as the process of taking a piece of technology from conception to widespread use [2], technology maturation was the impetus for the planning of the Department of Defense Software Initiative. It was also the cause of concern at the 1984 IEEE Workshop on Software Technology Transfer. Furthermore, it was the motivation for the United States Defense Departments 1984 establishment of the Software Engineering Institute (SEI) [3] [4], because during this time, it was well-known that incipient software technology was not coming to market fast enough to equal the expansive, complex and sizeable defense software systems. Regardless of the bottlenecks in order to remain competitive, time-to-market was imperative then and still is today [4]. Noted most recently in the emerging cybersecurity community, the demands of software marketing tend to dominate most correctness concerns [5]. Software engineers and developers tend to oversimplify the problems of technology transfer. They attribute problems to management pressures that complicate the use of software-engineering practices. However, a good understanding of the processes and problems is necessary to effectively tackle the technology-transfer problem [3]. Without that understanding, the transfer of inappropriate technology to an organization without the maturity to understand and absorb it is likely to do harm, rather than to bring benefits [7].

There has been limited success in technology transfer as some new ideas take hold immediately, but more times than none, a novel, proven idea takes many years to become accepted as standard practice [1]. Yet, it is this process of software maturation, to full propagation, that is at the heart of this study. This research aims to answer three research questions regarding the technology maturation. Namely,

- ***RQ1:*** *Is Redwine and Riddle's "Software Technology Maturation" study the accepted gold standard within the software engineering discipline for assessing the maturation of software technology?*
- ***RQ2:*** *Can the software technology maturation model be applied to current areas of software technology?*

The purpose of this research is to answer these questions of interest which will serve as the basis for the second implementation; applying the Redwine and Riddle maturation phases to the comparatively young discipline of software security.

Research Contribution

This study features two original research contributions: a current synthesis of literature concerning the treatment of Redwine and Riddle's proposed study of software technology maturity in peer reviewed articles, as well as an application of Redwine and Riddle's maturation phases to the discipline of software security.

CHAPTER 2

RELATED WORK

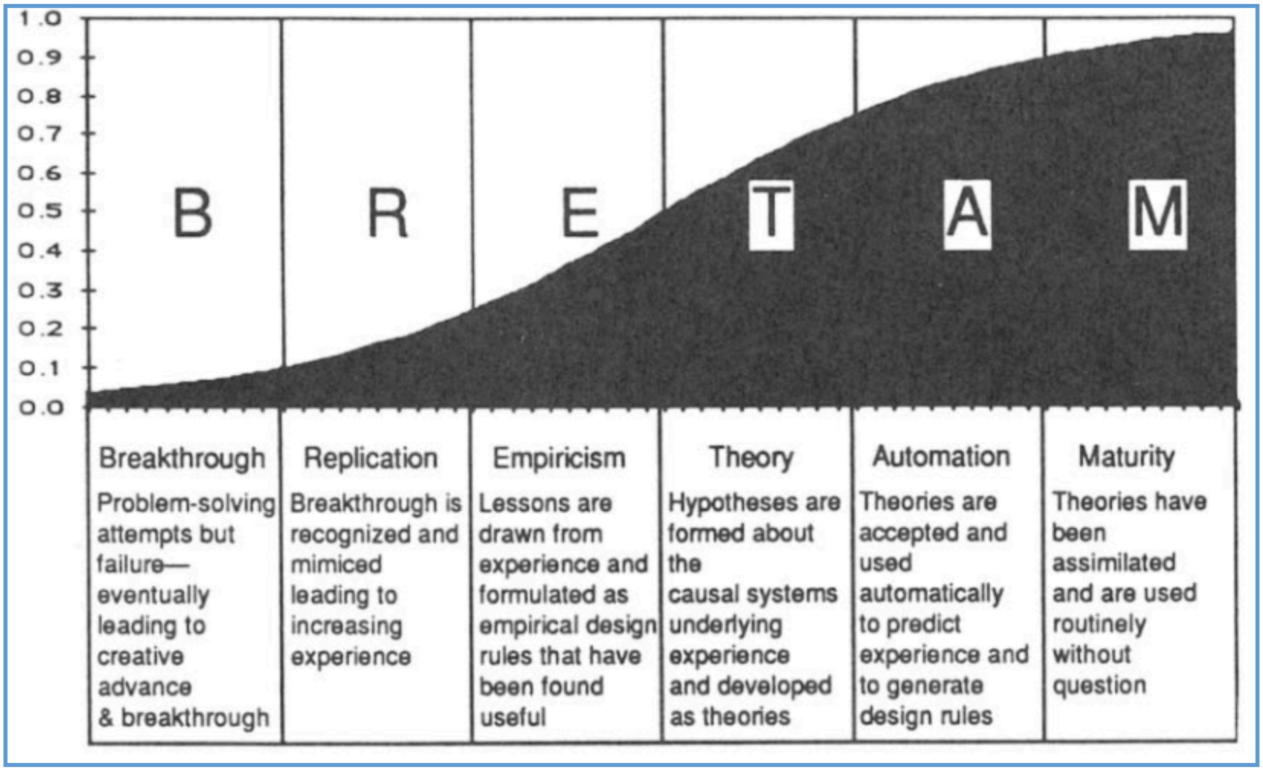
2.1 Software Technology Maturation Model

The Redwine and Riddle maturation model of 1985 is one of the first [1] models describing the software technology maturation process. Currently, "Software Technology Maturation" by Redwine and Riddle is one of the most cited models regarding technology maturation as well (ACM - 57 cite count, Google Scholar - 242 cite count). However, there are other works, with slight variations, that also articulate maturation, which others refer to commonly as technology transfer or technology infusion. Below are a few existing literatures, in chronological order, that are related to software technology maturation.

- **Raghavan, Chand (1989)** - Raghavan and Chand refer to technology maturation or transfer as diffusion, 'the process of transferring technology from those who develop it to those who apply it.' They propose a less linear alternative life cycle than Redwine and Riddle. Direct supports of the E. Rogers framework, Raghavan and Chand conduct and informal case studies trying to specialize Roger's framework. Their conclusions, detailed as practitioner's problems, and communication problems are similar to the findings of Redwine and Riddles study regarding critical factors, inhibitors, and facilitators. (IEEE - 36 cite count, Google Scholar - 94 cite count) [3].
- **Malcolm (1991)** - In 1991, J. N. Buxton and R. Malcolm authored "Software Technology Transfer." in which a generic model for technology transfer is proposed. Unlike Redwine and Riddle, who studied various technologies, Buxton and Malcolm demonstrate industrial

circumstances regarding the transfer process due to their stance that phases in an industrial setting is different. However, like Redwine and Riddle's inhibitors, Buxton and Malcolm discuss in detail the barriers to innovation. *Buxton and Malcolm's phases, known as phases in innovation, are* research*, evaluation of technical feasibility, evaluation of economic feasibility, adoption, maturation, and old age*. Their community roles of technology transfer are supplier, gatekeeper, top management, middle management, and educators. While Redwine and Riddle do mention various roles in the maturation process, Buxton and Malcolm specify in-depth the duties and responsibilities of each role within the technology transfer process. (IEEE - 6 cite count, Google Scholar - 32 cite count.) [7].

- **Gaines (1991)** - Brian Gaines proposes the BRETAM model. As shown in Figure 1 below, his phases, which he refers to as learning curves are: breakthrough (inventor makes a breakthrough), replicator (work is replicated at research institutions worldwide), empiricist (empirical design rules), theory (model the basis of success and failure and develop theories), automation (theoretical models make it possible to automate data gathering and analysis with the manufacturing processes), maturity (after automation, focus is placed on cost reduction and quality improvements in the mature technology). This model is used to account for past events as well as forecast future trends. His framework is based on logical progression of developments. (Google Scholar - 60 cite count) [105].



Breakthrough	Replication	Empiricism	Theory	Automation	Maturity
Problem-solving attempts but failure—eventually leading to creative advance & breakthrough	Breakthrough is recognized and mimiced leading to increasing experience	Lessons are drawn from experience and formulated as empirical design rules that have been found useful	Hypotheses are formed about the causal systems underlying experience and developed as theories	Theories are accepted and used automatically to predict experience and to generate design rules	Theories have been assimilated and are used routinely without question

Figure 1. BRETAM model. The y-axis indicates zero knowledge to complete knowledge [105].

- **Zelkowitz (1995)** - Zelkowitz evaluated engineering technology transfer at the National Aeronautics and Space Administration, NASA. In contrast to Redwine and Riddle, Zelkowitz's approach was to focus on specific problems faced by NASA and how new technologies addressed these issues. Also, Zelkowitz was clear to distinguish between technology transfer and infusion. For Zelkowitz, technology transfer was the insertion of new technology into an organization previously performing assignments and infusion was incorporation of new technology that had previously used nothing similar to the new technology. Zelkowitz also makes the distinction between a technology producer versus a consumer. ***Zelkowitz identifies 5 models to encourage transfer of technology: people-mover model, communication model, on-the-shelf model, vendor model and finally a rule model.*** Like Redwine and Riddle, Zelkowitz describes mechanisms that encourage technology transfer but in NASA. (Google Scholar - 12 cite count) [106].

- **Rogers (1995)** - Sociologist and management scientists have studied technology transfer and diffusion extensively and Everett Roger's framework is their primary literature of study. His framework has had great success and was widely accepted because it successfully predicted and explained the diffusion process for a broad variety of innovations. Unlike Redwine and Riddle focus of technologies, Rogers' study of technology transfer is based extensively on the study of agricultural innovations and organizations; including those not related to software. In Figure 2, ***Rogers phases or elements are innovation, the communication process, the adoption process and the social system.*** Rogers notes distinct patterns and speed with how technology is adopted. His distinctions are innovators, early adopters, early majority, late majority and laggards with the first adaptors being the innovators.

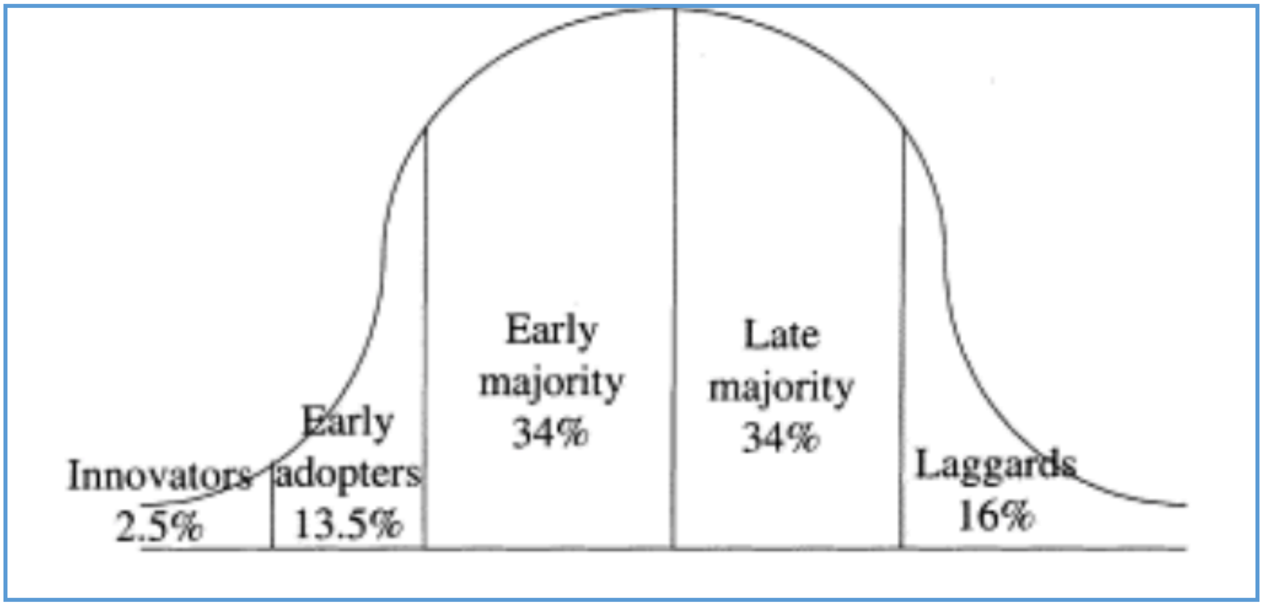


Figure 2. Roles and patterns in how technology is adopted [107].

Rogers explains in depth various characteristics of each grouping. Rogers' model correlates loosely to Zelkowitz in that different adopters use different styles. However, several of Rogers guidelines overlap with Redwine and Riddles inhibitors and facilitators. (Google Scholar - 20 cite count) [107].

- **Pfleeger (1999)** - In Pfleeger's model, she uses the terms, technology transfer and technology infusion interchangeably unlike Zelkowitz who distinguishes technology transfer from infusion. Pfleeger's work focuses on suggesting ways to help practitioners and researchers understand how to shorten the time between innovation and effective. Pfleeger is very precise, a noted criticism of Redwine and Riddle, with her definition of a technology. Pfleeger initially highlights that Redwine and Riddle included processes, standards and products as technology while she defines technology as any method, technique, tool, procedure or paradigm used in software development or maintenance. Pfleeger utilizes Redwine and Riddles model, along with Rogers and Zelkowitz for how successful technology transfer might be attempted by development organizations by identifying five key activities. In Figure 3, Pfleeger's first activity is technology creation.

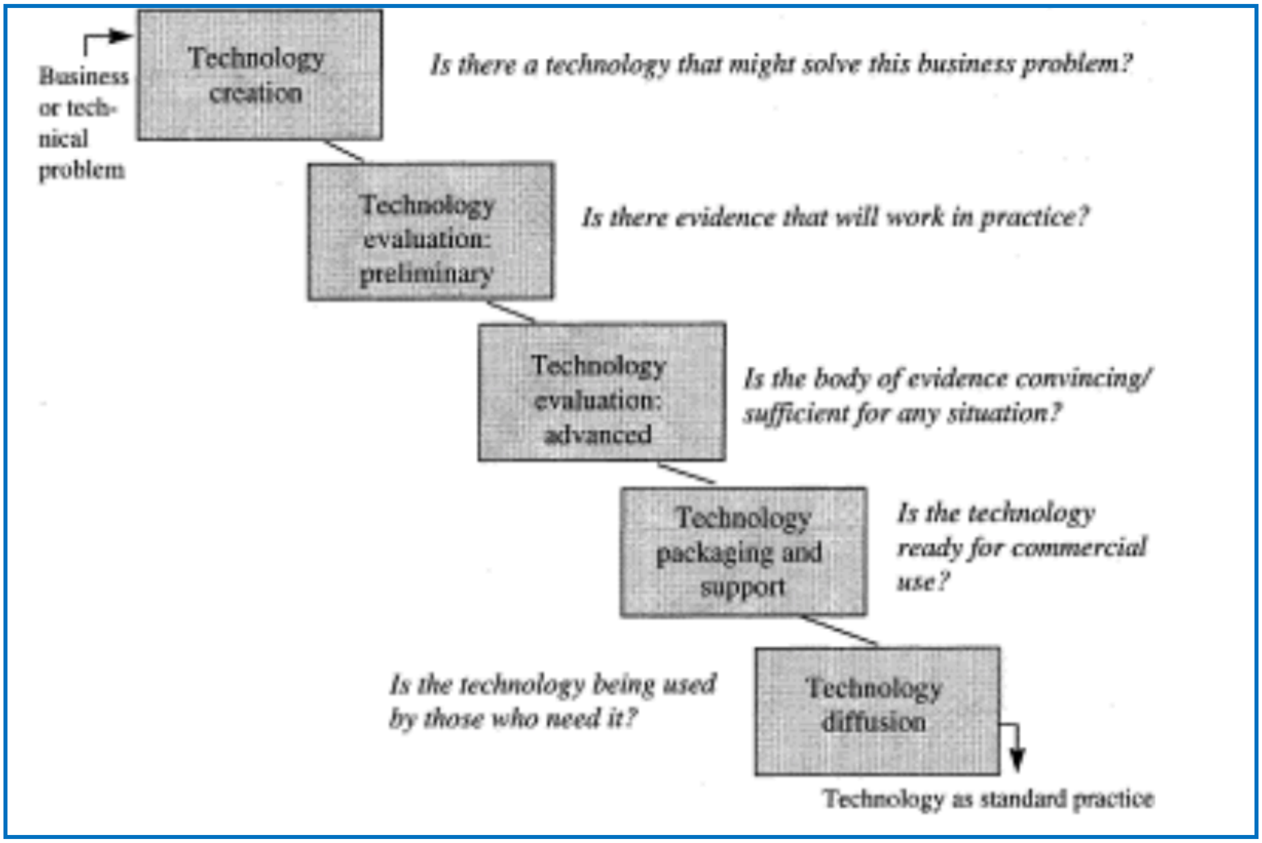

Business or technical problem
Technology creation
Is there a technology that might solve this business problem?
Technology evaluation: preliminary
Is there evidence that will work in practice?
Technology evaluation: advanced
Is the body of evidence convincing/ sufficient for any situation?
Technology packaging and support
Is the technology ready for commercial use?
Is the technology being used by those who need it?
Technology diffusion
Technology as standard practice

Figure 3. Steps from idea to standard practice. Pfleeger's model for successful transfer [1].

After the technology is created and found, the next step is preliminary investigation to determine whether there is evidence that a technology will work in practice. Evidenced it can work, the next step is more thorough evaluation of the body of evidence. With compelling evidence combined with commercially viable support the final step is promoting adoption with those who are likely to benefit from using the technology. (ACM - 35 cite count, Google Scholar - 139 cite count) [1].

2.2 Software Security Maturation Model Related Work

There does exist two studies that feature an in-depth study of the maturity of a software technology. These studies are previously discussed in the applications section from the previous implementation. In 2006, Shaw and Clements research and complete a maturation model for the discipline of software architecture. Their maturation model for software architecture demonstrates full maturity as well as an addition of a foundations phase [12]. The second maturation model is featured in a 2016 article surrounding human-centric design of information systems. In "The Impact of Human-Centric Design on the Adoption of Information Systems: A Case Study of the Spreadsheet," Scaffidi does not provide a graphical depiction of his case study of the spreadsheet. However, Scaffidi does discuss at depth numerous acme and milestones that signify the progression through phases to maturation. Scaffidi demonstrates each maturation point as illustrated in the original Redwine and Riddle study but the spreadsheet technology [81]. For example, for Basic Research, Scaffidi begins this phase with the highlighting Richard Mattessich as the inventor of the spreadsheet tool for budgetary resources. Furthermore, for the Enhancement and Exploration phase, Scaffidi points out that the first customer-oriented spreadsheet was created by Software Arts in 1979 which signifies the 'usable capability' feature applicable to that phase. Lastly, to show propagation, Scaffidi discusses patents as well as the 1989 statistic that 10% of American that used computers at work also used spreadsheets [81]. The third maturation model details the progression to maturity of the field of self-adaption from a 2017 publication, "Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges," D. Weyns implores a familiar graphical representation similar to the Shaw and Clements in the following diagram. For a diagram, please refer to the application section of the previous implementation.

CHAPTER 3

METHODOLOGY

The first research activity was to search and review the primary focus of this review, "*Software Technology Maturation*" by Redwine and Riddle. The second research activity was an intentional decision to include only those articles that cited the Redwine and Riddle study. There were two reasons for this approach. First, this approach renders the research repeatable [8]. Secondly, this approach features an unbiased [8] population of articles that cite the Redwine and Riddle maturity model and therefore provides a viewpoint of the authors utilization of the Redwine and Riddle model.

The most effective and efficient approach to researching the reputation of a model within the software engineering community is to research others treatment of the model. The inclusion criteria are a reference and citation to Redwine and Riddle's, "Software Technology Maturation" study. The exclusion criteria are no reference and no citation to Redwine and Riddle's, "Software Technology Maturation." It should be noted that some articles were not accessible or were duplicates and thus were not evaluated. However, these articles were included but were categorized as Errors. The searched databases include the ACM DL, IEEE Explore, Elsevier, Springer, Google Scholar, and ProQuest. Content published in scholarly journals, books, articles, workshops, and conference proceedings were included. Dissertations and thesis publications as well as self-published works were not included in this study.

3.1 Background: The Redwine and Riddle Study

In order to understand how others utilize and view Redwine and Riddle's maturity model, a thorough understanding of the original publication on software technology maturation seems logically consistent. In 1985, two 'well-known' computer scientists, Samuel T. Redwine Jr. at the Government's Institute for Defense Analysis and William E. Riddle at Software Design and Analysis, Inc., was curious about the length of time it took for newly formed ideas to become commonplace. Out of their 'curiosity' through 'rigorous' analysis, Redwine and Riddle published "Software Technology Maturation" [9]. The study researches the case studies of 17 software technologies in order to uncover similar characteristics of the maturation process. In the study, Redwine and Riddle define technology maturation as the process of taking a part of technology from conception to widespread use among professionals. Notably, the study prefaces that the amount of time necessary for technology maturation was longer than initially presumed. The primary interest of the report is in learning what could be done to speed up the maturation process in regard to attaining widespread use. (This aspect is probably due to their work with the defense arena and the issues previously stated surrounding inadequate technology propagation at the time.) The main subject of the study is maturation facilitators and inhibitors.

In the article, Redwine and Riddle describe technology transition as certain actions taken by a technology improvement program that move a part of technology to commercial use. These actions consist of packaging, intentional inclusion into influential arenas, and marketing and distribution. It is important to note that Redwine and Riddle differentiate technology maturation as the overarching process of technology development followed by technology transition.

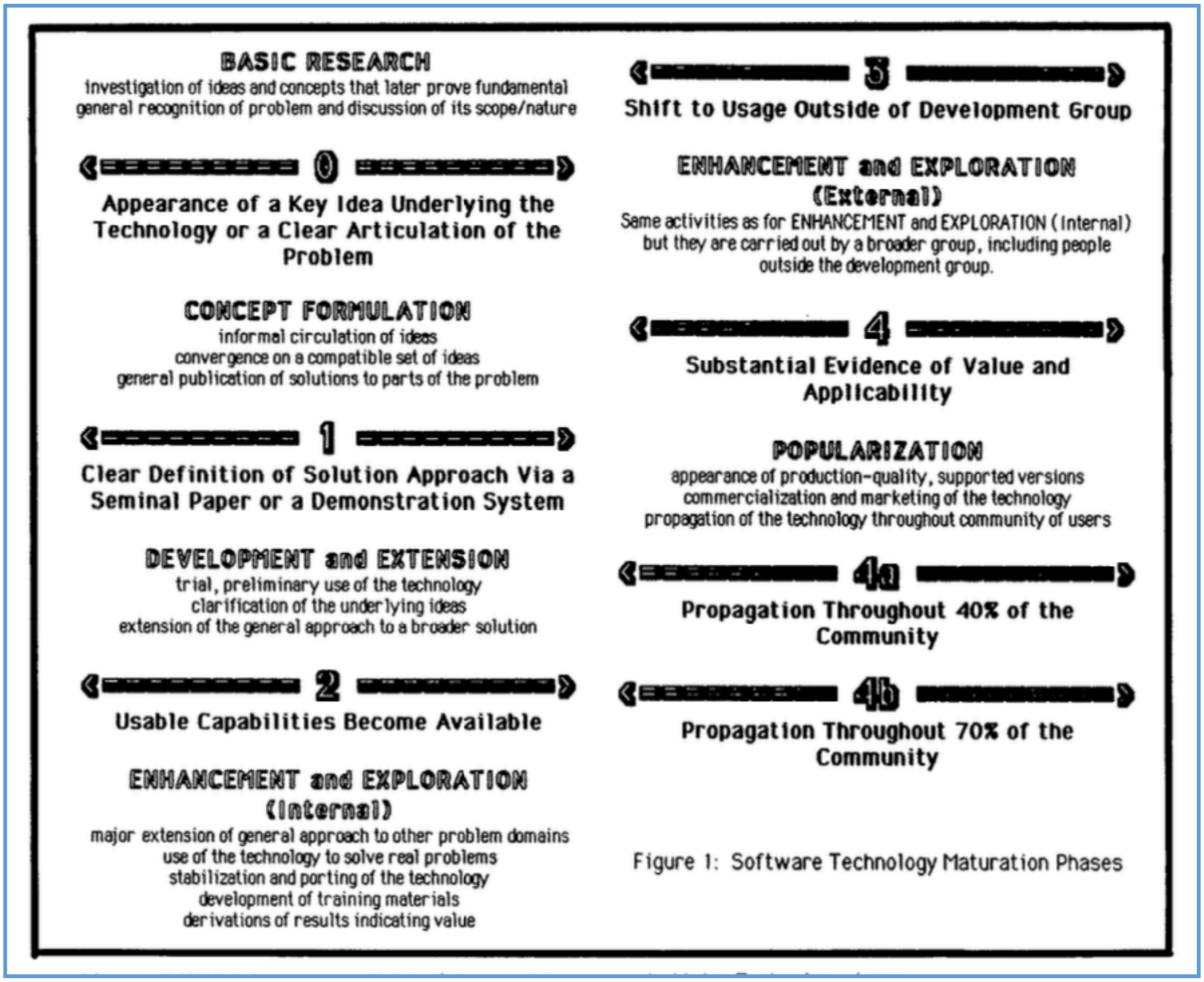
3.2 Four Types of Technologies

To assist them in the study of technology maturation, Redwine and Riddle review and analyze fourteen case studies performed by experts in their respective fields. The case studies are divided into four types and feature the following technologies:

- Major Technology Areas - knowledge-based systems, metrics, software engineering principles, compiler construction and formal verification.
- Technology Concepts - abstract data types (ADT) and structured programming.
- Methodology Technology - DOD-STD-SDS (Department of Defense Software Development Standard) - DOD software lifecycle model; AFR (Air Force Regulation) 800-14, software development/acquisition standards, and the SCR (Navy's Software Cost Reduction program) methodology.
- Consolidated Technology - cost models, automated software environments, Unix, Smalltalk-80, Software Requirements Engineering Methodology (SREM).

3.3 The Six Phases

In order to compare the case studies, the authors create a basic ordinal scale. As a key feature of the study, the scale describes the six main phases of software technology maturation by stationing time points that differentiate progression between the phases. The six phases are: Basic Research, Concept Formulation, Development and Extension, Internal Enhancement and Exploration, External Enhancement and Exploration, and Popularization. The authors make note that technology transition begins at time phase 2, Internal Enhancement and Exploration, which impacts technology maturation. The chart shown in Figure 4 is the depiction of the forward progressional ordinal scale of phases and the milestones and features between each phase.



BASIC RESEARCH
investigation of ideas and concepts that later prove fundamental
general recognition of problem and discussion of its scope/nature

0
Appearance of a Key Idea Underlying the Technology or a Clear Articulation of the Problem

CONCEPT FORMULATION
informal circulation of ideas
convergence on a compatible set of ideas
general publication of solutions to parts of the problem

1
Clear Definition of Solution Approach Via a Seminal Paper or a Demonstration System

DEVELOPMENT and EXTENSION
trial, preliminary use of the technology
clarification of the underlying ideas
extension of the general approach to a broader solution

2
Usable Capabilities Become Available

ENHANCEMENT and EXPLORATION (Internal)
major extension of general approach to other problem domains
use of the technology to solve real problems
stabilization and porting of the technology
development of training materials
derivations of results indicating value

3
Shift to Usage Outside of Development Group

ENHANCEMENT and EXPLORATION (External)
Same activities as for ENHANCEMENT and EXPLORATION (Internal) but they are carried out by a broader group, including people outside the development group.

4
Substantial Evidence of Value and Applicability

POPULARIZATION
appearance of production-quality, supported versions
commercialization and marketing of the technology
propagation of the technology throughout community of users

4a
Propagation Throughout 40% of the Community

4b
Propagation Throughout 70% of the Community

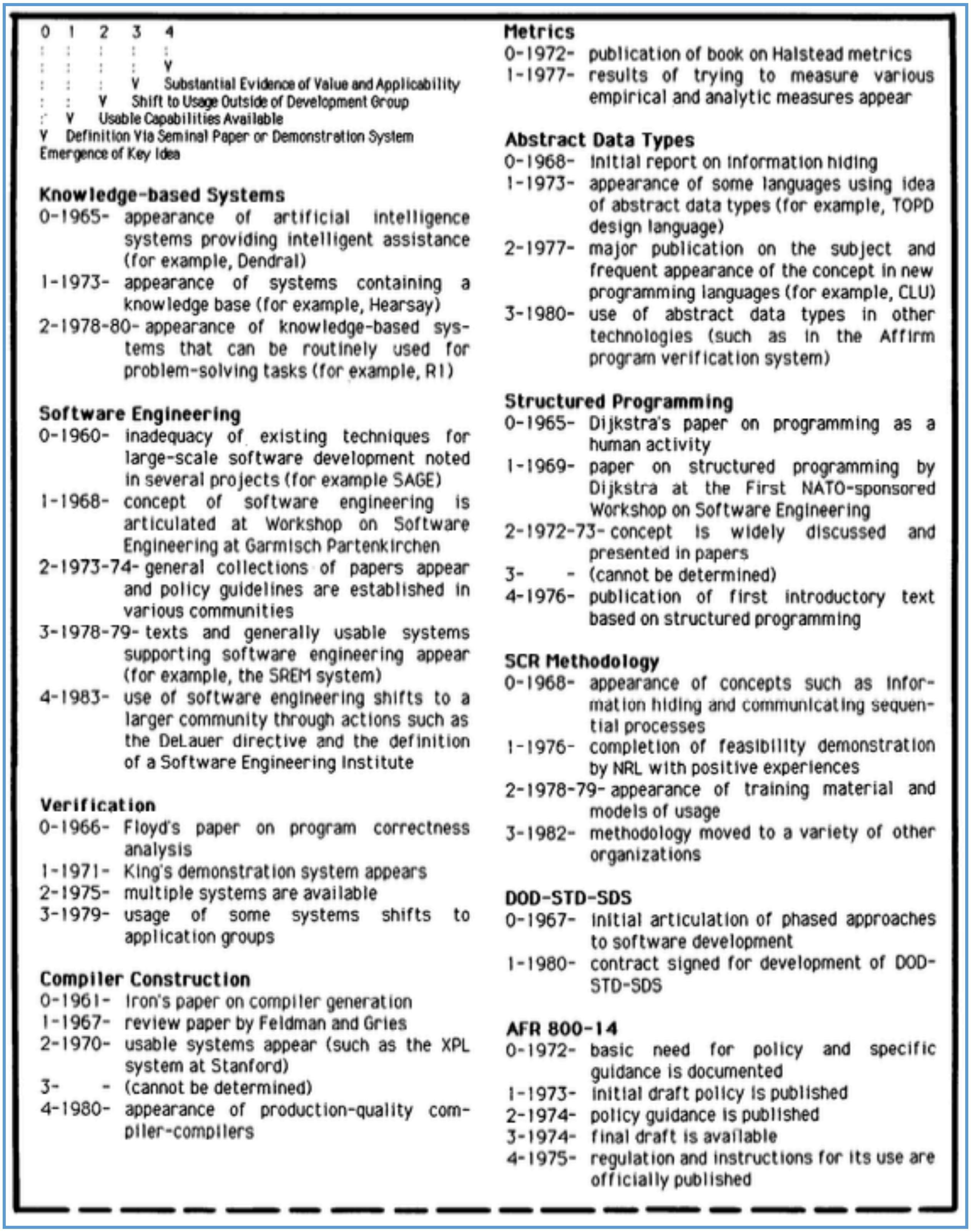
Figure 1: Software Technology Maturation Phases

Figure 4. The six phases of the software technology maturation process [2].

- Phase 1 is Basic Research. This phase which includes the investigation of ideas and concepts. The technology moves from Phase 1 to Phase 2 when there appears to be a key idea foundational to the technology or a clear statement of the problem.
- Phase 2 is Concept Formulation. Phase 2 is signified by an informal discussion of ideas or publication of a solution to portions of the problem. As stated previously, technology transition begins in Phase 2. Progression to Phase 3 begins with a clear solution presented in a paper or a demonstration.

- Phase 3 is Development and Extension. This phase involves a trial or initial use of the technology or clarification of the main primary ideas. Phase 3 may also include an expansion of the basic approach to an overall solution as a whole. A clear indication of advancement to Phase 4 is when operational functionalities are made available.
- Phase 4 is entitled Internal Enhancement and Exploration. Five activities indicate Phase 4. The approach to the solution is expanded into other domains, the technology is used to solve real world problems, the technology is stabilized and parted, training materials are created, and original results show value.
- Phase 5 is approached when the technology is used outside of the development group. Phase 5 is External Enhancement and Exploration. Phase five constitutes the same activities are the previous phase but by a broader group. If the technology demonstrates considerable evidence of value and validity, the technology is approaching the final phase.
- Phase 6 is Popularization. The technology looks production-quality, is commercialized and marketed, or the technology is widespread to a group of users. When dissemination has reached 40% and 70% of the users within the community these percentages represent milestones of the final phase.

The second key feature of the Redwine and Riddle study are time points. Each time a technology reached one of the six phases, Redwine and Riddle listed the year and the correlating event or activity. Figure 5 shows these significant time points for each technology, by phase, year, and event or activity.



```
0   1   2   3   4
:   :   :   :   :
:   :   :   :   V
:   :   :   V   Substantial Evidence of Value and Applicability
:   :   V   Shift to Usage Outside of Development Group
:   V   Usable Capabilities Available
V   Definition Via Seminal Paper or Demonstration System
Emergence of Key Idea
```

Knowledge-based Systems

0-1965- appearance of artificial intelligence systems providing intelligent assistance (for example, Dendral)

1-1973- appearance of systems containing a knowledge base (for example, Hearsay)

2-1978-80- appearance of knowledge-based systems that can be routinely used for problem-solving tasks (for example, R1)

Software Engineering

0-1960- inadequacy of existing techniques for large-scale software development noted in several projects (for example SAGE)

1-1968- concept of software engineering is articulated at Workshop on Software Engineering at Garmisch Partenkirchen

2-1973-74- general collections of papers appear and policy guidelines are established in various communities

3-1978-79- texts and generally usable systems supporting software engineering appear (for example, the SREM system)

4-1983- use of software engineering shifts to a larger community through actions such as the DeLauer directive and the definition of a Software Engineering Institute

Verification

0-1966- Floyd's paper on program correctness analysis

1-1971- King's demonstration system appears

2-1975- multiple systems are available

3-1979- usage of some systems shifts to application groups

Compiler Construction

0-1961- Iron's paper on compiler generation

1-1967- review paper by Feldman and Gries

2-1970- usable systems appear (such as the XPL system at Stanford)

3- - (cannot be determined)

4-1980- appearance of production-quality compiler-compilers

Metrics

0-1972- publication of book on Halstead metrics

1-1977- results of trying to measure various empirical and analytic measures appear

Abstract Data Types

0-1968- initial report on information hiding

1-1973- appearance of some languages using idea of abstract data types (for example, TOPD design language)

2-1977- major publication on the subject and frequent appearance of the concept in new programming languages (for example, CLU)

3-1980- use of abstract data types in other technologies (such as in the Affirm program verification system)

Structured Programming

0-1965- Dijkstra's paper on programming as a human activity

1-1969- paper on structured programming by Dijkstra at the First NATO-sponsored Workshop on Software Engineering

2-1972-73- concept is widely discussed and presented in papers

3- - (cannot be determined)

4-1976- publication of first introductory text based on structured programming

SCR Methodology

0-1968- appearance of concepts such as information hiding and communicating sequential processes

1-1976- completion of feasibility demonstration by NRL with positive experiences

2-1978-79- appearance of training material and models of usage

3-1982- methodology moved to a variety of other organizations

DOD-STD-SDS

0-1967- initial articulation of phased approaches to software development

1-1980- contract signed for development of DOD-STD-SDS

AFR 800-14

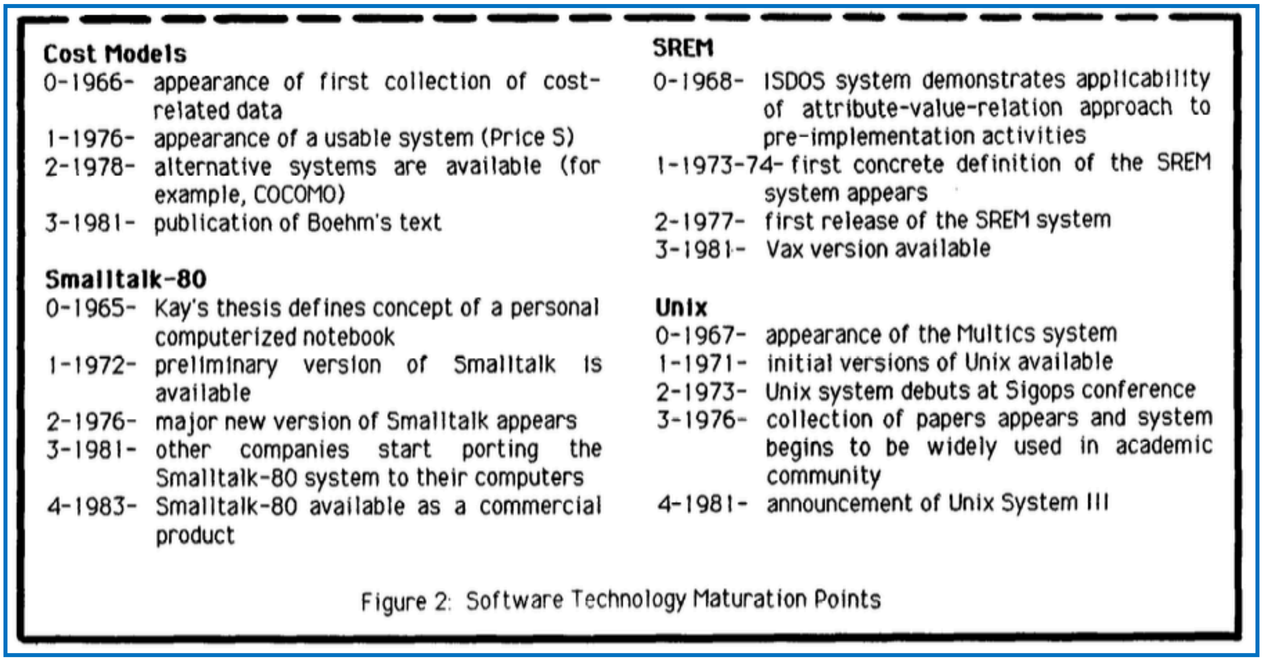
0-1972- basic need for policy and specific guidance is documented

1-1973- initial draft policy is published

2-1974- policy guidance is published

3-1974- final draft is available

4-1975- regulation and instructions for its use are officially published



Cost Models

0-1966- appearance of first collection of cost-related data

1-1976- appearance of a usable system (Price S)

2-1978- alternative systems are available (for example, COCOMO)

3-1981- publication of Boehm's text

Smalltalk-80

0-1965- Kay's thesis defines concept of a personal computerized notebook

1-1972- preliminary version of Smalltalk is available

2-1976- major new version of Smalltalk appears

3-1981- other companies start porting the Smalltalk-80 system to their computers

4-1983- Smalltalk-80 available as a commercial product

SREM

0-1968- ISDOS system demonstrates applicability of attribute-value-relation approach to pre-implementation activities

1-1973-74- first concrete definition of the SREM system appears

2-1977- first release of the SREM system

3-1981- Vax version available

Unix

0-1967- appearance of the Multics system

1-1971- initial versions of Unix available

2-1973- Unix system debuts at Sigops conference

3-1976- collection of papers appears and system begins to be widely used in academic community

4-1981- announcement of Unix System III

Figure 2: Software Technology Maturation Points

Figure 5. Yearly maturation time points [2].

Software Engineering, Compiler Construction, Structural Programming, AFR 800-14, Smalltalk-80 and Unix are all fully mature. Each technology progress to the final phase of popularization. Software Engineering achieved Phase 6 in 1983, Compiler Construction in 1975, Structural Programming in 1976, AFR 80-14 in 1975, Smalltalk-80 in 1983 and Unix in 1982. At the time of this publication in 1985, Verification, Abstract Data Types (ADT), Methodology, SREM and Cost Models were in Phase 5, External Enhancement and Exploration, of the maturity model. Knowledge Based Systems were the only technology in Phase 4. Metrics and DOD-STD-SOS were in Phase 3, Development and Extension.

3.4 Technology Observations

The study's general observations were that time varied in how long it took for a technology to mature from the second phase of Concept Formulation to the last phase of Popularization. Of the four types of technologies, the authors noted the following observations:

- Major Technology Area - maturation needed an extensive amount of time for two reasons. Due to this type's vast nature, specific parts of technology are wanted before a general advancement can occur in the area as a whole. Secondly, also, several major technologies were influenced by external forces.
- Technology Concepts - maturation occurred rather rapidly but not beyond the technical communities as fast as other technology based on them. This is anticipated as ideas can mature pretty quickly but it requires a specific technique, possibly supported by tools, for the majority of the technical community to use them.
- Methodology Technology - because this type concerns rules and guidelines governing other technology for the creation and evolution of software systems, several events must occur before this type can transition into popularization or widespread use. The reasons being that the core technology must first mature and secondly, the actual rules and guidelines must be developed.
- Consolidated Technology - this type is similar to the previous. Many things must come to fruition before a technology can fully mature. The Enhancement and Exploration phases for this type take longer than the Methodology Technology type probably because of the need to construct the adhesive that connects the pieces of technology together.

3.5 Factors of Maturation

Critical Factors

In the study, Redwine and Riddle admit that there are not enough case studies to determine the nominal case for technology maturation. Furthermore, the authors state it would be difficult if not impossible to predict the maturation time line for a technology by researching the time lines of other technologies. However, the case studies do suggest there are factors that can obstruct or assist the maturation of technology. In some cases, failure was shown when the following factors were not present. These factors are known as critical factors. These factors are critically necessary and trying to move toward widespread use is virtually pointless unless these factors are present.

- Conceptual Integrity - the technology must be thoroughly developed.
- Clear Recognition of Need - the technology must fulfil a clearly defined and well-recognized need.
- Tuneability - the technology must be pliable to the specific practices of a variety of user groups.
- Prior Positive Experience - readily available reports of previously positive experiences.
- Management Commitment - as stated, management must be committed and actively work to the introduction of new technology.
- Training - training on how to use the technology must be provided and the training should include numerous examples.

Inhibiting Factors

There are also factors that can obstruct or inhibit the technology maturation process. These factors slow down the maturation process versus forcing the process to a total standstill. These factors are:

- Internal Transfer - additional time to spread a technology throughout an organization.

- High Cost - the monetary or time costs to comprehend the technology must be reasonable.
- Contracting Disincentives - acquisition and contracting practices can slow the propagation of technology.
- Psychological Hurdles - many practitioners feel threatened by changing processes they have been capably performing for years.
- Easily Modified Technology - if a technology is easily modified, its introduction will be slowed down because it will be altered.

Facilitating Factors

Technology will propagate quickly when the previous inhibiting factors are nonexistent. Conversely, there are several factors that can accelerate the spread of technology. These factors are:

- Prior Success - a good history of success for the technology's creators will make it easy to sell and others will seek out a technology when they recognize an expert's new developments.
- Incentives - contracts can stipulate that a new technology be used.
- Technically Astute Managers - adoption of a new technology moved quicker when decision-makers were knowledgeable in modern software technology.
- Readily Available Help - well-versed staff can assist in explaining and selling the new technology.
- Latent Demand - if there is a recognizable crucial need, then the technology adoption is virtually instantaneous.

- Simplicity - although technology can be complex, adoption will move more surely and smoothly if the moments of it that are available for use easy to understand and minimally disruptive in practice.
- Incremental Extensions to Current Technology - technology that is an incremental enhancement of previous technology will be adopted fairly quickly.

3.6 Redwine and Riddle's Key Conclusions

In the study, Redwine and Riddle demonstrate:

- that it takes on the order of 15 to 20 years to mature a technology to widespread use to the technical community as a whole;
- the lengthiest scenario required 23 years to go from basic research to popularization;
- the shortest scenario required 11 years;
- the overall average 17 years;
- it took 7.5 years to go from a developed technology to popularization [10]. The study also suggested that the least amount of time needed for Phase 4, Internal Enhancement and Exploration was 3 years, and the average was between the low of 3.8 years and the high of 5 years.

Technology will not transition into widespread use without a recognized need, receptive community, believable demonstrations of cost/benefit, clearly defined attention and support, and an articulate advocate. The best process for technology transition is incremental expansion in small steps with trial use and the careful collecting of empirical evidence regarding the technology's value. Technology transition is inhibited by making small, simple mistakes that were corrected once identified. Case studies also demonstrated that technology transition is assisted by actions that improve the context in which the technology is taking place. There are several factors that can affect the speed at which technology matures and disseminates. Regardless of not providing the basic context or making mistakes, technology will take a lengthy time to mature. The degree to which the maturation can be accelerated appears limited, but there are numerous actions related to context that can be used to speed up technology maturation.

CHAPTER 4

IMPLEMENTATON

The articles that cite “Software Technology Maturation” by Redwine and Riddle are listed in Table 1. Each article was reviewed and evaluated based upon a series of questions. These questions served as the points of interest. The five points of interest make up the five main categories. The five categories are: Year (of publication), Primary Category, Opinion, Primary Source, and Alternative Source. Year (of publication) is tracked to determine the relevance of the study. Publication dates signify when the model is cited and incorporated into publications among authors. Primary Category demonstrates how the model was utilized in the article. Primary Category is divided into four subcategories: Application, Direct Reference, Indirect Reference, or Error. The Opinion category tracks the treatment of the model by the author in the article. The Primary Source category demonstrates how significant the model is in the article. The Alternative Source category demonstrates if the author proposed an alternative model. Each article was reviewed and categorized as shown in the following table. The results of each articles evaluation are summarized into the following table.

Number - The ID number of the article.

Year - Publication date of the article.

Hypertext - Link to article.

Title - Tile of the article.

Primary Treatment - How is the Redwine and Riddle study used in the article? (Options are:)

- App - Applied the Redwine and Riddle model to a technology.
- Direct - Direct reference to the authors name or the study's content.
- Indirect - Indirect reference to the study's content.
- Error - Error regarding the article. The article was a duplicate or is not accessible.

Opinion - Did the author(s) affirm or oppose the Redwine and Riddle study? (Options are:)

- Affirm - Affirmed the study directly or indirectly regarding technology maturation.
- Oppose - Opposed the study directly or indirectly regarding technology maturation.

Primary Source - Did the author(s) use the Redwine and Riddle study as the primary source regarding technology maturation? (Options are:)

- Yes
- No

Alternative Source - Did the author(s) utilize an alternative process for technology maturation? (Options are:)

- Yes
- No

Table 1. Assessment and evaluation of articles that cite Redwine and Riddle's study.

Number	**Year**	**Title**	**Primary Treatment**				**Opinion**		**Primary Source**		**Alternative Source**	
			App	**Direct**	**Indirect**	**Error**	**Affirm**	**Oppose**	**Yes**	**No**	**Yes**	**No**
1	1989	"Editor's Corner: How About Next Year? A Look at a Study of Technology Maturation" [9]		•			•		•			•
2	2019	"Software Project Management in High Maturity: Systematic Literature Mapping" [11]	•	•			•		•			•
3	2006	"The Golden Age of Software Architecture" [12]	•	•			•		•			•

4	2000	“Marketing Technology to Software Practitioners” [13[		•			•		•			•
5	2005	“An Empirical Study of Programming Language Trends” [14]			•		•		•			•
6	2000	“Software Engineering: A Roadmap” [15]			•		•		•			•
7	2014	“Ready-Set-Transfer: Exploring the Technology Transfer Readiness of Academic Research Projects” [16]		•			•		•			•
8	2002	“Software Engineering. Technology Watch” [17]		•			•			•	•	
9	2006	“Intelligent Decision Support for Road Mapping a Technology Transfer Case Study with Seimens Corporate Technology” [18]		•				•		•	•	
10	1987	“An Experiment in Technology Transfer: PAISLey Specification of Requirements for an Undersea Lightwave Cable System” [19]		•			•		•			•
11	2018	“From Craft to Science: The Road Ahead for Empirical Software Engineering Research” [20]		•				•		•	•	
12	2014	“Bridging the Gap: SE Technology Transfer into Practice: Study Design and Preliminary Results” [21]		•				•		•	•	
13	1994	“Key Lessons in Achieving Widespread Inspection Use” [22]			•		•		•			•
14	2006	“Co-Evolutionary Service-Oriented Model of Technology Transfer in Software Engineering” [23]		•				•		•	•	
15	1989	“Diffusing Software Engineering Methods” [3]		•				•		•	•	
16	2015	“Patterns of Cooperative Technology Development and Transfer for Engineering in the Large” [24]		•				•		•	•	
17	2012	“Infusing Scientific Foundations into Enterprise Interoperability” [25]	•	•			•		•			•
18	2003	“Experimental Validation of New Software Technology” [26]				• No Access						
19	2006	“Software Architecture at a Large Financial Firm” [27]			•		•		•			•
20	2002	“Requirements Researchers: Do We Practice What We Preach?” [28]			•		•		•			•

21	2018	"Generality vs. Reusability in Architecture-Based Self-Adaption: The Case for Self-Adaptive Microservices" [29]		•			•		•			•
22	2009	"Experiences in Developing and Applying a Software Engineering Testbed" [30]		•			•		•		•	
23	2010	"Confronting the Myth of Rapid Obsolescence in Computing Research" [31]		•			•		•		•	
24	2003	"Writing Good Software Engineering Research Papers: Minitutorial" [32]			•		•		•			•
25	1990	"Lessons from the Design of the Eiffel Libraries" [33]		•			•		•			•
26	1986	"Software Engineering: An Emerging Discipline" [34]				• No Access						
27	2013	"A Systematic Review of System-of-Systems Architecture Research" [35]	•	•			•		•			•
28	2002	"Software Engineering Technology Watch" [36]				• Duplicate						
29	2011	"A Quantitative Model for Software Engineering Trends" [37]		•				•		•	•	
30	2007	"An Empirical Study of Slice-Based Cohesion and Coupling Metrics" [38]			•		•			•	•	
31	2017	"Programming Language Adoption as an Epidemiological Phenomenon" [39]		•			•			•		•
32	2002	"Software Engineering Technology Watch" [40]				• Duplicate						
33	2013	"Empirical Studies for Innovation Dissemination: Ten Years of Experience" [41]		•			•		•			•
34	1986	"The Department of Defense Software Initiative - A Status Report" [42]		•			•		•			•
35	2003	"Influences on the Design of Exception Handling ACM SIGSOFT Project on the Impact of Software Engineering Research on Programming Language Design" [43]		•			•			•		•
36	2003	"Influences on the Design of Exception Handling ACM SIGSOFT Project on the Impact of Software				• Duplicate						

		Engineering Research on Programming Language Design" [44]										
37	2003	"External Experiments: A Workable Paradigm of Collaboration Between Industry and Academia" [45]				• No Access						
38	2009	"Software Engineering Technology Innovation - Turning Research Results into Industrial Success" [46]		•				•		•	•	
39	2007	"Debugging Aspect-Enabled Programs" [47]			•		•			•	•	
40	2002	"Business Process Reengineering and Workflow Automation: A Technology Transfer Experience" [48]		•				•		•	•	
41	2009	"Evaluating Legacy System Migration Technologies Through Empirical Studies" [49]		•			•		•			•
42	2012	"An Industrial Case Study of Performance and Cost Design Space Exploration" [50]			•		•		•			•
43	2012	"A Systematic Review of Software Architecture Evolution Research" [51]	•	•			•		•			•
44	2011	"Impact of Software Resource Estimation Research on Practice: A Preliminary Report on Achievements, Synergies, and Challenges" [52]			•		•		•			•
45	2016	"How Do Free/Open Source Developers Pick Their Tools? A Delphi Study of the Debian Project" [53]		•			•			•	•	
46	2008	"Developing Legacy System Migration Methods and Tools for Technology Transfer" [54]				• No Access						
47	2016	"Software Architecture for Robotic Systems" [55]	•	•			•		•			•
48	2001	"The Coming-of-Age of Software Architecture Research" [56]	•	•			•		•			•
49	2011	"A Method for Evaluating Rigor and Industrial Relevance of Technology Evaluations" [57]		•			•		•			•
50	2015	"A Systematic Review of Argumentation Techniques for Multi-Agent Systems Research" [58]	•	•			•		•			•
51	2008	"Design Rationale: Researching Under Uncertainty"[59]		•			•			•		•
52	1988	"Mapping the Design Information Representation Terrain" [60]		•			•		•			•

53	2005	“The Impact of Software Engineering Research on Modern Programming Languages” [61]		•			•			•	•	
54	2007	“The Impact of Research on Middleware Technology” [62]		•			•		•			•
55	2007	“The Impact of Research on Middleware Technology” [63]				• Duplicate						
56	2008	“The Impact of Research on the Development of Middleware Technology” [64]				• Duplicate						
57	2007	“Research Directions in Requirements Engineering” [65]		•			•			•	•	
58	2009	“Routinizing the offshore choice: applying diffusion of innovation to the case of EDS” [66]		•			•		•			•
59	2018	“Mapping the values of IoT” [67]		•			•		•			•
60	2009	“Technology Transfer decision support in requirements engineering research: a systematic review of REj” [68]		•				•		•	•	
61	2011	“An Analysis and Survey of the Development of Mutation Testing” [69]		•			•		•			•
62	1999	“Understanding and improving technology transfer in software engineering” [1]		•			•			•	•	
63	2004	“Capture-recapture in software inspections after 10 years research-theory evaluation and application” [70]			•		•		•			•
64	2004	“A framework for classifying and comparing software architecture evaluation methods” [71]		•				•		•	•	
65	2002	“What makes good research in software engineering?” [72]		•			•		•			•
66	1997	“Maintenance of COTS-intensive software systems” [73]				• No Access						
67	2005	“Design Considerations for Information Systems to Support Critical Infrastructure Management” [74]		•			•		•			•
68	1988	“The role of measurement in software engineering” [75]		•			•		•			•
69	1994	“Inspecting module interface specifications” [76]				• No Access						
70	1996	“Software engineering technology infusion within NASA” [77]		•				•	•			•

71	2018	"Continuous and collaborative technology transfer: Software engineering research with real-time industry impact" [78]		•			•			•	•	
72	2003	"Can We Influence Students' Attitudes About Inspections? Can We Measure a Change in Attitude?" [79]		•			•		•			•
73	2000	"The impact: project: determining the impact of software engineering research upon practice" - Impact Project Whitepaper [80]		•			N/A	N/A	•			•
74	2016	"The impact of human-centric design on the adoption of information systems: A case study of the spreadsheet" [81]	•	•			•		•			•
75	1995	"Software Engineering Technology Transfer: Understanding the Process" [82]		•			•		•			•
76	2017	"Engineering of self-adaptive systems: an organized tour" [83]	•	•			•		•			•
77	2008	"Improving Situational Ontologies to Support Adaptive Crisis Management Knowledge Architecture" [84]		•			•		•			•
78	1995	"Bottlenecks in the Transfer of Software Engineering Technology: Lessons Learned from a Consortium Failure" [85]		•			•		•			•
79	2016	"Technology Transfer Concepts" [86]		•			•		•			•
80	2017	"Center for High Integrity Software System Assurance" [87]		•			•			•	•	
81	2010	"CS in CSCL" [88]		•			•		•			•
82	2003	"PRISM: A Systematic Approach to Planning Technology Transfer Campaigns" [89]		•			•		•			•
83	2008	*Software engineering: principles and practice* [90]		•			•		•			•
84	2011	"Considerations for a Generalized Reuse Framework for System Development" [91]		•			•			•	•	
85	2012	"Systems Engineering Perspectives on Technology Readiness Assessments in Software-Intensive System Development" [92]				• No Access						
86	2003	"Experimental Validation of New Software Technology" [93]				• No Access						
87	2013	"What Industry Needs from Architectural		•			•		•			•

		Languages: A Survey" [94]										
88	2007	"Monitoring knowledge: A text-based approach" [95]				• No Access						
89	2013	"On The Scientific Maturity of Digital Forensics Research" [96]		•			•		•			•
90	2018	"Approaches, success factors, and barriers for technology transfer in software engineering - Results of a systematic literature review" [97]				• No Access						
91	1996	"Formal Methods Are a Surrogate for a More Serious Software Concern" [98]		•			•		•			•
92	1995	"Advances in Computers - Case Adoption: A Process, Not an Event" [99]				• No Access						

.

4.1 Findings

After a thorough search, the total number of publications which cite the Redwine and Riddle study are 92. In total, 76 articles were reviewed and evaluated. The results for each category are discussed and depicted by a graphical representation to demonstrate the outcomes.

4.1.1 Date

In Figure 6, from 1986 - 1989 there were 6 articles. From 1990 - 1999 there were 7 articles. From 2000 - 2009 there were 33 articles. From 2010 - 2019 there were 30 articles.

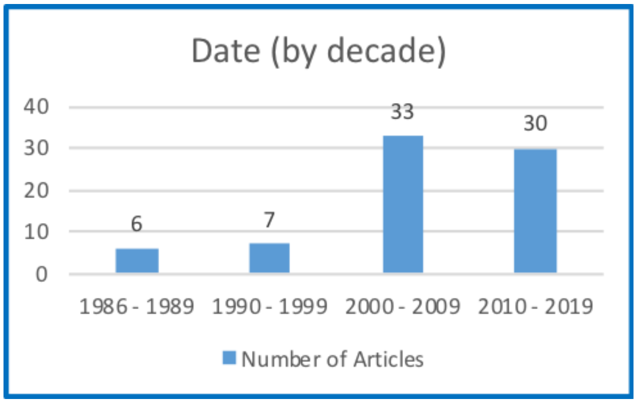


Figure 6. Bar chart of the results for the Date category.

4.1.2 Primary Treatment - 4 subcategories: Application, Direct Reference, Indirect Reference, and Error

During the evaluation process, 16 of the articles are categorized as an Error. Of those nine Error articles, 5 are specified as Duplicate. In several instances, the same article was published in different publications. These articles are categorized as Error Duplicate. The remaining 11 articles are not accessible. These articles are categorized as Error - No Access. In Figure 7, these articles account for approximately 17% (17.39%) of the total number of articles. Error articles are numbers 18, 26, 28, 32, 36, 37, 46, 55, 56, 66, 69, 85, 86, 88, 90, 92.

There 10 articles are categorized as an Application. The Application subcategory refer to those articles which apply the Redwine and Riddle phase criteria to a specific software technology. In Figure 7, Application articles are significant as these articles demonstrate the findings for the second implementation in this study. These articles account for approximately 13% (13.15%) of the total number of articles. All Application articles are also Direct Reference articles. Application articles are numbers 2, 3, 17, 27, 43, 47, 48, 50, 74, and 76.

As shown in Figure 7 below, Direct Reference is the largest subcategory of the Primary Treatment category. The Direct Reference subcategory are the articles which directly cite the authors names or the study's contents. There are 65 Direct Reference articles. In Figure 7, Direct Reference articles account for approximately 86% (85.52%) of the total number of articles. Direct Reference articles are numbers 1 - 4, 7 - 12, 14 - 17, 21 - 23, 25, 27, 29, 31, 33 - 35, 38, 40 - 41, 43, 45, 47 - 54, 57 - 62, 64 - 65, 67 - 68, 70 - 84, 87, 89, and 91.

The Indirect Reference subcategory represent the articles which indirectly cite the study's contents. In Figure 7, there are 11 indirect articles. These articles account for approximately 14% (14.47%) of the total number of articles. Indirect Reference articles are numbers 3, 4, 5, 6, 13, 19, 20, 24, 30, 39, 42, 44, and 63.

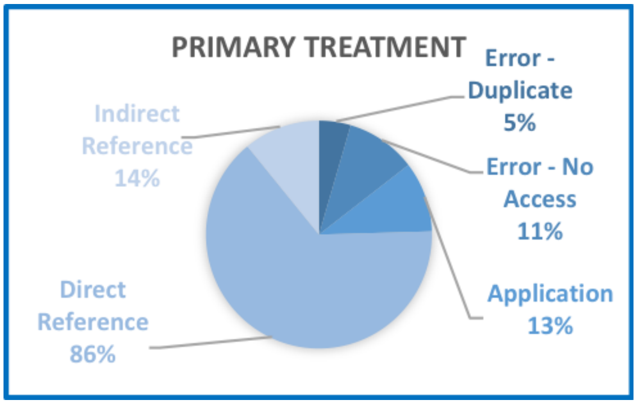


Figure 7. Pie chart of the results for the Primary Treatment category.

4.1.3 Opinion Category - 2 subcategories: Affirm or Oppose

There are 63 articles subcategorized as Affirm. The Affirm subcategory refers to those articles which affirm or agreed with the contents of the Redwine and Riddle study. These articles account for approximately 83% (82.89%) of the evaluated articles.

In Figure 8 below, the Oppose subcategory are the articles which directly oppose or counter the Redwine and Riddle study's contents. There are 12 articles which oppose the Redwine and Riddle study. Oppose articles account for approximately 16% (15.78%) of the evaluated articles.

Special Note: There is one article that neither affirms or oppose the Redwine and Riddle study. This article is a whitepaper for the Impact Project with results to come. The results would affirm or oppose the Redwine and Riddle study.

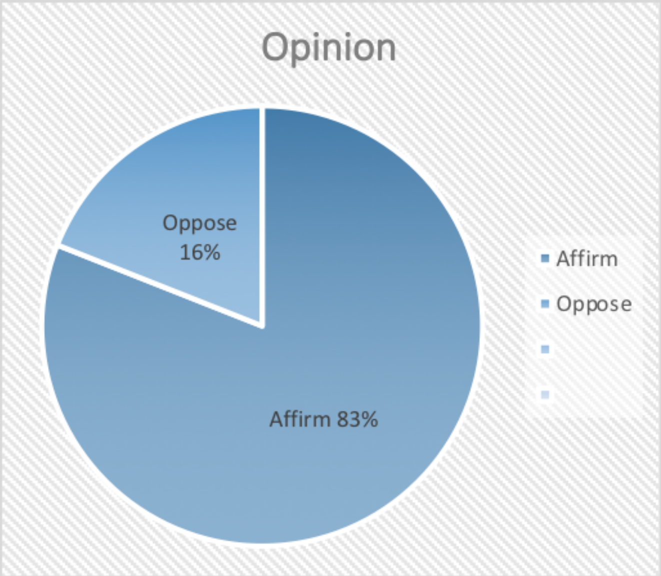


Figure 8. Pie chart of the results for the Opinion category.

4.1.4 Primary Source - 2 Subcategories: Yes or No

There are 52 articles subcategorized as Primary - Yes. The Primary subcategory refer to those articles which used the Redwine and Riddle study as their primary source regarding software technology maturation. These articles account for approximately 68% (68.42%) of the evaluated articles.

The Primary - No subcategory are the articles which did not use the Redwine and Riddle study as their primary source regarding software technology maturation. As shown in Figure 9, there are 24 Primary - No articles. Primary - No articles accounted for approximately 32% (31.57%) of the evaluated articles.

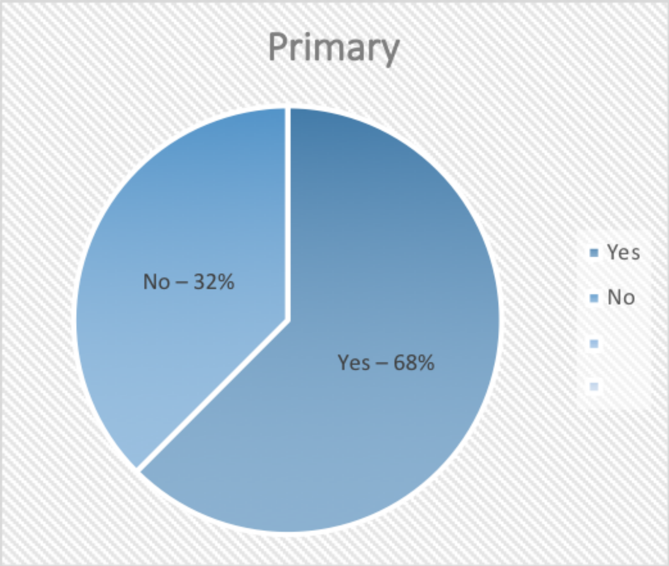


Figure 9. Pie chart of the results for the Primary category.

4.1.5 Alternate Source

There are 23 articles subcategorized as Alternate - Yes. The Alternate - Yes subcategory refer to those articles which use an alternate source instead of *or* in addition to the Redwine and Riddle study regarding software technology maturation. Alternate - Yes articles account for approximately 30% (30.26%) of the evaluated articles.

The Alternate - No subcategory are the articles which did not use an alternate source regarding software technology maturation. As seen in Figure 10 below, there are 53 Alternate - No articles. Alternate - No Direct articles account for approximately 70% (69.73%) of the evaluated articles.

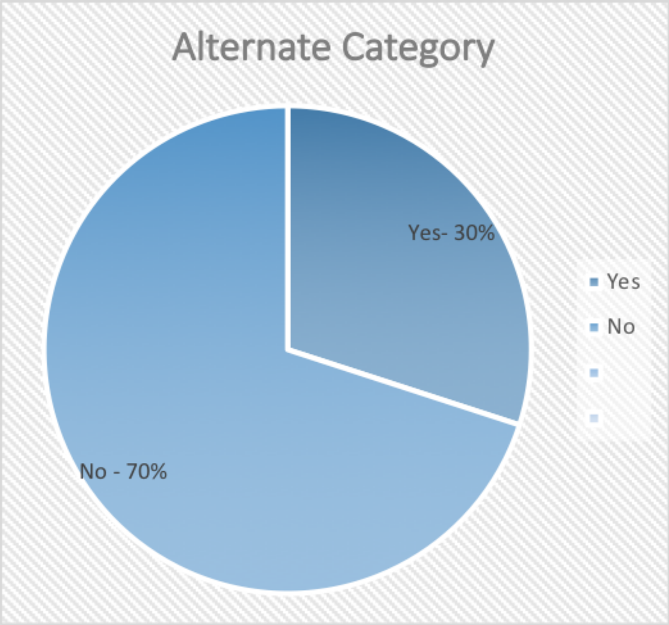


Figure 10. Pie chart of the results for the Alternate source category.

4.1.6 Applications of Redwine and Riddle's Six Phase Criteria to Other Software Technologies

The results addressed in this section of the implementation comes by determining if an author applied the six phases (maturation model) featured in the Redwine and Riddle study to another form of software technology. This designation is included via the previously discussed evaluation process. As noted in the assessment, there are 10 articles which feature applications of Redwine and Riddle's six phase criteria to other software technologies. These 'Application' articles are reviewed in order by date of publication.

In 2001, Mary Shaw applied the maturity model to the discipline of software architecture in two articles. The first application article, #48, entitled "The Coming of Age of Software Architecture" Shaw addresses each of the six phases of software technology maturity and discusses in great length each phase as it relates to software architecture. Then in 2006, the follow-up article, #3, entitled, "The Golden Age of Software Architecture," provides an elegantly revamped graphic of the original time points diagram by year, phase and event as depicted below in Figure 11.

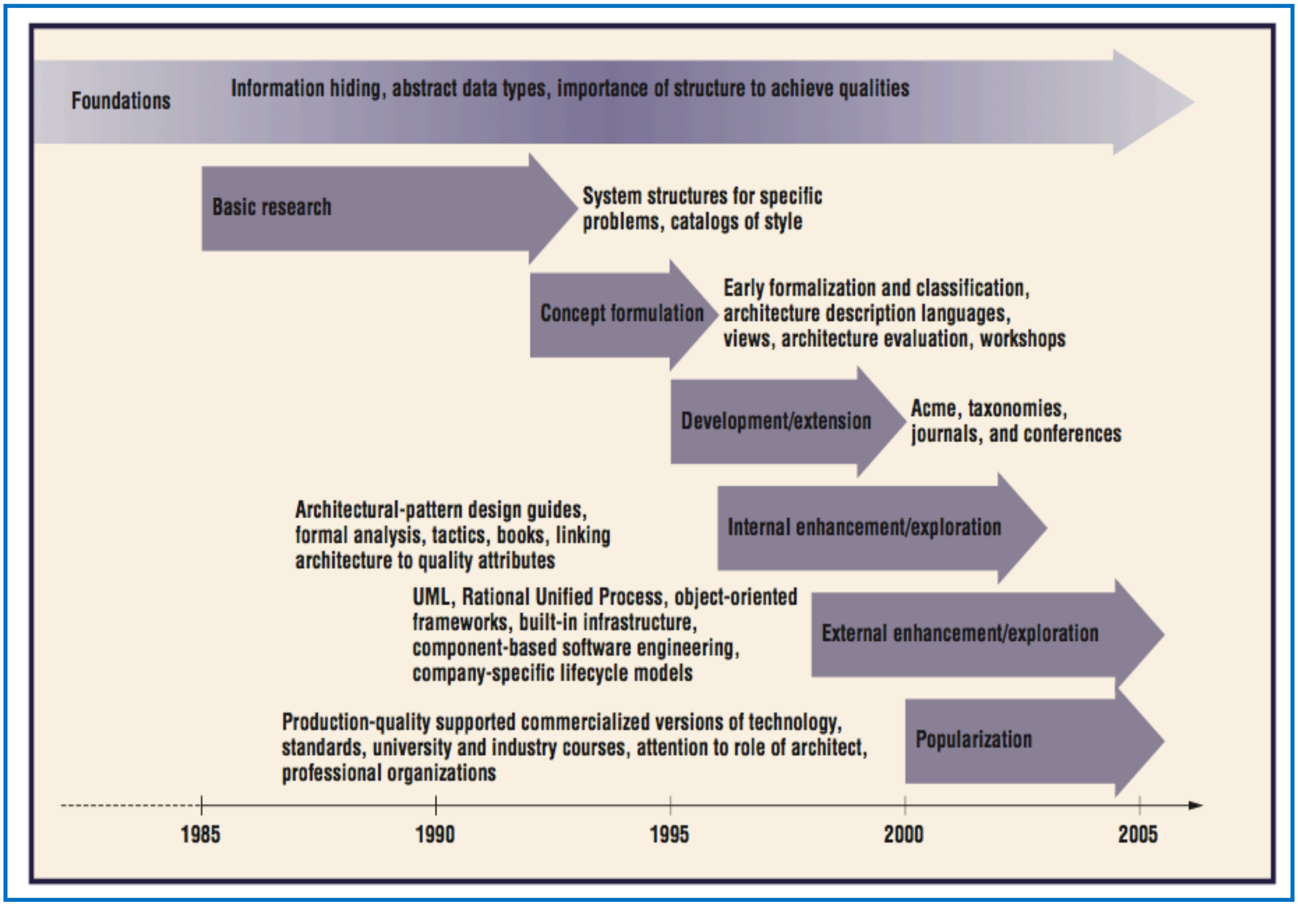
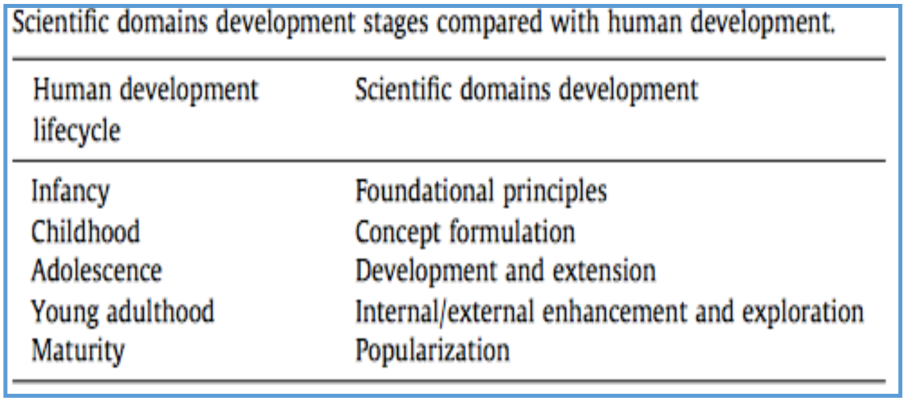

Foundations
Information hiding, abstract data types, importance of structure to achieve qualities
Basic research
System structures for specific problems, catalogs of style
Concept formulation
Early formalization and classification, architecture description languages, views, architecture evaluation, workshops
Development/extension
Acme, taxonomies, journals, and conferences
Architectural-pattern design guides, formal analysis, tactics, books, linking architecture to quality attributes
Internal enhancement/exploration
UML, Rational Unified Process, object-oriented frameworks, built-in infrastructure, component-based software engineering, company-specific lifecycle models
External enhancement/exploration
Production-quality supported commercialized versions of technology, standards, university and industry courses, attention to role of architect, professional organizations
Popularization
1985
1990
1995
2000
2005

Figure 11. Graphical depiction of the advancement to maturity of software architecture [12].

In “The Golden Age of Software Architecture,” Shaw and Clements chart portrays the six phases original to the Redwine and Riddle study. Shaw and Clements admit to expanding the model as they tracked the progression to maturity in software architecture. The authors include a foundational phase at the beginning of their maturation model [12].

In 2012, # 17, Lampathaki, Koussouris, Agostinho, Jardim-Goncalves, Charalabidis, and Psarras discuss each of the six phases specifically to their action plan: software engineering regarding Enterprise Interoperability (EI) technology in a third article. The purpose of the article is to establish a science baseline and provide an overview of the main events that will eventually define Interoperability as a scientific discipline. However, in their study the authors correlate the human development lifecycle to the scientific domains development as evidenced in Figure 12.



Scientific domains development stages compared with human development.

Human development lifecycle	Scientific domains development
Infancy	Foundational principles
Childhood	Concept formulation
Adolescence	Development and extension
Young adulthood	Internal/external enhancement and exploration
Maturity	Popularization

Figure 12. A correlation of human development and maturation phases [25].

In a fourth article, # 43, regarding the field of architecture evolution and software evolvability, Breivold, Crnkovic, and Larsson depict the maturity model in the following bar diagram. Each of the six phases are represented on the y-axis. The x-axis represents the number of number of studies identified in each phase from the 82 independently reviewed studies, see Figure 13.

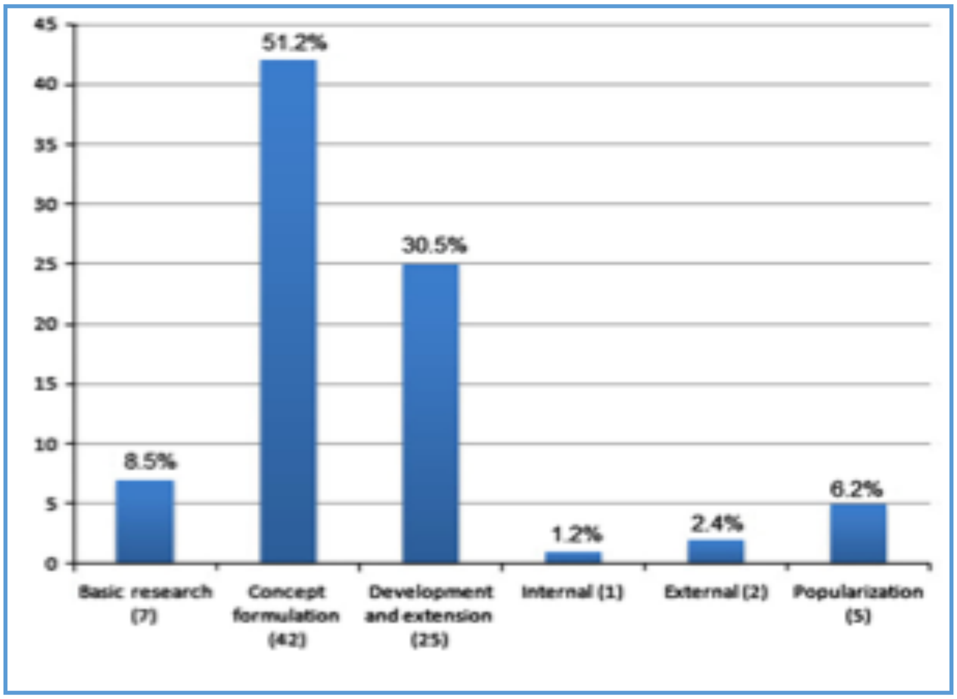


Figure 13. Bar graph of the maturity model in architecture evolution and software evolvability [51].

In this 2012 study, 7 articles are identified in the basic research phase which represent 8.5% of the 82 studies. Concept formulation has 42 articles which represent 51.2%, Development and Extension has 25 articles representing 30.5%, Internal has 1 which account for 1.2%, External has 2 which doubles internal at 2.4%, and Popularization has 5 articles, representing 6.2% [51].

The next article, # 27, Klein and van Vliet in "A Systematic Review of System-of-Systems Architecture Research" apply the maturity model to the system-of-systems technology architecture discipline as depicted in the following chart. Similarly, the authors track the number of articles identified for each phase of the Redwine and Riddle maturity model [35] as shown in Figure 14 below.

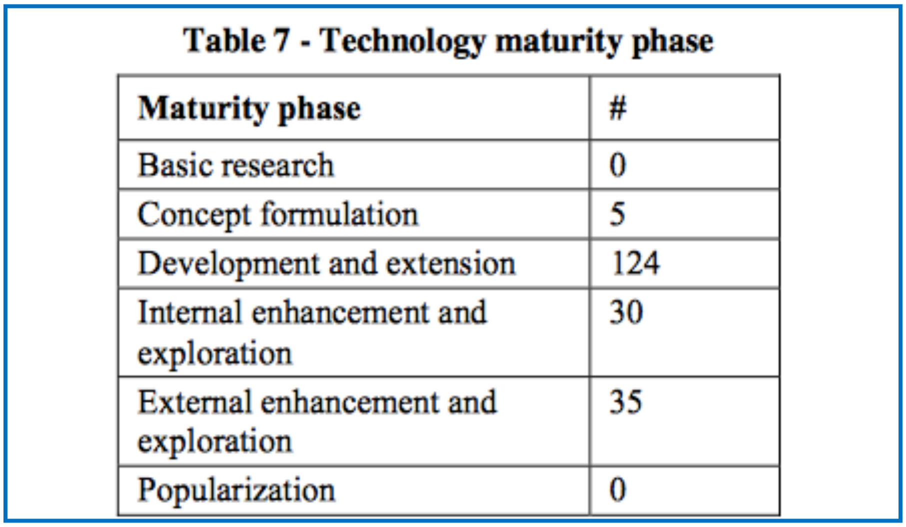


Table 7 - Technology maturity phase

Maturity phase	#
Basic research	0
Concept formulation	5
Development and extension	124
Internal enhancement and exploration	30
External enhancement and exploration	35
Popularization	0

Figure 14. Maturity phase and the number of articles in system-of-systems technology [35].

Klein and van Vliet note that the system-of-systems technology field is approximately 14 years into the maturation process from the date of their publication. The authors also comment that when compared to other software technologies, the system-of-systems architecture technology field is maturing rather slowly as of 2013 [35].

The sixth application article features a partial application. The phases of the software technology maturation model are partially applied in the article, *"A Systematic Review of Augmentation Techniques for Multi-Agent Systems Research"* by Carrera and Iglesias in 2015. The following Table 2 depicts the partial application.

Table 2. Maturity level of studies for multi-agent research technology [58].

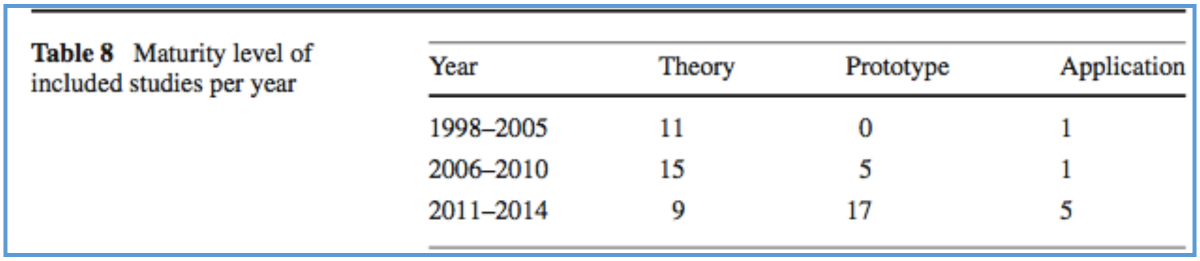


Table 8 Maturity level of included studies per year

Year	Theory	Prototype	Application
1998–2005	11	0	1
2006–2010	15	5	1
2011–2014	9	17	5

The authors note an increase in studies that present results from a prototype. Furthermore, it is possible to say that the solutions that apply the technology to problems will be offered in the future. This implies an early placement of augmentation technology into Phase 4.

A seventh application highlights a different interpretation on the depiction of the Redwine and Riddle maturation model. Instead of addressing each of the six phases individually, Ahmad & Babar show the yearly progression to maturity for robotic systems in Figure 15.

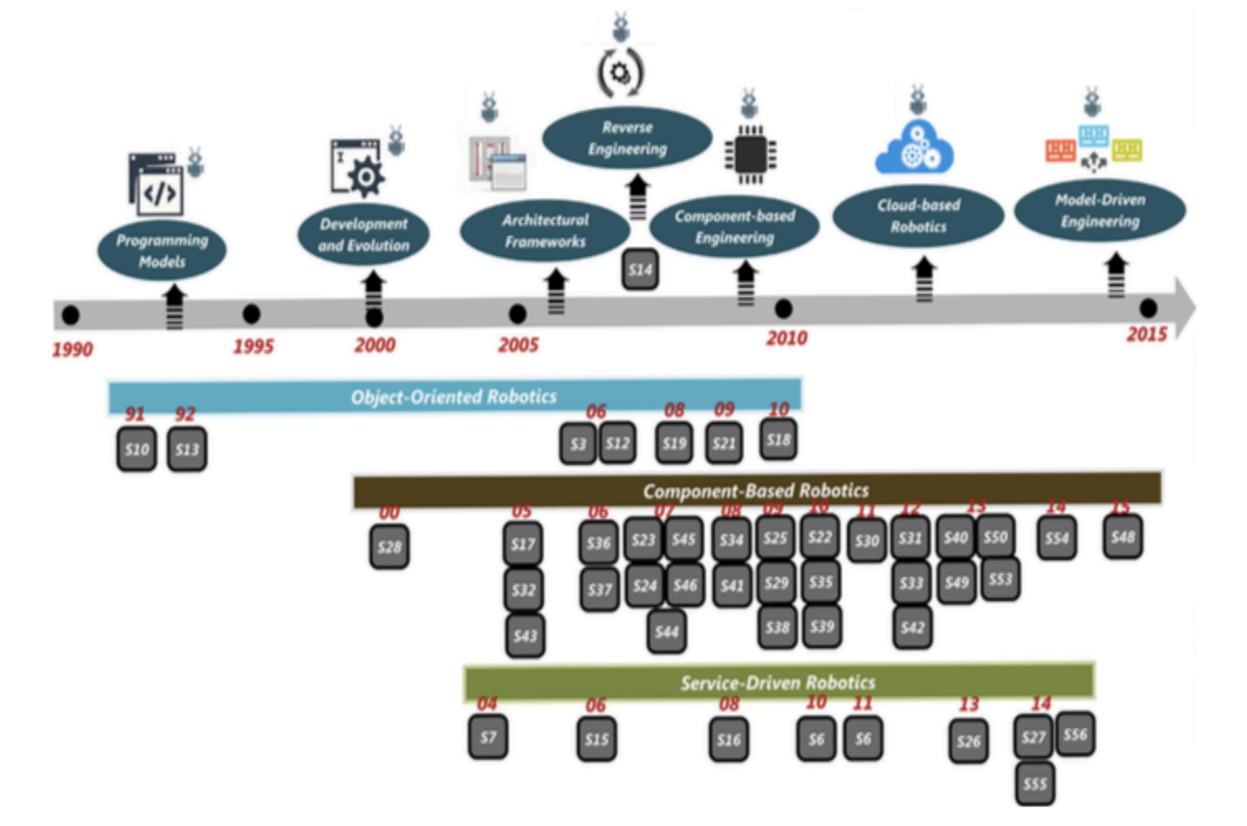


Figure 15. The 20-year progression to maturity for robotic systems [55].

Figure 15 demonstrates that early research on the architectural solutions of robotic systems began in the 1990's. However, the maturation and bulk of the state-of-the-research is only apparent in the last decade. These results demonstrate an approximate 20-year maturation timeline which aligns with the Redwine and Riddle study [55].

The eighth application featured in a 2016 article surrounds human-centric design of information systems. In "The Impact of Human-Centric Design on the Adoption of Information Systems: A Case Study of the Spreadsheet," Scaffidi does not provide a graphical depiction of his case study of the spreadsheet. However, Scaffidi does discuss at depth numerous acme and milestones that signify the progression of phases to maturation. Scaffidi demonstrates each maturation point as illustrated in the original Redwine and Riddle study but the spreadsheet technology [81]. For example, for Basic Research, Scaffidi begins this phase with the highlighting Richard Mattessich as the inventor of the spreadsheet tool for budgetary resources. Furthermore, for the Enhancement and Exploration phase, Scaffidi points out that the first customer-oriented spreadsheet was created by Software Arts in 1979 which signifies the 'usable capability' feature applicable to that phase. Lastly, to show propagation, Scaffidi discusses patents as well as the 1989 statistic that 10% of American that used computers at work also used spreadsheets [81].

The ninth article details the progression to maturity of the field of self-adaption from a 2017 publication, "Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges," D. Weyns implores a familiar graphical representation similar to the Shaw & Clements in Figure 16 below.

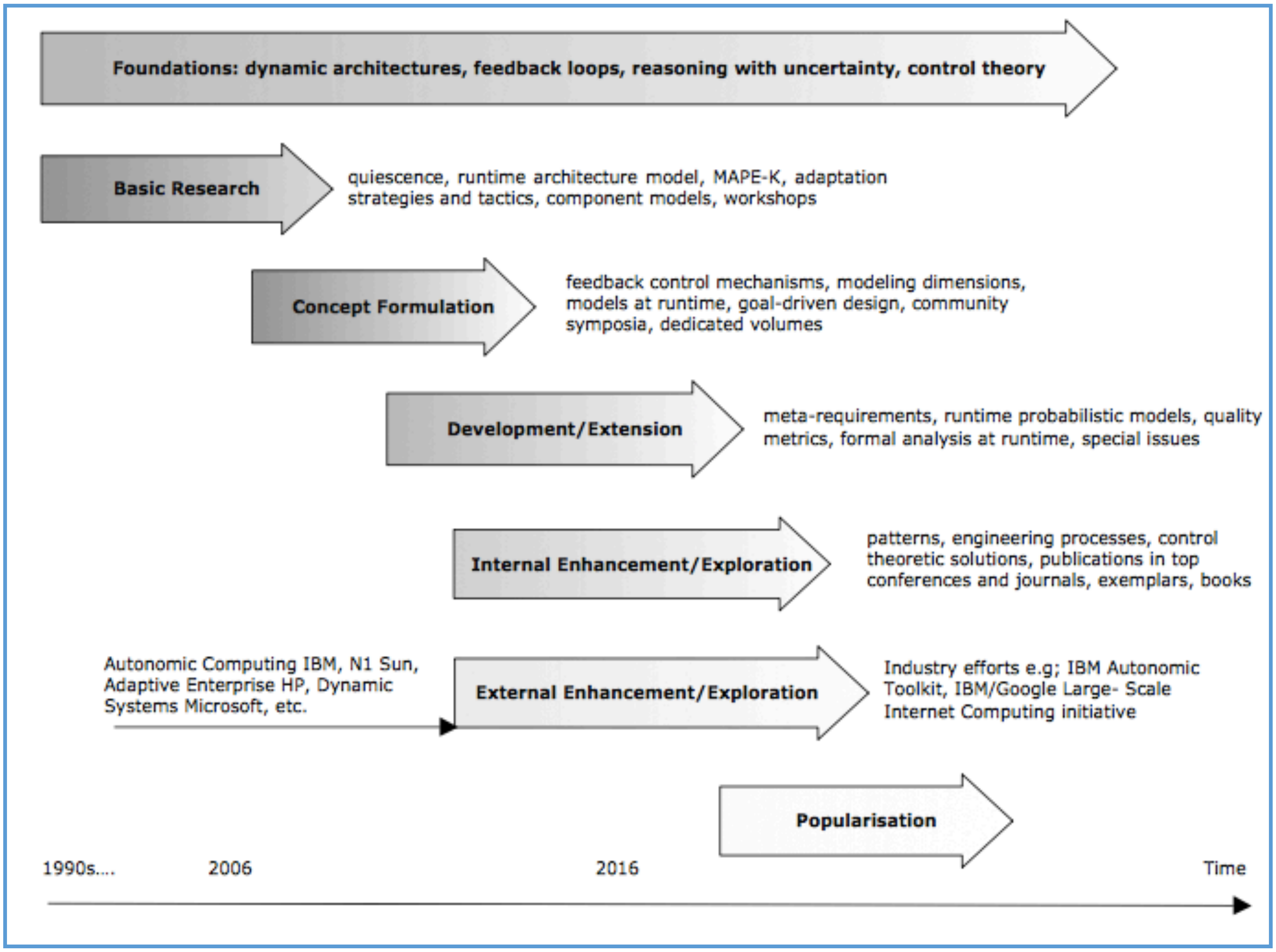

Foundations: dynamic architectures, feedback loops, reasoning with uncertainty, control theory
Basic Research
quiescence, runtime architecture model, MAPE-K, adaptation strategies and tactics, component models, workshops
Concept Formulation
feedback control mechanisms, modeling dimensions, models at runtime, goal-driven design, community symposia, dedicated volumes
Development/Extension
meta-requirements, runtime probabilistic models, quality metrics, formal analysis at runtime, special issues
Internal Enhancement/Exploration
patterns, engineering processes, control theoretic solutions, publications in top conferences and journals, exemplars, books
Autonomic Computing IBM, N1 Sun, Adaptive Enterprise HP, Dynamic Systems Microsoft, etc.
External Enhancement/Exploration
Industry efforts e.g; IBM Autonomic Toolkit, IBM/Google Large- Scale Internet Computing initiative
Popularisation
1990s....
2006
2016
Time

Figure 16. The self-adaption systems progression to maturity [83].

The tenth and final application article to feature an application maturity model is demonstrated in the article, "Software Project Management in High Maturity: Systematic Literature Mapping" by Cerdeiral and Santos. The following pie chart demonstrates the specific phase of software technology maturation for the selected papers within the study. Papers were categorized into four of Redwine and Riddle's six phases of the maturation process.

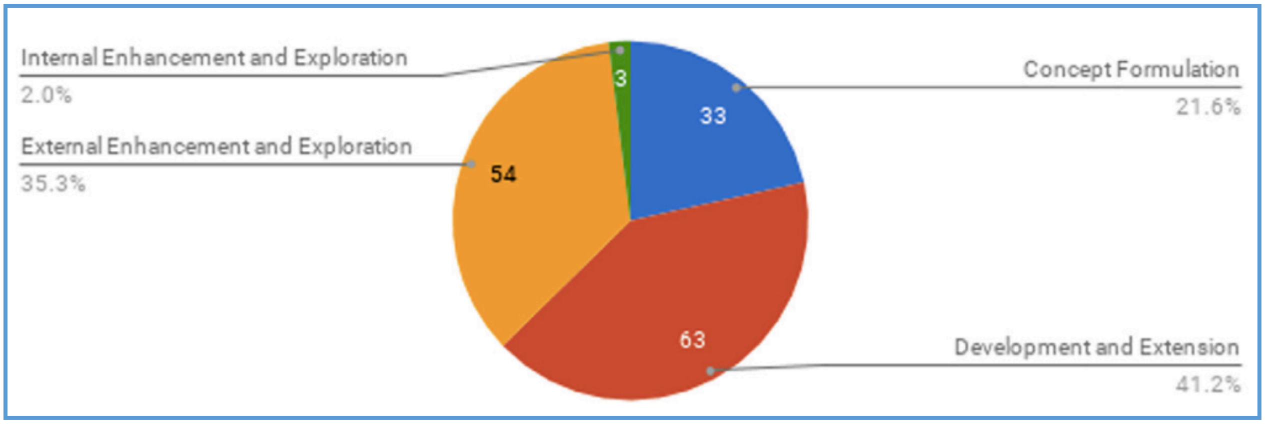


Figure 17. Maturity of software project management [11].

Four of the original six phases represented in this study in Figure 17 above are Concept Formulation, Development and Extension, Internal Enhancement and Exploration and Eternal Enhancement and Exploration. A significant percentage of papers provides evidence of their adoption thereby falling into the External Enhancement and Exploration phase. The largest amount of papers fell into the Development and Extension phase.

CHAPTER 5

SOFTWARE SECURITY INTRODUCTION

Since the late ’90s, computer security has evolved toward software security, also known as application security. A simple definition for software security is the idea of engineering software so that it continues to function correctly under malicious attack [6]. Software security is not security software and does not include firewalls, intrusion detection, encryption, or protecting the environment within which the software operates. The reasons for software insecurity can vary form complexity, flawed specification, flawed specifications, poor implementation of software interfaces, not thinking like an attacker, zero or minimal consideration for security during each phase of the SDLC or an inadequate knowledge of secure coding practices [100]. This is significant because all new software can be assumed to have errors [101. As a result, the discipline of software security is imperative. Although as a discipline software security is comparatively young to other disciplines, much progress has been made on ways to integrate security best practices into the software development life cycle [6]. The primary goal of this implementation is to extend the second research question and demonstrate the state of maturity for the field of software security.

5.1 Background and History: Software Security Foundation and Roots

Gary McGraw, a founding father of software security, stated that the taxonomy of software security came from computer security [6]. No better demonstration of this fact exists than the computer security seminal papers listed on the CSRC - NIST website. These 16 seminal papers are from 1970 - 1985. The names and authors of the papers along with the keywords are listed below for comparison. The keywords illustrate the computer security roots of the discipline software security.

i. James P. Anderson, *Computer Security Technology Planning Study Volume II*, ESD- TR-73-51, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01730 (Oct. 1972). **Keywords** - security kernel, reference monitor, Trojan horse, penetration, disclosure.

ii. James P. Anderson, *Computer Security Threat Monitoring and Surveillance*, James P. Anderson Co, Fort Washington, PA (1980). **Keywords** - audit, log, surveillance, monitoring, variation, intrusion detection.

iii. David E. Bell and Leonard J. LaPadula, *Secure Computer System: Unified Exposition and MULTICS Interpretation*, MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA 01730 (Mar. 1976); also, ESD-TR-75-306, rev. 1, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01731.

    Keywords - security policy, model simple security condition, star property, asterisk-property, mathematical model, secure computer system, security, trusted subject.

iv. Richard Bisbey II and Dennis Hollingworth, *Protection Analysis: Final Report*, ISI/SR-78-13, University of Southern California/Information Sciences Institute, Marina Del Rey, CA 96291 (May 1978). **Keywords** - vulnerability, penetration, access control, error analysis, error-driven evaluation, error type, operating system security, protection evaluation, protection policy, software security.

v. Department of Defense, *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, National Computer Security Center, Ft. Meade, MD 20755 (Dec. 1985). Also known

as the “Orange Book.” **Keywords** - standard, trusted system, evaluation, Orange Book, protection, class, security requirement.

vi. Ford Aerospace and Communications Corporation, *Secure Minicomputer Operating System (KSOS) Executive Summary: Phase I: Design of the Department of Defense Kernelized Secure Operating System*, WDL-781, Palo Alto, CA 94303 (Mar. 1978). **Keywords** - trusted system, UNIX, formal specification, multilevel, security kernel, KSOS.

vii. Paul A. Karger and Roger R. Schell, *MULTICS Security Evaluation, Volume II: Vulnerability Analysis*, ESD-TR-74-193, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01731 (June 1974). **Keywords** - access control, multi-level system, operating system vulnerability, privacy, monitor, secure computer system, security kernel, penetration, security testing, segmentation.

viii. Theodore Linden, *Operating System Structures to Support Security and Reliable Software* NBS Technical Note 919, Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, Washington DC 20234 (Aug. 1976). **Keywords** - capability, capability-based addressing, extended-type objects, operating system structures, protection, reliable software, reliability, security, small protection domains, types.

ix. Philip A. Myers, *Subversion: The Neglected Aspect of Computer Security*, Master Thesis, Naval Postgraduate School, Monterey CA 93940 (June 1980). **Keywords** - subversion, protection policy, trap door, Trojan horse, penetration, access control, evaluation criteria, protection system, leakage of data, security kernel.

x. James P. Anderson, *Computer Security Technology Planning Study Volume II*, ESD- TR-73-51, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01730 (Oct. 1972). **Keywords** - trusted system, formal specification, security kernel, PSOS, provably secure.

xi. Grace H. Nibaldi, *Proposed Technical Evaluation Criteria for Trusted Computer Systems*, M79-225, The MITRE Corporation, Bedford, MA 01730 (Oct. 1979). **Keywords** - formal verification, classification, secure computer system, trusted computing base, evaluation criteria, evaluation process, policy, mechanism, assurance, level.

xii. J. M. Schacht, *Jobstream Separator System Design*, MTR-3022 Vol. 1, The MITRE Corporation, Bedford, MA 01730 (May 1975). **Keywords** - job stream separator, jobstream, isolation, security level, add on, reference monitor.

xiii. Roger R. Schell, Peter J. Downey, and Gerald J. Popek, *Preliminary Notes on the Design of Secure Military Computer Systems*, MCI-73-1, The MITRE Corporation, Bedford, MA 01730 (Jan. 1973). **Keywords** - secure computer system, secure model, secure design.

xiv. W. L. Schiller, *The Design and Specification of a Security Kernel for the PDP- 11/45*, MTR-2934, The MITRE Corporation, Bedford, MA 01730 (Mar. 1975). **Keywords** - security kernel, secure computer system, specification, model.

xv. Willis H. Ware, *Security Controls for Computer Systems (U): Report of Defense Science Board Task Force on Computer Security*, The RAND Corporation, Santa Monica, CA (Feb. 1970). **Keywords** - secure computing, trap door, Trojan horse, penetration, disclosure, physical security.

xvi. Jerold Whitmore, Andre Bensoussan, Paul Green, Douglas Hunt, Andrew Kobziar, and Jerry Stern, *Design for MULTICS Security Enhancements*, ESD-TR-74-176, Electronic

Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01731 (Dec. 1973). **Keywords** - MULTICS, containment, access control, operating system secure computing [102].

Further sources also illustrate software security's succession from computer security such as Saltzer-Schroeder's, *Security Principles* in 1975 [104]. This work features ten basic formulated security principles. Another source is Matt Curtin's book on "developing trust." Various articles at the beginning of the software security discuss trust. Microsoft's Trustworthy Computing Initiative spurred on by Bill Gates infamous memo that too denotes trust. NSA's Principles of Secure Design in 1993 and Generally Accepted Systems Security Principles (GASSP) are featured throughout software security as confidentiality, integrity, and availability are features attributes noted in Bill Gates' memo in January 2002. Finally, the International Information Security Foundation (I^2SF) of 1997 is significant as well [103].

Viewing the seminal publications of computer security assists with the dubious task of trying to differentiate computer security history and software security beginnings. The historical works of computer security also serve as part of the 'foundations' phase for software security progression to maturation. Although Redwine and Riddle do not feature a foundational phase in their maturation model others have [12]. The remaining portion of this study will extend the second research question with an implementation of the Redwine and Riddle maturation model for the field/discipline of software security.

CHAPTER 6

SOFTWARE SECURITY METHODOLOGY

A proven technique to see a fields growth is to examine the rate at which earlier works were a basis for successor works, or in other words, the most cited articles [12]. To determine this phenomenon within the field of software security, an advanced search on Google Scholar was completed. The keyword search included: "SOFTWARE SECURITY" OR "SECURITY DEVELOPMENT LIFECYCLE" OR "SECURE CODE" in the title of the publication. Only articles with these exact keywords in the titles were included in this search in order to keep the results manageable. However, this approach does introduce threats to validity. Older publications have more time to accumulate citations and vice versa for newer more recent publications. Also, publications without the keyword search are not included and may have more citations that those publications listed. However, the number of articles had to be limited to a reasonable number. The second threat to validation is that the following table lists only the 50 most cited publications from 1990 - 2019. Each publication is categorized according to the Redwine and Riddle software maturation phases.

CHAPTER 7

SOFTWARE SECURITY IMPLEMENTATION

A total of 50 publications were reviewed. The following Table 3 features an ID number, article title, publication name/type, publication year, citation count and phase. This information is demonstrated for replication purposes. Each publication is categorized according to the Redwine and Riddle software maturation phases.

Table 3. Categorized software security publications with highest citation count.

No.	Title	Publication	Year	Cited by	Phase
1	Software Security [104]	IEEE Security & Privacy	2004	513	CONCEPT FORMULATION
2	Software Security: Building Security In [105]	Book	2006	966	INTERNAL ENHANCEMENT & EXPLORATION
3	Milk or wine: does software security improve with age? [106]	USENIX Security Symposium	2006	184	DEVELOPMENT & EXTENSION
4	Secure software development by example [107]	IEEE Security & Privacy	2005	103	DEVELOPMENT & EXTENSION
5	Knowledge of software security [108]	IEEE Security & Privacy	2005	78	DEVELOPMENT & EXTENSION
6	Secure software architectures [109]	IEEE Symposium on Security & Privacy	1997	82	BASIC RESEARCH
7	Secure software updates: disappointments and new challenges [110]	HotSec	2006	109	DEVELOPMENT & EXTENSION
8	Is complexity really the enemy of software security [111]	Proceedings of the 4th workshop on Quality of protection	2008	103	INTERNAL ENHANCEMENT & EXPLORATION
9	Secure software installation on smartphones [112]	IEEE Security & Privacy	2010	87	INTERNAL ENHANCEMENT & EXPLORATION
10	The art of software security testing: identifying software security flaws [113]	Book	2006	84	INTERNAL ENHANCEMENT & EXPLORATION
11	A methodology for Secure Software Design [114]	Software Engineering Research and Practice	2004	76	CONCEPT FORMULATION
12	Processes for producing secure software [115]	IEEE Security & Privacy	2004	71	INTERNAL ENHANCEMENT & EXPLORATION
13	Threat-driven modeling and verification of secure software using aspect-oriented Petri nets [116]	IEEE Transactions on Software Engineering	2006	154	INTERNAL ENHANCEMENT & EXPLORATION
14	Software Security Engineering [117]	Book	2008	158	INTERNAL ENHANCEMENT & EXPLORATION
15	Software Security and Privacy Risks in Mobile E-Commerce [118]	ACM	2001	360	BASIC RESEARCH

16	Software Security Testing [119]	IEEE Security & Privacy	2004	189	DEVELOMENT EXTENSION
17	Raksha: a flexible information flow architecture for software security [120]	ACM SIGARCH Computer Architecture News	2007	336	DEVELOPMENT & EXTENSION
18	Seven pernicious kingdoms: A Taxonomy of software security errors [121]	IEEE Security & Privacy	2005	233	DEVELOPMENT & EXTENSION
19	19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them [122]	Emeryville: McGraw-Hill/Osborne	2005	200	CONCEPT FORMULATION
20	Building Secure Software: How to Avoid Security Problems the right Way (Paperback) [123]	Book	2001	980	CONCEPT FORMULATION
21	Fuzzing for software security testing and quality assurance [124]	Book	2018	273	EXTERNAL ENHANCEMENT & EXPLORATION
22	Low-level software security: Attacks and defenses [126]	Foundations of security analysis and design	2007	72	INTERNAL ENHANCEMENT & EXPLORATION
23	Byzantine-resilient secure software-defined networks with multiple controllers in cloud [127]	IEEE Transactions on Cloud Computing	2014	97	INTERNAL ENHANCEMENT & EXPLORATION
24	Secure code update for embedded devices via proofs of secure erasure [128]	European Symposium on Research in computer security	2010	86	INTERNAL ENHANCEMENT & EXPLORATION
25	Key Management and secure software updates in wireless process control environments [129]	Proceedings of the first ACM conference on wireless network security	2008	84	DEVELOPMENT & EXTENSION
26	On the importance of the separation-of-concerns principles in secure software engineering [130]	Workshop on the Application of Engineering Principles to System Security Design	2002	95	CONCEPT FORMULATION
27	Embedded systems security: practical methods for safe and secure software and systems development [131]	Book	2012	58	INTERNAL ENHANCEMENT & EXPLORATION
28	Let the pirate's patch? an economic analysis of software security patch restrictions [132]	Information systems research	2008	82	INTERNAL ENHANCEMENT & EXPLORATION
29	MAC and UML for secure software design [133]	2004 ACM Workshop on formal methods in security engineering	2004	62	CONCEPT FORMULATION
30	Improving software security with precise static and runtime analysis [134]	Book	2006	62	INTERNAL ENHANCEMNT & EXPLORATION
31	The security development lifecycle [135]	Book	2006	614	EXTERNAL ENHANCEMENT & EXPLORATION
32	The trustworthy computing security development lifecycle [136]	20th Annual Computer Security Applications	2004	244	DEVELOPMENT & EXTENSION
33	Safe-ops: An approach to embedded software security [137]	ACM	2005	57	CONCEPT FORMULATION
34	Securing traceability of ciphertexts - towards a secure software key escrow system [138]	International Conference on the Theory and Application of Cryptographic Techniques	1995	67	BASIC RESEARCH
35	Softwarepot: An encapsulated transferable file system for secure software circulation [139]	International Symposium on Software Security	2002	58	CONCEPT FORMULATION
36	Towards a structured unified process for software security [140]	2006 International Workshop on Software Engineering for Secure Systems	2006	49	EXTERNAL ENHANCEMENT & EXPLORATION
37	Secure code distribution in dynamically programmable wireless sensor networks 141]	Proceedings of the 5th International Conference on Information processing in sensor networks	2006	167	DEVELOPMENT & EXTENSION
38	Software security and SOA: danger, Will Robinson! [142]	IEEE Security & Privacy	2006	69	INTERNAL ENHANCEMENT & EXPLORATION
39	Prioritizing software security fortification throughcode-level metrics [143]	Proceedings of the 4th ACM workshop on quality of protection	2008	77	INTERNAL ENHANCEMENT & EXPORATION
40	On the secure software development process: CLASP, SDL and Touchpoints compared [144]	Information and software	2009	110	EXTERNAL ENHANCEMENT & EXPLORATION

41	Hiding program slices for software security [145]	Proceedings of the International Symposium on Code generation and optimization: feedback directed and runtime optimization	2003	62	DEVELOPMENT & EXTENSION
42	Software security testing [146]	IEEE Security & Privacy	2004	189	DEVELOPMENT & EXTENSION
43	Software security checklist for the software life cycle [147]	WET ICE	2003	85	CONCEPT FORMULATION
44	From the Ground up: The DIMACS software security workshop [148]	IEEE Security & Privacy	2003	89	CONCEPT FORMULATION
45	Writing Secure Code [149]	Book	2003	1143	DEVELOPMENT & EXTENSION
46	Improving software security with a c pointer analysis [150]	Proceedings of the 27th International Conference on Software Engineering	2005	87	DEVELOPMENT & EXTENSION
47	Applying formal methods to a certifiably secure software system [151]	IEEE Transactions on Software Engineering	2008	100	INTERNAL ENHANCEMENT & EXPLORATION
48	The art of software security assessment: identifying and preventing software vulnerabilities [152]	Book	2006	213	INTERNAL ENHANCEMENT & EXPLORATION
49	Network software security and user incentives [153]	Management Science	2006	141	INTERNAL ENHANCEMENT & EXPLORATION
50	Software security for open-source systems [154]	IEEE Security & Privacy	2003	114	CONCEPT FORMULATION

7.1 Redwine and Riddle maturation phases for software security.

7.1.1 Basic Research - Investigation of ideas and concepts that later prove fundamental; general recognition of problem and discussion of its scope [2]. Time period for this phase is 1995 - 2001 as illustrated in Table 3.

7.1.2 Concept Formulation - informal circulation of ideas; convergence on a compatible set of ideas; general publication of solutions to parts of the problem [2]. Demonstrated in Table 3, the time period for this phase is 2001 - 2005. This phase can feature workshops, evaluations, early formalization and classifications [12].

7.1.3 Development and Extension - trial, preliminary use of the technology; clarification of the underlying ideas, extension of the general approach to a broader solution [2]. Table 3 portrays the time period for this phase is 2003 - 2008. This phase can feature conferences, journals, and taxonomies [12].

7.1.4 Internal Enhancement and Exploration - Major extension of general approach to other domains; use of the technology to solve real problems; stabilization and parting of the technology; development of training materials; derivations of results indicating value [2]. Time period for this phase is 2006 - 2014 in Table 3. There is no officially adopted security development lifecycle model. This phase also features books and formal analysis [12].

7.1.5 External Enhancement and Exploration - Same activities are for Enhancement and Exploration (internal) but they are carried out by a broader group, including people outside the development group [2]. Time period for this phase is 2006 - 2018. This phase features outside personal security development lifecycles models, such as Microsoft's SDL, Citigal's Touchpoints, and OWASP's CLASP as shown in Table 3.

7.1.6 Popularization (Insufficient Data) - Appearance of production-quality, supported versions; commercialization and marketing of the technology; propagation of the technology throughout community of users - at 40% and at 70% [2]. From the publications listed, none of the titles correlate to this phase in Table 3. However, new publications can amend this discrepancy in the future.

CHAPTER 8

FUTURE WORK AND CONCLUSION

8.1 Redwine and Riddle's Software Technology Maturation

Future work for assessing the significance of the Redwine and Riddle maturation model will be an update on the number of works that cite the model in their scholarly articles. In addition, these articles can be added to the assessment to be evaluated.

The final conclusions regarding the Redwine and Riddle study are based upon the findings of the assessment tool and evaluation applied to each article within the implementation. To precisely apply the findings, I revisit the initial questions of interest.

- ***RQ1:*** *Is Redwine and Riddle's "Software Technology Maturation" study the accepted gold standard within the software engineering discipline for assessing the maturation of software technology?*

In the category of date signifying relevance, the decade with the second highest articles is the present decade (2010 - 2019) with 30 articles. And the year isn't over! The decade with the highest number of articles was the previous decade (2000 - 2009) which had 33. It is obvious that the Redwine and Riddle, "Software Technology Maturation" study has legs and is very relevant among modern scholars. Furthermore, approximately:

- 86% of the articles referenced the study directly;
- 83% affirmed the study;
- 68% used Redwine and Riddle as the primary study and

- 70% of the articles did not feature an alternate study.
- Plus, the study was most cited with 292 citations on Google Scholar.

These are relatively high numbers in favor of Redwine and Riddle's maturation model as the primary study and gold standard for assessing software technology maturation.

- ***RQ2:*** *Can the software technology maturation model be applied to current areas of software technology?*

There are 10 articles which feature applications of Redwine and Riddle's six phase criteria to other software technologies. These articles and the software technology are:

- Software Architecture in 2001 and 2006
- Enterprise Interoperability (EI) in 2012
- Architecture Evolution and Software Evolvability in 2012
- System-of-Systems Architecture in 2013
- Multi-Agent Systems in 2015
- Robotic Systems in 2016,
- Spreadsheet in 2016,
- Self-Adaptive Systems in 2017 and
- Software Project Management in 2019

Eight of the ten, or 80%, of the articles applied the maturation model to a software technology from the present decade, 2010 - 2019. The two articles from 2000 - 2009 discuss software architecture, a software technology whose maturation process reached full maturity around the year 2000. These dates illustrate the modern applications of the maturation model by Redwine and Riddle as well as answer affirmatively to the second research question regarding application to current areas of software technology. Proven as the golden standard and applicable to modern software technologies, the maturation model is applied to the discipline of software security.

8.2 Software Security Maturation

The inadequacies in the software security maturation model in Table 3 are areas for future research. Redwine and Riddle articulate in their original study that a technology can take 15 - 20 years to mature. Software security has made progress in that amount of time but has not reached full maturity from the publications listed in the implementation table. Significant work still needs to be completed, as noted in the final phase popularization as well as internal enhancement and exploration in Table 3. A security development lifecycle model for the field needs to be adopted by the software security community. In the Redwine and Riddle maturation model, he lengthiest scenario in the original study was 23 years. This may be the trajectory for the software security discipline. In order to properly compare the publications for software security, an updated study should be completed.

The conclusions of the software security maturation model can be viewed as accomplishments and shortcomings. A notable accomplishment is the software security technology extending to other domains as noted by a few studies featured in the Internal Enhancement and Exploration phase in Table 3. Also, the External Enhancement and Exploration phase is well represented by Microsoft's SDL, Citigal's Touchpoints and OWASP's CLASP. Finally, there appears to be a set of standards for software security as mentioned in the only article listed in the Popularization phase.

The inadequacies in the software security maturation model or shortcomings appear numerous. There are few if any classification schemes for vulnerabilities and threats for the Concept Formulation phase in Table 3. There does appear to be a comparatively large timeframe in the Basic Research phase of software security. As the for the Internal Enhancement and Exploration phase, the most glaring absence; of an overall generally accepted and adopted secure software development lifecycle from the software security community. Another notable absence is figures that support the overall popularization of the discipline within the tech community. Both Microsoft and Citigal boast their own popularization numbers regarding their models [6] [158] but there appears to be insufficient evidence of popularization for the discipline.

BIBLIOGRAPHY

1. S.L. Pfleeger, “Understanding and improving technology transfer in software engineering,” Journal of Software systems, vol. 46, (2-3), pp. 111 - 124, 1999.

2. S. T. Redwine and W. E. Riddle, “Software Technology Maturation,” Proceedings of the 8th International Conference on Software Engineering (ICSE 85), pp. 189 - 200, 1985.

3. S.A. Raghavan, D. R. Chand, “Diffusing Software-Engineering Methods,” IEEE Software, vol. 6, (4), pp. 81 - 90, 1989.

4. E. Lieblein, “The Department of Defense software initiative - a status report,” Communications of the ACM, vol. 29 (8), pp. 734 - 744, 1986.

5. E. Amoroso, “Recent Progress in Software Security,” IEEE Software, vol. 35 (2), pp. 11 - 13, 2018.

6. G. McGraw, “Technology Transfer, A Software Security Marketplace Case Study,” IEEE Software, vol. Gary McGraw, IEEE Software, vol. 28 (5), pp. 9 - 11, 2011.

7. J. N. Buxton, R. Malcolm, “Software Technology Transfer,” Software Engineering Journal, vol. 6 (1), pp. 17, 1991.

8. B. Kitchenham, R. Pretorius, D. Budgen and O. P. Brereton, M. Turner, M. Niazi and S. Linkman, “Systematic Literature Reviews in Software Engineering - A Tertiary Study,” Information and Software Technology, vol. 52, (8), pp. 792 - 805, 2010.

9. R. L. Glass, “Editor’s Corner: How About Next Year? A Look at a Study of Technology Maturation,” Journal of Systems and Software, vol. 9 (3), pp. 167 - 168, 1989.

10. S. L. Pfleeger, “Making change: understanding software technology transfer,” The Journal of Systems & Software, vol. 47 (2), pp. 111 - 124, 1999.

11. C.T. Cerdeiral and G. Santos, “Software Project Management in High Maturity: A Systematic Literature Mapping,” Journal of Systems and Software, vol. 148, pp. 56 - 87, 2019.

12. M. Shaw and P. Clements, “The Golden Age of Software Architecture,” IEEE Software, vol. 23 (2), pp. 31 - 39, 2006.

13. S. L. Pfleeger and W. Menezes, “Marketing Technology to Software Practicitioners,” IEEE Software, vol.17 (1), pp. 27 - 22, 2000.

14. Y. Chen, R. Dios, A. Mili, L. Wu, and K. Wang, "An Empiorical Study of Programming Language Trends," IEEE Software, vol. 22 (3), pp. 72 - 78, 2005.

15. A. Finkelsteiin and J. Kramer, "Software Engineering: A Roadmap," Proceedings of the Conference on The Future of Software Engineering, pp. 3 - 22, June 4 - 11, 2000, Limerick, Ireland.

16. J. Cleland-Huang, D. Damian, and S. Ghaisas, "Ready-Set-Transfer: Exploring the Technology Readiness of Academic Research Projects (panel)," Companion Proceedings of the 36 International Conference on Software Engineering, May 31 - June 7, 2014, Hyderabad, India.

17. R. Cowan, A. Mili, H. Ammar, A. McKendall, Jr., L. Yang, D. Chen, and T. Spender, "Software Engineering Technology Watch," IEEE Software, vol. 19 (4), pp. 123 - 129, 2002.

18. P. Bhawnani, G. Ruhe, F. Kudorfer, L. Meyer, "Intelligent Decision Support for Road Mapping a Technology Transfer Case Study with Seimens Corporate Technology," Proceedings of the 2006 International Workshop on Software Technology Transfer in Software Engineering, May 22 - 22, 2006, Shanghai, China.

19. E. F. Berliner and P. Zave, "An Experiement in Technology Transfer: PAISLey Specification of Requiremetns for an Undersea Lightware Cable System," Proceedings of the 9th International Conference on Software Engineering, pp. 42 - 50, 1987, Monterey, California, United States.

20. M. Galster, D. Weyns, A. Tang, R. Kazman, M. Mirakhorli, "From Craft to Science: The Road Ahead For Empirical Software Engineering Research," Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, May 27 - June 3, 2018, Gothenburg, Sweden.

21. P. Diebold and A. Vetro, "Bridging the Gap: Software Engineering Technology Transfer Into Practice: Study Design and Preliminary Results," Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, September 18 - 19, 2014, Torino, Italy.

22. R. B. Grady and T. Van Slack, "Key Lessons in Achieving Widespread Inspection Use," IEEE Software, vol. 11 (4), pp. 46 - 57, 1994.

23. M. Aoyama, "Co-Evolutionary Service-Orientied Model of Technology Transfer in Software Engineering," Proceedings of the 2006 International Workshop on Software Technology Transfer in Software Engineering, May 22 - 22, 2006, Shaghai, China.

24. C. Henrique and C. Duarte, "Patterns of Cooperative Technology Development and Transfer for Software Engineering in the Large," Proceedings of the Second International Workshop on Software Engineering Research and Industrial Practice, May 16 - 24, 2015, Florence, Italy.

25. F. Lampathaki, S. Koussouris, C. Agostinho, R. Jardim-Goncalves, Y. Charalabidis, and J. Psarras, "Infusing Scientific Foundations Into Enterprise Interoperability," Computer in Industry, vol. 63 (8), pp. 858 - 866, 2012.

26. M. V. Zelkowitz, D. R. Wallace, and D. W. Binkley, "Experimental Validation of New Software Technology, Lecture Notes on Empirical Software Engineering, World Scientific Publishing Co., Inc., River Edge, New Jersey, 2003.

27. G. Fairbanks, K. Bierhoff, and D. D'Souza, "Software Architecture at a Large Financial Firm," Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, October 22 - 26, 2006, Portland, Oregon, USA.

28. A. M. Davis and A. M. Hickey, "Requirements Researchers: Do We Practice What We Preach?," Requirements Engineering, vol. 7, (2), pp. 107 - 111, 2002.

29. N. C. Mendonca, D. Garlan, B. Schmerl, and J. Camara, "Generality vs. Reusability in Architecture-Based Self-Adaptation: The Case For Self-Adaptive Microservices," Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, September 2 - 28, 2018, Madrid, Spain.

30. A. Lam and B. Boehm, "Experiences in Developing and Applying Software Engineering Technology Testbed," Empirical Software Engineering, vol. 14, (5), pp. 579 - 601, 2009.

31. D. I. K. Sjoberg, "Confronting the Myth of Rapid Obsolescence in Computing Research," Communications of the ACM, vol. 53, (9), 2010.

32. M. Shaw, "Writing Good Software Engineering Research Papers: Minitutorial," Proceedings of the 25th International Conference on Software Engineering, May 3 - 10, 2003, Portland, Oregon.

33. B. Myer, "Lessons From the Design of the Eiffel Libraries," Communications of the ACM, vol. 33 (9), pp. 68 - 88, 1990.

34. R. Goldberg, "Software Engineering: An Emerging Discipline," IBM Systems Journal, vol. 25 (3 - 4), pp. 334 - 353, 1986.

35. J. Klein and H. van Vliet, "A Systematic Review of System-of-Systems Architecture Research," Proceedings fo the 9th International ACM SIGSOFT Conference on Quality of Software Architectures, June 17 - 21, 2013, Vancouver, British Columbia, Canada.

36. R. D. Cowan, A. McKendall, Jr., A. Mili, L. Yang, D. Chen, V. Janardhana and T. Spencer, "Software Engineering Technology Watch," Information Sciences - Informatics and Compurter Science: An International Journal, vol. 140, (3), pp. 195 - 215, 2002.

37. L. B. A. Rabai, Y. Z. Bai and A. Mili, "A Quantitative Model for Software Engineering Trends," Information Sciences: An International Journal, vol. 181 (22), pp. 4993 - 5009, 2011.

38. T. M. Meyers and D. Binkley, “An Empirical Study of Slice-Based Cohesion and Coupling Metrics,” ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 17, (1), pp. 1 - 27, 2007.

39. E. Barreiros, J. Albuquerque, J. F. L. de Oliveira, H. Lins and S. Soares, “Programming Language Adoption as an Epidemiological Phenomenon,” Proceedings of the 31st Brazilian Symposium on Software Engineering, September 20 - 22, 2017, Fortaleza, CE, Brazil.

40. R. D. Cowan, A. Mili, H. Ammar, A. McKendall, Jr., L. Yang, D. Chen and T. Spencer, “Software Engineering Technology Watch,” IEEE Software, vol. 19 (4), pp. 123 - 129, 2002.

41. M. T. Baldassarre, D. Caivano and G. Visaggio, “Empirical Studies For Innovation Disseminiation: Ten Year of Experience,” Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, April 14 - 16, 2013, Porto de Galinhas, Brazil.

42. E. Lieblein, “The Department of Defense Software Initiative - A Status Report,” Communications of the ACM, vol. 29 (8), pp. 737 - 744, 1986.

43. B. G. Ryder and M. L. Soffa, “Influences on the Design of Exception Handling: ACM SIGSOFT Project on the Impact of Software Enginering Research on Programming Language Design,” ACM SIGSOFT Engineering Notes, vol. 28 (4), 2003.

44. B. G. Ryder and M. L. Soffa, “Influences on the Design of Exception Handling: ACM SIGSOFT Project on the Impact of Software Engineering Research on Programming Language Design,” ACM SIGPLAN Notices, vol. 38 (6), 2003.

45. F. Houdek, “External Experiments: A Workable Paradigm For Collaboration Between Industry and Academia,” Lecture Notes on Empirical Software Engineering, World Scientific Publishing Co., Inc., River Edge, New Jersey, 2003.

46. T. Punter, R. L. Krikhaar and R. J. Bril, “Software Engineering Technology Innovation - Turning Research Results Into Industrial Success,” Journal of System and Software, vol. 82 (6), pp. 993 - 1003, 2009.

47. M. Eaddy, A. Aho, W. Hu, P. McDonald and J. Burger, “Debugging Aspect-Enabled Programs,” Proceedings of the 6th International Conference on Software Composition, March 24 - 25, 2007, Braga, Portugal.

48. L. Aversano, G. Canfora, A. De Lucia and P. Gallucci, “Business Process Reengineering and Workflow Automation: A Technology Transfer Experience,” Journal of Systems and Software, vol. 63, (1), pp. 29 - 44, 2002.

49. M. Colosimo, A. De Lucia, G. Scanniello and G. Tortora, “Evaluating Legacy System Migration Technologies Through Empirical Studies,” Information and Software Technology, vol. 51, (2), pp. 433 - 447, 2009.

50. T. de Gooijer, A. Jansen, H. Koziolek and A. Koziolek, “An Industrial Case Study of Performance and Cost Design Space Exploration,” Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, April 22 - 25, 2012, Boston, Massachusetts, USA.

51. H. P. Breivold, I. Crnkovic and M. Larsson, “A Systematic Review of Software Architecture Evolution Research,” Information and Software Technology, vol. 54, (1), pp. 16 - 40, 2012.

52. B. Boehm and R. Valerdi, “Impact of Software Resource Estimation Resource on Practice: A Preliminary Report on Achievements, Synergies, and Challenges,” Proceedings of the 33rd International Conference on Software Engineering, May 21 - 28, 2001, Waikiki, Honolulu, HI, USA.

53. M. F. Krafft, K. Stol and B. Fitzgerald, “How Do Free/Open Source Developers Pick Their Tools?: A Delphi Study on the Debian Project,” Proceedings of the 38th International Conference on Software Engineering Companion, May 14 - 22, 2016, Austin, Texas.

54. A. De Lucia, R. Francese, G. Scanniello and G. Tortora, “Developing Legacy System Migration Methods and Tools For Technology Transfer,” Software - Practice & Experience, vol. 38 (13), pp. 1333 - 1364, 2008.

55. A. Ahmad, and M. A. Baber, “Software Architectures For Robotic Systems,” Journal of Systems and Software, vol. 122, pp. 16 - 39, 2016.

56. M. Shaw, “The Coming of Age of Software Architecture Research,” Proceedings of the 23rd International Conference on Software Engineering, p. 656, May 12 - 19, 2001, Toronto, Ontario, Canada.

57. M. Ivarsson and T. Gorschek, “A Method For Evaluating Rigor and Industrial Relevance of Technology Evaluations,” Empirical Software Engineering, vol. 16, (3), pp. 365 - 395, 2011.

58. A. Carrera and C. A. Iglesias, “A Systematic Review of Argumentation Techniques for Multi-Agent Systems Research,” Artificial Intelligence Review, vol. 44 (4,) pp. 509 - 535, 2015.

59. J. E. Burge, “Design Rationale: Researching Under Uncertainty,” Artificial Intelligence For Engineering Design, Analysis and Manufacturing, vol. 22 (4), pp. 311 - 324, 2008.

60. D. E. Webster, “Mapping the Design Information Representation Terrain,” Computer, vol. 21 (12), pp. 8 - 23, 1988.

61. B. G. Ryder, M. L. Soffa and M. Burnett, “The Impact of Software Engineering Research on Modern Programming Languages,” ACM Transaction on Software Engineering and Methodology (TOSEM), vol. 14 (4), pp. 431 - 477, 2005.

62. W. Emmerich, M. Aoyama and J. Sventek, “The Impact of Research on Middleware Technology,” ACM SIGSOFT Softwre Engineering Notes, vol. 32 (1), 2007.

63. W. Emmerich, M. Aoyama and J. Sventek, “The Impact of Research on Middleware Technology,” ACM SIGOPS Operating Systems Review, vol. 41 (1), 2007.

64. W. Emmerich, M. Aoyama and J. Sventek, “The Impact of Research on the Development of Middleware Technology,” ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 17, (4), pp. 1 - 48, 2008.

65. B. H. C. Cheng and J. M. Atlee, “Research Directional in Requirements Engineering,” 2007 Future of Software Engineering, pp. 285 - 303, May 23 - 25, 2007.

66. E. Carmel, J.Dedrick and K. Kraemer, “Routinizing the Offshore Choice: Applying Diffusion of Innovation to the Case of EDS,” Strategic Outsourcing: An International Journal, vol. 3 (3), pp. 223 - 239, 2009.

67. “Mapping the Values of IoT”, Information Technology Newsweekly, 2019.

68. M. Ivarsson and T. Gorschek, “Technology Transfer Decision Support in Requirements Engineering Research: A Systematic Review of REj,” Requirements Engineering, vol. 14 (3), 2009.

69. J. Yue and M. Harman, “An Analysis and Survey of the Development of Mutation Testing,” IEEE Transactions on Software Engineering, vol. 37 (5), pp. 649 - 678, 2011.

70. H. Petersson, T. Thelin, P. Runeson, and C. Wohlin, “Capture-recapture in Software Inspections After 10 Years Research-Theory Evaluation and Application,” The Journal of Systems & Software, vol. 72 (2), pp. 249 - 264, 2004.

71. M. A. Babar, L. Zhu and R. Jeffery, “A Framework for Classifying and Comparing Software Architecture Evaluation Methods,” 2004 Australian Software Engineering Conference Proceedings, IEEE, pp. 309 - 318, 2004.

72. M. Shaw, “What Makes Good Research in Software Engineering?,” International Journal on Software Tools for Technology Transfer, vol. 4 (1), pp. 1 - 7, 2002.

73. D. Hybertson, W. Duane, A. D. Ta, and W. M. Thomas, “Maintenance of COTS-intensive Software Systems,” Journal of Software Maintenance: Research and Pratice, vol. 9 (4), pp. 203 - 216, 1997.

74. “M. M Chakrabart and D. Mendonca, “Design Considerations for Information Systems to Support Critical Infrastructure Management,” Proccedings of the Second International ISCRAM Conference, pp. 13 - 18, 2005.

75. D. N. Card., “The Role of Measurement in Software Engineering,” Proceedings of ELECTRO ’94, IEEE, pp. 223 - 229, 1994.

76. A. Jackson and D. Hoffman, “Inspecting Module Interface Specifications,” Software Testing, Verification and Reliability,” vol. 4 (2), pp. 101 - 117, 1994.

77. M. V. Zelkowitz, “Software Engineering Technology Infusion Within NASA,” IEEE Transactions on Engineering Management, vol. 43 (3), pp. 250 - 261, 1996.

78. T. Mikkonen, C. Lassenius, T. Manisto, M. Oivo, J. Jarvinen, “Continuous and collaborative technology transfer: Software engineering research with real-time industry impact,” Information and Software Technology, vol. 95, pp. 35 - 45, 2018.

79. T. Mikkonen, C. Lassenius, T. Manisto, M. Oivo, J. Jarvinen, “Can We Influence Students’ Attitudes About Inspections? Can We Measure a Change in Attitude?,” Information and Software Technology, vol. 95, pp. 34 - 45, 2018.

80. L. J. Osterwell, C. Ghezzi, J. Kramer, A. L. Wolf, “The impact: project: determining the impact of software engineering research upon practice” - Impact Project Whitepaper, IEEE Computer, vol. 41 (3), pp. 39 - 49, 2008.

81. C. Scaffidi, “The impact of human-centric design on the adoption of information systems: A case study of the spreadsheet,” 2016 11th Iberian Conference on Information Systems and Technologies (CISTI), pp. 1 - 7, 2016.

82. M. V. Zelkowitz, “Software Engineering Technology Transfer: Understanding the Process,” 1993. www.ntrs.nasa.gov.

83. D. Weyns, “Engineering of Self-Adaptive Systems: An Organized Tour,” IEEE 3rd International Workshops on Foundations and Applications of Self Systems (FASW), pp. 1 - 2, 2018.

84. G. Berg-Cross, “Improving Situational Ontologies to Support Adaptive Crisis Management Knowledge Architecture,” Conference on Information for Crisis, 2008.

85. H. Krasner, “Bottlenecks in the Transfer of Software Engineering Technology: Lessons Learned from a Consortium Failure,” Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences, vol. 4, pp. 635 - 641, 1995.

86. T. Bandyszak, P. Diebold, A. Heuer, T. Kuhn, A. Vetro, T. Weyer, “Technology Transfer Concepts,” Advanced Model-Based Engineering of Embedded System: Extensions of the SPE 2020 Methodology, 2016.

87. D. Wallace and M. Zelkowitz, “Center for High Integrity Software System Assurance,” IFAC Proceedings Volumes, 1995, www.nist.gov.

88. J. Lonchamp, “CS in CSCL,” 13th International Conference on Interactive Computer Aided Learning -ICL 2010, 2010.

89. B. Capps and R. E. Fairley, “PRISM: A Systematic Approach to Planning Technology Transfer Campaigns,” PICMET 03: Portland International Conference on Management of Engineering and Technology Technology Management for Reshaping the World, 2003, IEEE, pp. 393 - 399, 2003.

90. H. Van Vliet, H. Van Vliet and J. D. Van Vliet, *Software Engineering: Principles and Practice,* vol. 13, John Wiley & Sons, Hoboken, NJ, 2008.

91. G. Wang and J. Rice, “Considerations for a Generalized Reuse Framework for System Development,” INCOSE International Symposium, vol. 21 (1), pp. 3278 - 3294, 2011.

92. P. Hantos, “Systems Engineering Perspectives on Technology Readiness Assessments in Software-Intensive System Development,” Journal of Aircraft, vol. 48 (3), pp. 738 - 748, 2011.

93. M. V. Zelkowitz, D. R. Wallace, and D. W. Binkley, “Experimental Validation of New Software Technology,” Lecture Notes on Empirical Software Engineering, pp. 229- 263, 2003.

94. I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, “What Industry Needs from Architectural Languages: A Survey,” IEEE Transactions on Software Engineering, vol. 39 (6), pp. 869 - 891, 2012.

95. A. C. Schierz, “Monitoring Knowledge: A Text-Based Approach Terminology,” International Journal of Theoretical and Applied Issues in Specialized Communication, vol. 13 (2), pp. 125 - 154, 2007.

96. M. Olivier and S. Gruner, “On the Scientific Maturity of Digital Forensics Research,” IFIP Advances in Information and Communication Technology, vol. 410, pp. 33 - 49, 2013.

97. J. Brings, M. Daun, S. Brinckmann, K. Keller and T. Weyer, "Approaches, Success Factors, and Barriers for Technology Transfer in Software Engineering - Results of a Systematic Literature Review," Journal of Software: Evolution and Process, vol. 30 (11), 1981.

98. R. L. Glass, "Formal Methods Are a Surrogate for a More Serious Software Concern," Computer, vol. 29 (4), 1996.

99. J. A. Rader, "Case Adoption: A Process, Not an Event," Advances in Computers, vol. 41, pp. 83 - 156, 1995.

100. R. C. Seacord, "Secure Coding," 2008 Census Bureau Software Process Improvement Conference, SEI Carnegie Mellon University, 2008.

101. W. Collins, K. Miller, B. Spielman, P. Wherry, "How Good is Good Enough? An Ethical Analysis of Software Construction and Use," Communications on ACM, vol. 37 (1), pp. 81 - 91, 1994.

102. https://csrc.nist.gov/CSRC/media/Publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/early-cs-papers-1970-1985.pdf

103. "Processes to Produce Secure Software" - Towards More Secure Software, Vol. 2, National Cyber Security Summit, March 2004. The Software Process Subgroup Within the Task Force on Security Across the Software Development Lifecycle of the Cyber Security Summit – Co-Chaired by Sam Redwine (JMU), Geoff Shively (PivX), and Gerlinde Zibulski (SAP) – produced this report.

104. Saltzer, J.H. and Schroeder, M. D., "The Protection of Information in Computer Systems," Proceedings of the IEEE, vol. 63 (9), pp. 1278 - 1308, 1975.

105. Gaines B., "Modeling and Forecasting the Information Sciences", Elsevier Inc. vol. 57, pp. 3 - 22, 1991.

106. M. V. Zelkowitz, "Assessing Software Engineering Technology Transfer Within NASA. NASA Technical Report NASA-RPT- 003095," National Aeronautics and Space Administration, Washington, DC, 1995.

107. E. M. Rogers, "Diffusion of Innovations," 4th ed. Free Press, New York, 1995.

108. G. McGraw, "Software Security," IEEE Security & Privacy, vol. 2 (2) pp. 80-83, 2004.

109. G. McGraw, "*Software security: building security in*," vol. 1. Addison-Wesley Professional, 2006.

110. A. Ozment and S. E. Schechter, "Milk or Wine: Does Software Security Improve with Age?" USENIX Security Symposium, pp. 93-104, 2006.

111. A. Apvrille and M. Pourzandi, “Secure Software Development by Example,” IEEE Security & Privacy, vol. 3 (4), pp. 10-71, 2005.

112. S. Barnum and G. McGraw, “Knowledge for Software Security,” IEEE Security & Privacy, vol. 3 (2), pp. 74-78, 2005.

113. M. Moriconi, X. Qian, R. A. Riemenschneider, and L. Gong, “Secure Software Architectures,” Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097), pp. 84-93, 1997.

114. A. Bellissimo, J. Burgess and K. Fu, “Secure Software Updates: Disappointments and New Challenges,” HotSec, 2006.

115. Y. Shin and L. Williams, “Is Complexity Really the Enemy of Software Security?” Proceedings of the 4th ACM Workshop on Quality of Protection, ACM, pp. 47-50, 2008.

116. D. Barrera and P. Van Oorschot, “Secure Software Installation on Smartphones. IEEE Security & Privacy, vol. 9 (3), pp. 42-48, 2010.

117. C. Wysopal, L. Nelson, E. Dustin and D. Dai Zovi, “The Art of Software Security Testing: Identifying Software Security Flaws,” Pearson Education, 2006.

118. E. B. Fernandez, “A Methodology for Secure Software Design, “Software Engineering Research and Practice, pp. 130-136, 2004.

119. N. Davis, W. Humphrey and S. T. Redwine, G. Zibulski and G. McGraw, “Processes for Producing Secure Software,” IEEE Security & Privacy, vol. 2 (3), pp. 8-25, 2004.

120. D. Xu and K. E. Nygard, “Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets,” IEEE Transactions on Software Engineering, vol. 32 (4), pp. 265-278, 2006.

121. J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw and N. R. Mead, *Software security engineering*, Pearson, India; 2008.

122. A. K. Ghosh and T. M. Swaminatha, “Software Security and Privacy risks in Mobile E-Commerce,” Communications of the ACM, vol. 44 (2), pp. 51-57, 2001.

123. B. Potter and G. McGraw, “Software Security Testing,” IEEE Security & Privacy, vol. 2 (5), pp. 81-85, 2004.

124. M. Dalton, H. Kannan and C. Kozyrakis, “Raksha: a Flexible Information Flow Architecture for Software Security,” ACM SIGARCH Computer Architecture News. vol. 35 (2), pp. 482-493, 2007.

125. K. Tsipenyuk, B. Chess and G. McGraw, "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors," IEEE Security & Privacy, vol. 3 (6), pp. 81-84, 2005.

126. M. Howard, D. LeBlanc and J. Viega, "19 Deadly Sins of Software Security. Programming Flaws and How to Fix Them," 2005, pdf.

127. J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way* (Paperback), Addison-Wesley Professional, 2011.

128. A. Takanen, J. D. Demott, C. Miller and A. Kettunen, "Fuzzing for Software Security Testing and Quality Assurance," Artech House, 2018.

129. U. Erlingsson, "Low-Level Software Security: Attacks and Defenses," Foundations of Security Analysis and Design vol. IV, pp. 92-134, 2007.

130. H. Li, P. Li, S. Guo and A. Nayak, "Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers in Cloud," IEEE Transactions on Cloud Computing, vol. 2 (4), pp. 436-447, 2014.

131. D. Perito and G. Tsudik, "Secure Code Update for Embedded Devices Via Proofs of Secure Erasure," European Symposium on Research in Computer Security, pp. 643-662, 2010.

132. D. K. Nilsson, T. Roosta, U. Lindqvist and A. Valdes, "Key Management and Secure Software Updates in Wireless Process Control Environments," Proceedings of the First ACM Conference on Wireless Network Security, pp. 100-108, 2008.

133. B. De Win, F. Piessens, W. Joosen and T. Verhanneman, "On the Importance of the Separation-of-Concerns Principle in Secure Software Engineering, "Workshop on the Application of Engineering Principles to System Security Design, pp. 1-10, 2002.

134. D. Kleidermacher and M. Kleidermacher, "Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development," Elsevier; 2012.

135. T. August and T. I. Tunca, "Let the Pirates Patch? An Economic Analysis of Software Security Patch Restrictions," Information Systems Research, vol. 19 (1), pp. 48-70, 2008.

136. T. Doan, S. Demurjian, T. C. Ting and A. Ketterl, "MAC and UML For Secure Software Design," Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering, pp. 75-85, 2004.

137. B. Livshits, "Improving Software Security with Precise Static and Runtime Analysis," 2006.

138. M. Howard and S. Lipner, *The Security Development Lifecycle*, Redmond, WA, Microsoft Press, 2006.

139. S. Lipner, “The Trustworthy Computing Security Development Lifecycle,” 20th Annual Computer Security Applications Conference, IEEE, pp. 2 - 13, 2004.

140. J. Zambreno, A. Choudhary, R. Simha, B. Narahari, N. Memon and N. Memon, “SAFE-OPS: An Approach to Embedded Software Security,” ACM Transactions on Embedded Computing Systems (TECS), vol. 4 (1), pp. 189 - 210, 2005.

141. Y. Desmedt, “Securing Traceability of Ciphertexts—Towards a Secure Software Key Escrow System,” International Conference on the Theory and Applications of Cryptographic Techniques, Springer, pp. 147 - 157, 1995.

142. K. Kato and Y. Oyama, “Softwarepot: An Encapsulated Transferable File System for Secure Software Circulation,” International Symposium on Software Security, Springer, pp. 112 - 132, 2002.

143. S. Ardi, D. Byers and N. Shahmehri, “Towards a Structured Unified Process for Software Security,” Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems, ACM, pp. 3 - 10, 2006.

144. J. Deng, R. Han, S. Mishra, “Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks,” Proceedings of the 5th International Conference on Information Processing in Sensor Networks, ACM, pp. 292 - 300, 2006.

145. J. Epstein, S. Matsumoto and G. McGraw, “Software Security and SOA: Danger, Will Robinson!” IEEE Security & Privacy, vo. 4 (1), pp. 80 - 83, 2006.

146. M. Gegick, L. Williams, J. Osborne and M. Vouk, “Prioritizing Software Security Fortification Throughcode-level Metrics,” Proceedings of the 4th ACM Workshop on Quality of Protection, ACM, pp. 31 - 38, 2008.

147. B. De Win, R. Scandariato, K. Buyens, and J. Grégoire and W. Joosen, “On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared,” Information and software technology, vo. 51 (7), pp. 1152 - 1171, 2009.

148. X. Zhang and R. Gupta, “Hiding Program Slices for Software Security,” Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization, IEEE Computer Society, pp. 325 - 336, 2003.

149. B. Potter and G. McGraw, “Software Security Testing,” IEEE Security & Privacy, vol. 2 (5), pp. 81 - 85, 2004.

150. D. P. Gilliam, T. L. Wolfe, J. S. Sherif and M. Bishop, “Software Security Checklist for The Software Life Cycle. In WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, pp. 243 - 248, 2003.

151. G. McGraw, “From the Ground Up: The DIMACS Software Security Workshop,” IEEE Security & Privacy, Vol. 1 (2), pp. 59 - 66, 2003.

152. M. Howard and D. LeBlanc, *Writing Secure Code*, Pearson Education, 2003.

153. D. Avots, M. Dalton, V. B. Livshits and M. S. Lam, “Improving software security with a C pointer analysis,” Proceedings of the 27th international conference on Software engineering, ACM, pp. 332 - 341, 2005.

154. C. Heitmeyer, M. Archer, E. Leonard and J. McLean, “Applying Formal Methods to a Certifiably Secure Software System,” IEEE Transactions on Software Engineering, vol. 34, (1), pp. 82 - 98, 2008.

155. M. Dowd, J. McDonald and Schuh J. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities,* Pearson Education, 2006.

156. T. August and T. I. Tunca, “Network Software Security and User Incentives,” Management Science, vol. 52 (11), pp. 1703 - 1720, 2006.

157. C. Cowan, “Software Security for Open-Source Systems,” IEEE Security & Privacy, vol. 1(1), pp. 38 - 45, 2003.

158. “Life in the Crosshairs: The Dawn of the Microsoft Security Development Lifecycle,” Microsoft Press, Microsoft Corporation, Redmond, Washington, 2014.