Applying 1970 Waterfall Lessons Learned Within Today's Agile Development Process

Johnny D. Morgan, PhD
General Dynamics Information Technology; johnny.morgan@gdit.com

ABSTRACT

While working for TRW in 1970, Dr. Winston Royce, published an IEEE paper that described the waterfall development process. In his paper he states "I believe in the concept but the implementation described is risky and invites failure." He then "presents five additional features that must be added to this basic approach to eliminate most of the development risks." This paper reviews these five additional features recommended in 1970 and describes how they are incorporated into today's agile development process to reduce agile project development risks.

Key Words: Agile, architecture, design, Manifesto for Agile Software Development, project management, software development, waterfall, Winston Royce

This paper is based on empirical observations, current literature, and engineering and project management experiences.

INTRODUCTION

Since the publishing of the Manifesto for Agile Software Development in 2001, many books and journal articles have been published that first characterizes the waterfall software development process and then identifies process steps that attempt to separate and uniquely characterize agile development practices as different and better. In many cases, each of these publications either references or directly draws the basic waterfall development process shown in Figure 1 and then attempts to make a comparison of historical project management concepts and agile project management concepts. For example, one publication states:

> "In the historical approach, which locks the requirements and delivers the product all in one go, the result is all or nothing. We either succeed completely or fail absolutely. The stakes are high because everything hinges on work that happens at the end...of the final phase of the cycle, which includes integration and customer testing...In the testing phase of a waterfall project, the customers get to see their long-awaited product. By that time, the investment and effort have been huge, and the risk of failure is high. Finding defects among all completed project requirements is like looking for a weed in a cornfield…The following list summarizes the major aspects of the waterfall approach to project management:
>
> - The team must know all requirements up front to estimate time, budgets, team members and resources…
> - The customer and stakeholders may not be available to answer questions during the development period, because they may assume that they

provided all the information during the requirements-gathering and design phases…

- The team needs to resist the addition of new requirements or document them as change orders, which adds more work to the project and extends the schedule and budget…
- Full and complete customer feedback is not possible until the end of the project when all functionality is complete." (Layton and Ostermiller 2017)

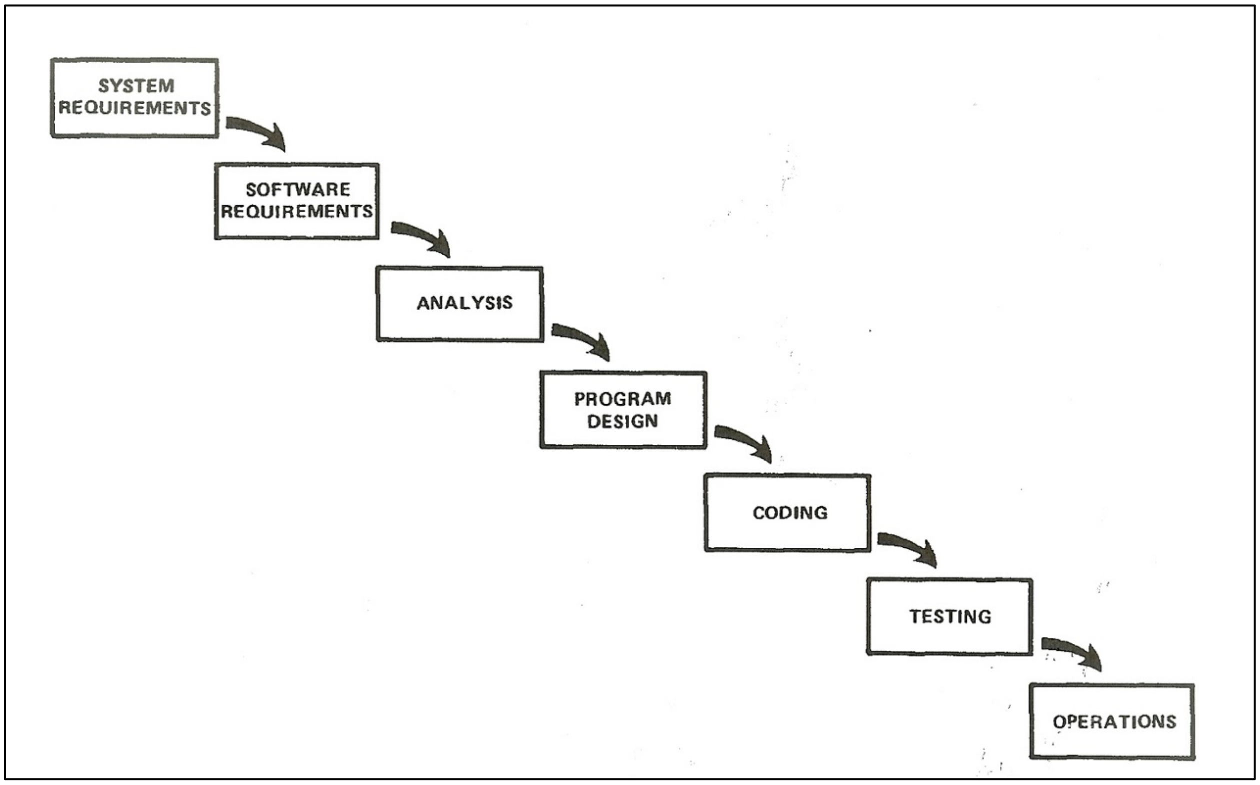


Figure 1 – The Basic Waterfall Development Methodology (Royce 1970)

In another publication, the abbreviated comparison shown in Table 1 is made to distinguish the characteristics between agile and non-agile development methodologies.

Table 1: Abbreviated Comparison of Agile versus Non-Agile Methodologies (Schuh 2006)

Project Environment Variable	Agile	Non-Agile
Communication Style	Regular Collaboration	Only When Necessary
Continuous Learning	Embraced	Discouraged
Team Participation	Mandatory	Unwelcome
Planning	Continuous	Up Front
Feedback Mechanisms	Several	Not Available
Customer Involvement	Throughout the Project	During Analysis Phase

The diagram in Figure 1 was published in 1970 by Dr. Winston W. Royce in a paper entitled “Managing the Development of Large Software Systems.” It is actually one of ten figures within the paper but the diagram in Figure 1 is the most cited. As one author now states:

> “Unfortunately, Royce’s paper was widely misunderstood. He presented the above model (Figure 1) as ‘**risky and invites failure**’ and was proposing modifications to make it much more iterative and incremental. However, that element of his work is largely forgotten, and his waterfall picture remains in common use, although the names of the stages have changed a little over time.” (Girvan and Paul 2017)

This paper examines Royce’s paper and proposed modifications and discusses how they are now applied to agile development practices. However, it is important to understand that the steps identified in this paper are highly relevant to all software development methodologies.

INTRODUCING WILLIAM ROYCE AND HIS PAPER

Wikipedia states that Winston W. Royce was born in 1929 and went to college at the California Institute of Technology where he received a Bachelor of Science in physics, and both a Master of Science and Doctor of Philosophy in aeronautical engineering. In 1961 he started as a project manager in the aerospace division of TRW. Having managed multiple large, complex software development projects, he published the paper “Managing the Development of Large Software Systems” in 1970. In the 1980s, he became a Director at the Lockheed Software Technology Center in Austin, Texas. He retired in 1994 and died the following year. (Anonymous2017)

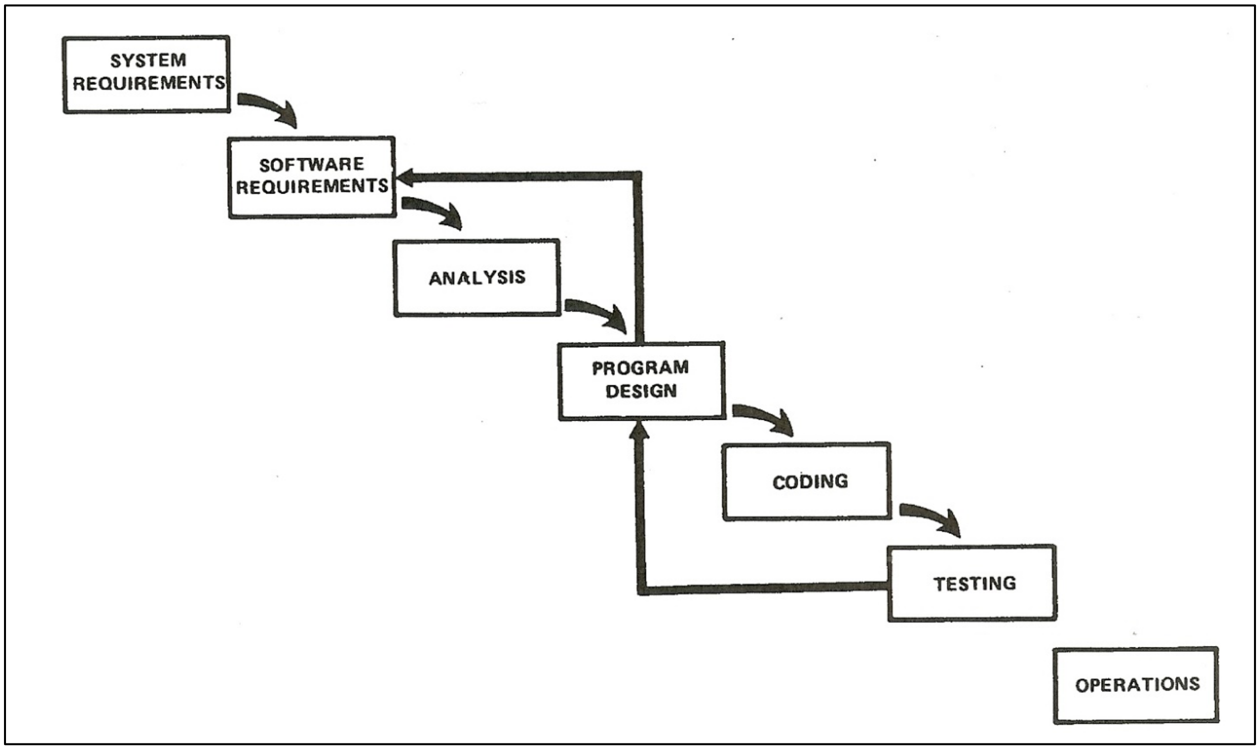


Figure 2: If Testing is the First Event for Assessing Key Elements (Royce 1970)

In this paper, Royce states:

> "I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years...In these assignments, I have experienced different degrees of success…I have become prejudiced by my experiences and am going to relate some of these prejudices in this presentation…(An) approach to software development is illustrated in (Figure 1). I believe in this concept but the implementation is risky and invites failure. The problem is illustrated in (Figure 2). (If) the testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced…the required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect, the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs…However, I believe the illustrated approach to be fundamentally sound. The rest of this discussion presents five additional features that must be added to this basic approach to eliminate most of the development risks." (Royce 1970)

ROYCE WAS A LEAN THINKER

Nearly all agile practices are based on Lean thinking, many dating back to W. Edwards Deming's work in Japan in the 1950s. For example, today's Scaled Agile Framework (SAFe)® provides multiple sections on the development and usage of Lean thinking in the execution of this agile framework. (Leffingwell and others 2017)

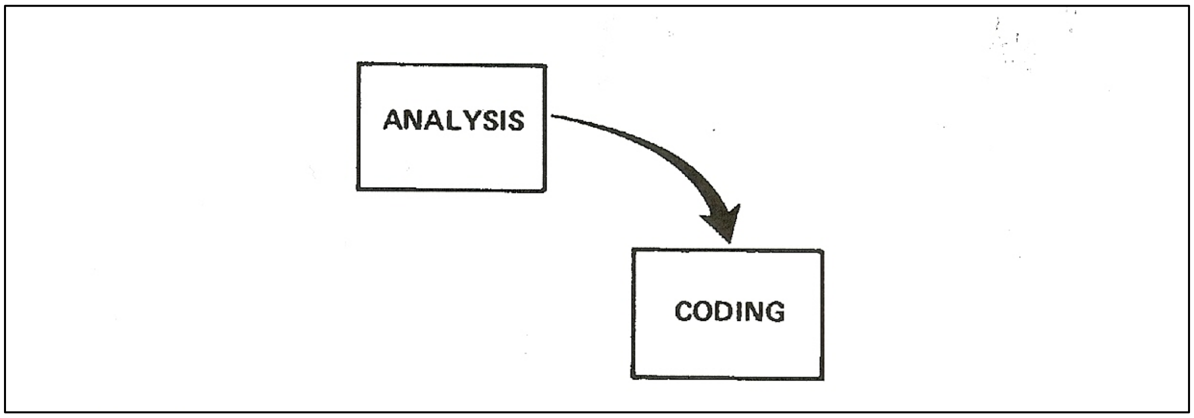


Figure 3: Two Step Process Described in Royce's Paper (Royce 1970)

Dr. Royce also discussed Lean thinking in his paper:

> "There is first an analysis step, followed by a coding step as depicted in (Figure 3). This sort of very simple implementation is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it…It is also the kind of development effort for which most customers are

happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product."

But Dr. Royce then provides a warning to project managers:

> "An implementation plan to manufacture larger software systems, and keyed only to these steps, however, is doomed to failure. Many additional steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development costs. Customer personnel typically would rather not pay for them, and development personnel would rather not implement them. The prime function of management is to sell these concepts to both groups and then enforce compliance on the part of development personnel." (Royce 1970)

INTERACTIVE INTERSECTION BETWEEN THE VARIOUS PHASES

Today's agile development practices employ short development cycles to:

- Incorporate customer feedback on the product
- Incorporate lessons learned from within the development team
- Allow for the insertion and prioritization of new customer requirements and priorities into the development process

In some cases, agile teams use the term "fail fast" to describe the process for incorporating these traits into the software development process.

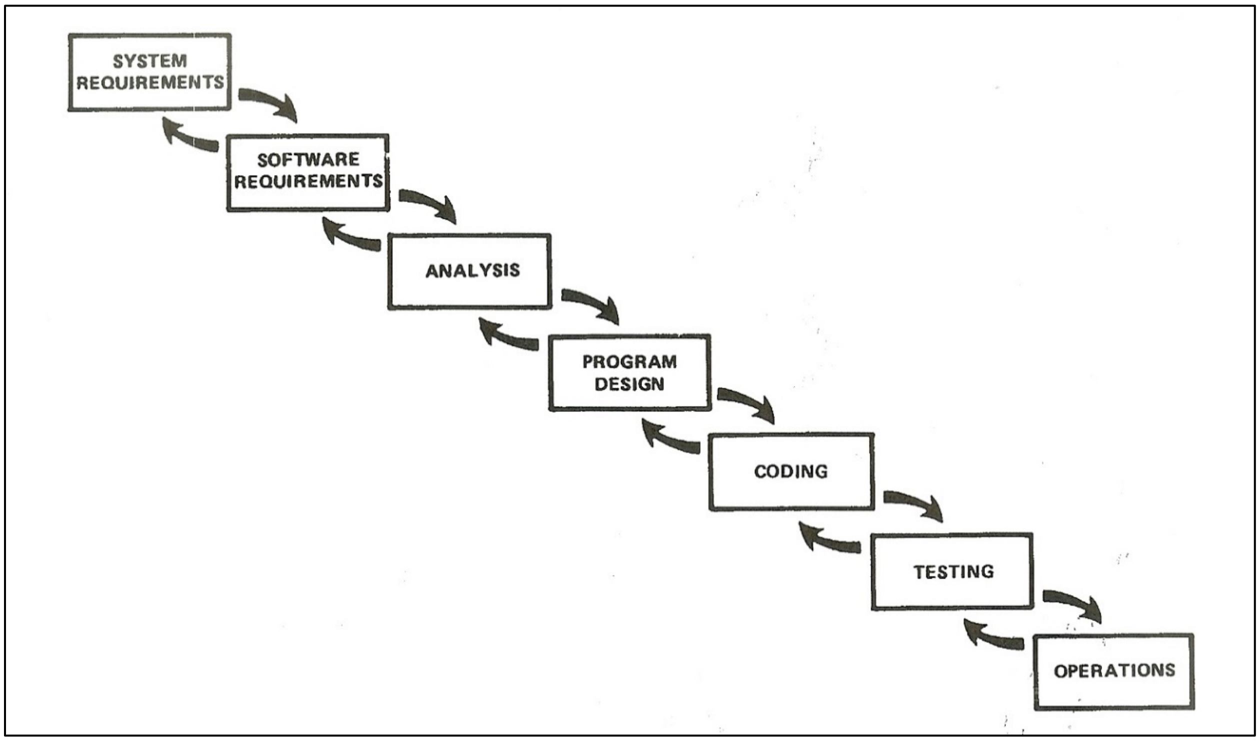


Figure 4: Iterative Relationship Between Successive Development Phases

Equally important to a waterfall development process is the iterative relationship between successive development phases as shown in Figure 4. As described by Royce:

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

Based on personal experience, this author has found that an effective change process is key to any organization and supporting software development process being agile or responsive. This change process incorporates the concept of moving, or versioning baselines in which to return to in the event of unforeseen difficulties and includes the ability to incorporate changes in a timely manner resulting from:

- Changing requirements from any stakeholder source
- Results from modeling, simulation, prototyping and user feedback
- Inability to translate paper-based concepts into working, performant software

STEP 1: PROGRAM DESIGN COMES FIRST

The first feature that Royce recommends is to complete a high-level program design of the system. He states:

> "The following steps are required.
>
> 1) Begin the design process with program designers, not analysts or programmers.
> 2) Design, define and allocate the data processing modes even at the risk of being wrong…
> 3) Write an overview document that is understandable, informative and current. Each and every worker must have an elemental understanding of the system. At least one person must have a deep understanding of the system which partially comes from having to write an overview document."

In today's terminology, a high-level program design refers to establishing the solution architecture for a system and to accomplish this before any major software coding efforts commence as shown in Figure 5.

The agile manifesto has one value statement and two principles that affect architecture. One states that "the best architectures, requirements, and designs emerge from self-organizing teams" while the other two state that the team should welcome and respond to change for the customer's competitive advantage. (Anonymous2001)

Implementing these elements without applying energy to architecture can have serious consequences. As Roger Sessions states:

> "Architectures naturally seek the maximum possible level of complexity all on their own. If it is a complex architecture you are after, you don't need architects. You might as well just fire them all and let developers work on their own. This observation that architectures are naturally attracted to complexity is actually predicted by physics—in particular, the law of entropy. This fundamental law of physics states that left to their own, all systems evolve into a state of maximal disorder (entropy). It takes a constant inflow of energy into a system to keep the disorder at bay. In this regard, enterprise architectures are just another natural system, like gas molecules in a balloon. The law of entropy tells us that the battle for simplicity is never over. It requires a constant influx of energy to keep enterprise systems simple. It isn't enough to design them so that they are simple. It isn't enough to even build them so that they are simple. You must continue working to prevent an erosion of simplicity for the life of the system. In this sense, the work of the enterprise architect is never done." (Sessions 2008)

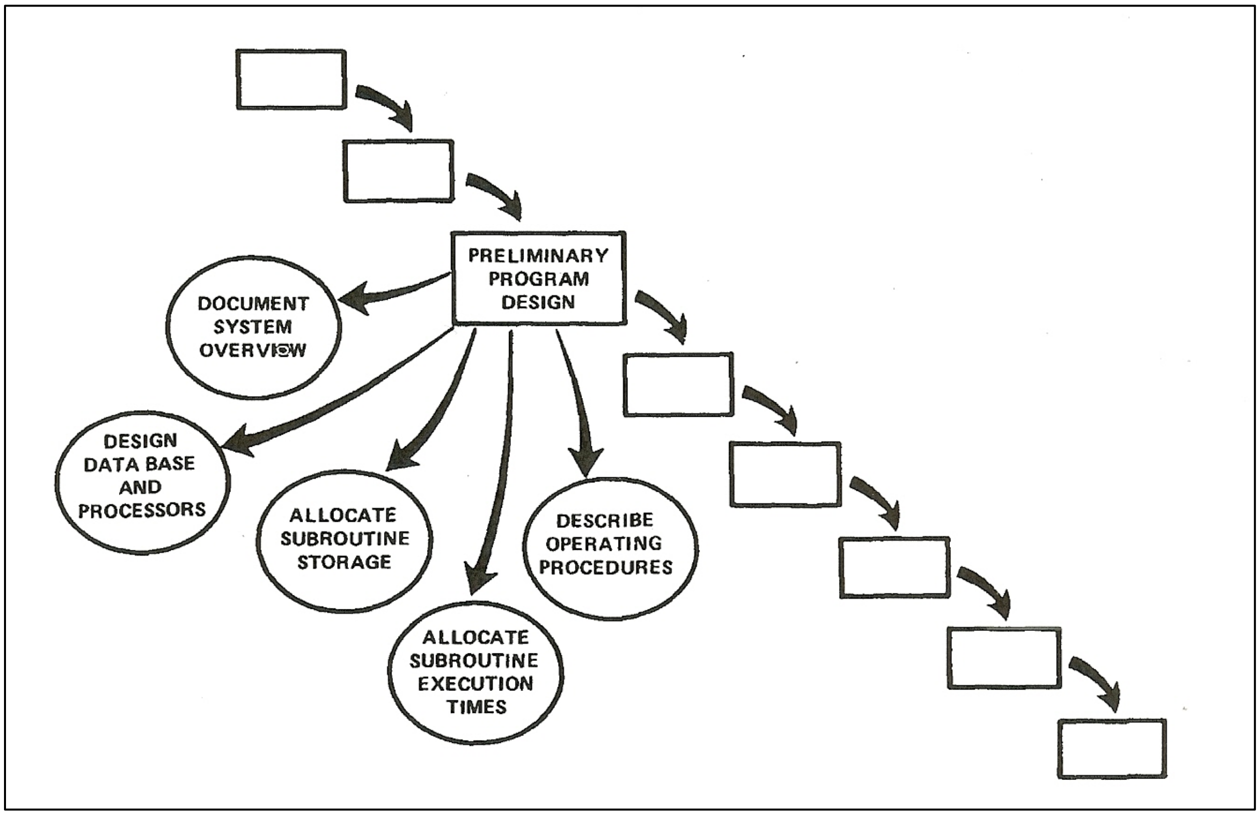

PRELIMINARY PROGRAM DESIGN
DOCUMENT SYSTEM OVERVIEW
DESIGN DATA BASE AND PROCESSORS
ALLOCATE SUBROUTINE STORAGE
ALLOCATE SUBROUTINE EXECUTION TIMES
DESCRIBE OPERATING PROCEDURES

Figure 5: Complete a High-Level Design of the System

Successfully incorporating architecture into agile projects is a difficult balancing act.

> "Companies where architectural practices are well developed often tend to see agile practices as amateurish, unproven, and limited to very small, Web-based sociotechnical systems. Conversely, proponents of agile approaches usually see

> little value for a system's customers in the upfront design and evaluation of architecture. They perceive architecture as something from the past, equating it with big design up-front (BDUF)—a bad thing—leading to massive documentation and implementation of YAGNI (you ain't gonna need it) features. They believe that architectural design has little value, that a metaphor should suffice in most cases, and that the architecture should emerge gradually sprint after sprint, as a result of successive small refactoring…The tension seems to lie on the axis of adaptation versus anticipation. Agile methods want to be resolutely adaptive: deciding at the 'last responsible moment' or when changes occur. Agile methods perceive software architecture as pushing too hard on the anticipation side: planning too much in advance." (Abrahamsson, Babar, and Kruchten 2010, 16-22)

Grady Booch provides insight into the birth and maturing of many systems and the need for architecture:

> There are many examples of notable systems that began with the code of one or two people and grew to become a dominant design: the packet-switched multiple-protocol router, first developed by Bill Yeager; a graphics editing system, first developed by Thomas and John Knoll; a social network, first popularized by Mark Zuckerberg. The list goes on. In each of these cases, architecture was not a primary concern. I'd be surprised if it was on their radar at all, save for the reality that each of these developers had the chops, the experience, and the intuition to deliver something Good Enough that could be grown...Quite often, the developers who did the internal exploration are not the most skilled at production. Furthermore, the risk profile changes, and the success of a system is less dependent on rapid innovation and much more dependent on quality and efficiency in manufacturing and delivery…it's also these times that intentional architecting becomes intensely important." (Booch 2011, 10-11)

The following discussion provides insight into the application architecture processes into an agile activity:

> "Do not dilute the meaning of the term architecture by applying it to everything in sight. Not all design decisions are architectural…(apply architecture) early enough because architecture encompasses the set of significant decisions about the structure and behavior of the system. These decisions prove to be the hardest to undo, change, and refactor, which means to not only focus on architecture, but also interleave architectural 'stories' and functional 'stories' in early iterations...User stories in agile development relate primarily to functional requirements; this means that nonfunctional requirements can sometimes be completely ignored. Unfulfilled nonfunctional requirements can make an otherwise functioning system useless or risky. A main objective of integrating architectural approaches in agile processes is to enable software development teams to pay attention to both functional and nonfunctional requirements." (Abrahamsson, Babar, and Kruchten 2010, 16-22)

The Scale Agile Framework ® incorporates the concept of Architecture Runway in its agile practice. The framework states:

> "Architectural runway provides one of the means by which SAFe implements the concepts of Agile architecture. The runway provides the necessary technical basis for developing business initiatives and implementing new features and capabilities. An architectural runway exists when the enterprise's platforms have sufficient technological infrastructure to support the implementation of the highest priority, near-term features without excessive, delay-inducing redesign." (Leffingwell and others 2017)

STEP 2: DOCUMENT THE DESIGN

Royce's second feature or step is to document the design. In describing this feature, Royce provides the following insights:

> "At this point it is appropriate to raise the issue – 'how much documentation?' My own view is "quite a lot" certainly more than most programmers, analysts, or programmers are willing to do if left to their own devices…Why so much documentation?
>
> > 1) Each designer must communicate with interfacing designers, with his management and possibly with the customer. A verbal record is too intangible to provide an adequate basis for an interface or management decision…
> > 2)…If the documentation does not exist there is as yet no design, only people thinking and talking about the design which is of some value, but not much…
> > 3) The real monetary value of good documentation begins downstream in the development process during the testing phase and continues through the operations and redesign phase." (Royce 1970)

As shown in Figure 6, Royce recommends six documents that are developed and updated throughout the system's life cycle.

In reviewing the current literature, the phrase "comprehensive documentation" is directly associated with the concept of big up-front design (BUFD) where all elements of a system's design are completely thought out and documented before any software coding begins. This massive amount of documentation then quickly falls out of sync soon after development starts (Erdogmus 2009, 2-4). As previous discussed, agile teams need to be concerned about the system's architecture but "what is architecture" versus "what is design" may not be obvious.

> "What does a particular project or organization mean by architecture? The concept has fuzzy boundaries. In particular, not all design is architecture. Agreeing on a definition is a useful exercise and a good starting point...Do not dilute the meaning of the term architecture by applying it to everything in sight. Not all design decisions are architectural. Few are, actually, and in many projects, they're already made on day one." (Abrahamsson, Babar, and Kruchten 2010, 16-22)

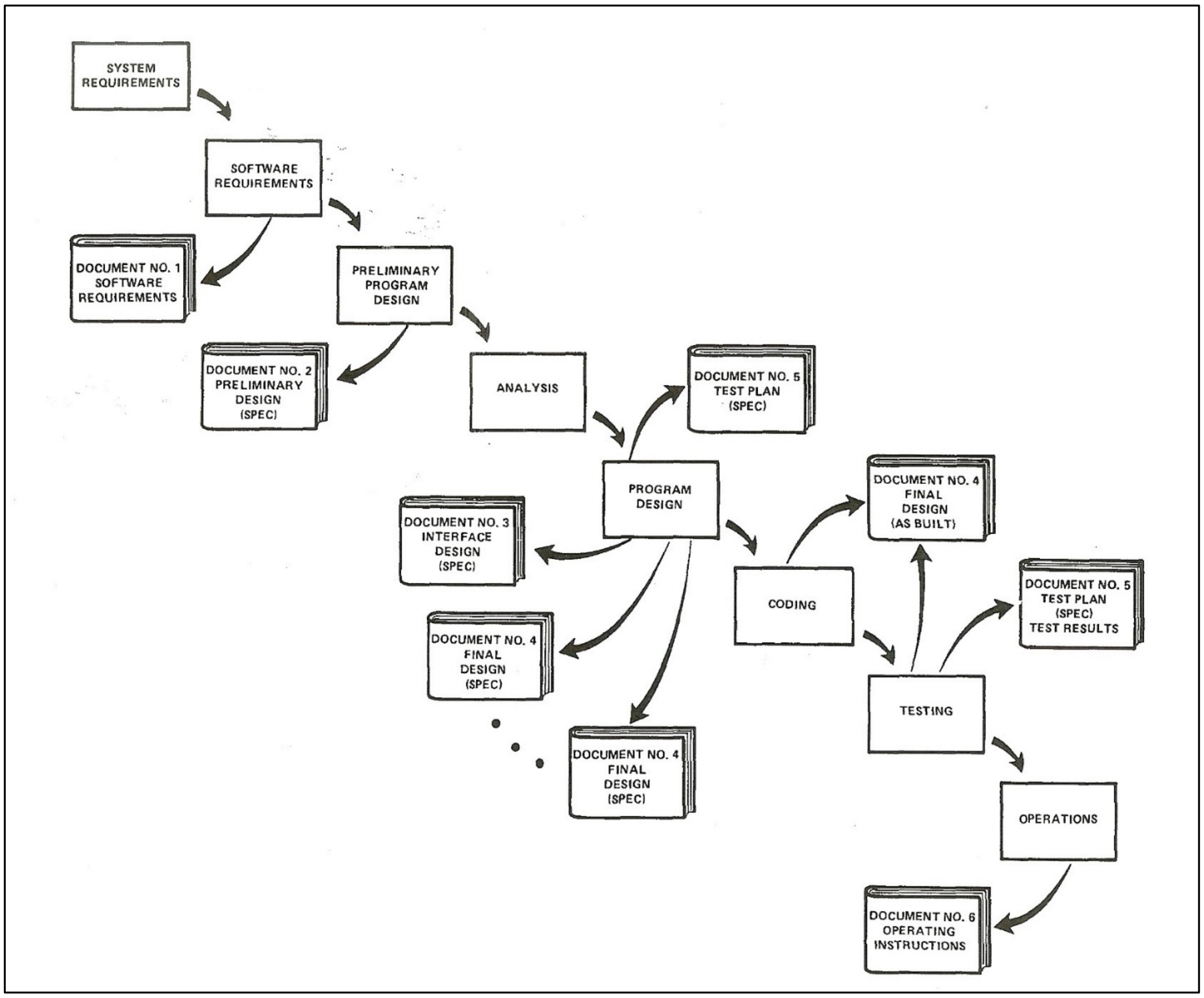

SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
ANALYSIS
DOCUMENT NO. 5 TEST PLAN (SPEC)
PROGRAM DESIGN
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
CODING
DOCUMENT NO. 4 FINAL DESIGN (AS BUILT)
DOCUMENT NO. 5 TEST PLAN (SPEC) TEST RESULTS
TESTING
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS

Figure 6: Royce Recommends Six Documents for a System (Royce 1970)

Grady Booch offers these insights:

> "As I've often said, the code is the truth, but not the whole truth, meaning that there are certain architectural decisions that cannot easily be discerned in the code itself. This is so because such decisions are manifested as mechanisms that are either scattered or tangled throughout the code, their meaning and presence are in the heads of the code's creators and not easily evident by staring at it (the code)...It's these bits of architectural decisions that are best documented elsewhere, external to the code base. Such decisions often live in tribal memory, in the heads of people. This is fine when the team is small, but when the system grows to economic significance, tribal memory is a particularly noisy and inefficient repository of architecture.
>
> The architectural mechanisms that are not baked into the code and thus are in the heads are the things you want to (a) take time to document and, where possible,

> (b) create a domain-specific language that is baked into the code to implement it. My experience is that every reasonably software-intensive system will have a couple dozen such architectural mechanisms…These are the kinds of decisions that can be documented in a static document of two or three dozen pages—any longer and no one will read it…this artifact becomes a vehicle for orienting new folks to the code based as well as attending to some degree of architectural governance, whose simple goal is getting people to continue to grow the system according to those architecture principles." (Booch 2011, 10-11)

Alistair Cockburn shares similar advice:

> "the designer's job is not pass along 'the design' but to pass along 'the theories' driving the design. The latter goal is more useful and more appropriate. It also highlights that knowledge of the theory is tacit in the owning, and so passing along the theory requires passing along both explicit and tacit knowledge."

Cockburn promotes a Theory Building View of a system and then summarizes by providing these recommendations for what should be put into documentation:

> "*That which helps the next programmer build an adequate theory of a program.* This is enormously important. The purpose of the documentation is to jog memories in the reader, set up relevant pathways of thought about experiences and metaphors. This sort of documentation is more stable over the life of the program than just naming the pieces of system currently in place. The designers are allowed to use whatever forms of expression are necessary to set up those relevant pathways…Experienced designers often start their documentation with just:
>
> - The metaphors
> - Text describing the purpose of each major component
> - Drawings of the major interactions between the major components
>
> These three items alone take the next team a long way to constructing a useful theory of the design…Documentation cannot—and so need not—say everything. Its purpose is to help the next programmer build an accurate theory about the system." (Cockburn 2007)

For a client who had incurred tremendous amounts of technical debt caused by the absence of credible explicit knowledge and about his technical systems, the author was asked to identify, in priority order, those items that need to be explicitly described and the following was recommended:

1. Any knowledge that defines to users, operators, and maintainers how to operate and maintain the system
2. Any knowledge that describes how to rebuild and redeploy all of the system, should a disaster occur
3. Any knowledge that is used to verify that the system has been successfully rebuilt and redeployed such that it can again support the business
4. Any knowledge that allows future personnel to modify the system over its life cycle

STEP 3: DO IT TWICE

Both Royce and today's agile methods agree that incorporating knowledge and feedback into the development process is essential. Many of today's agile practices identify the term Minimum Viable Product (MPV) which is a minimum operationally acceptable product that a customer can field and use in operations. Furthermore, many agile processes describe the process of "failing fast," to denote the concept of building something quickly to see if it works and then incorporating the knowledge gained by successes and failures back into the agile development process. Figure 7 shows the diagram Royce used in his paper and he describes this feature or step as follows:

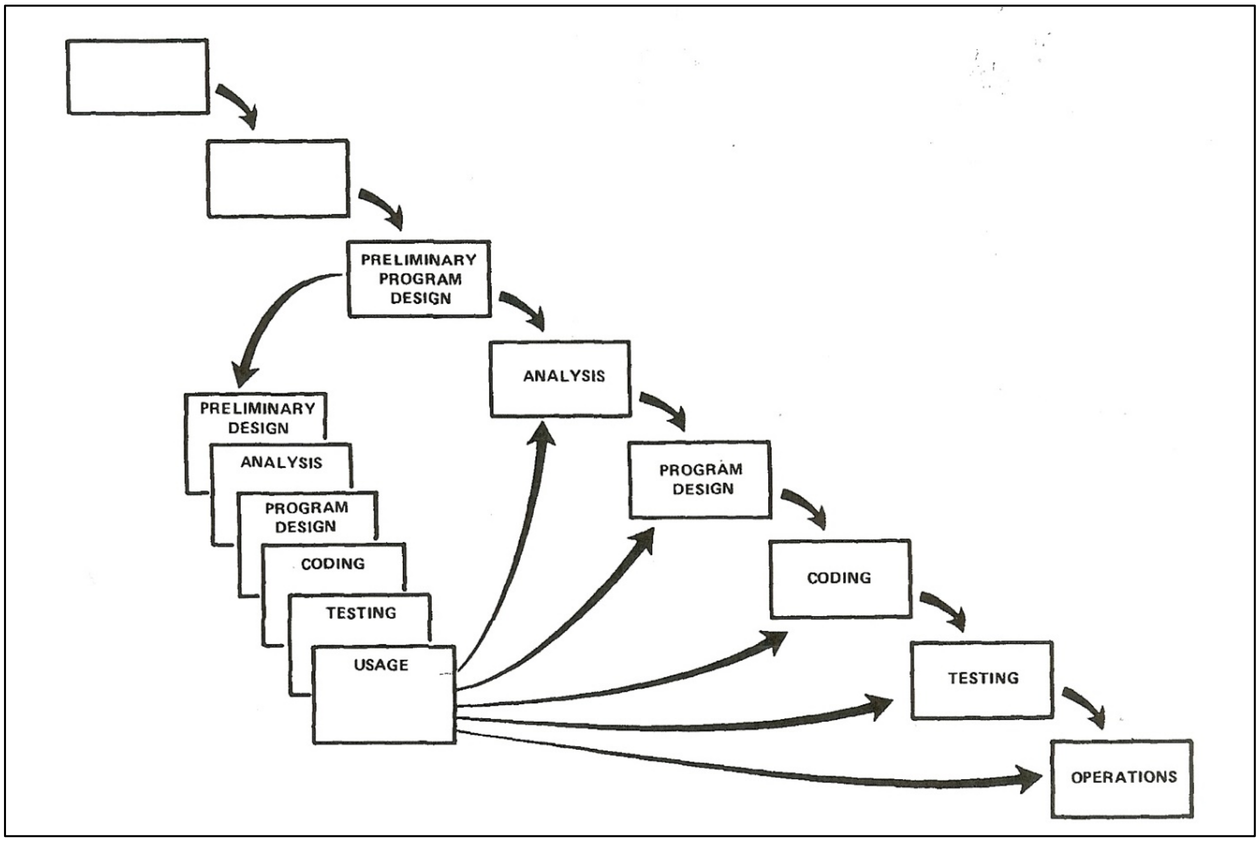


Figure 7: Build the System Twice (Royce 1970)

> "If the computer program in question is being developed for the first time, arrange matters for the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned…Figure 7 illustrates how this may be carried out by means of a simulation…This pilot effort could be compressed…in order to gain sufficient leverage to the mainline effort. In this case a very special kind of broad competence is required on the part of the personnel involved. They must have an intuitive feel for analysis, coding, and program design. They must quickly sense the trouble spots in the design, model them, model their alternatives, forget the straight forward aspects of the design which aren't worth studying at this early

point, and finally arrive at an error-free program…Without this simulation the project manager is at the mercy of human judgement." (Royce 1970)

STEP 4: PLAN, CONTROL AND MONITOR TESTING

Of the five features discussed by Royce, feature or step four is to plan, control and monitor testing. Royce provides the diagram in Figure 8 and writes:

> "Without question the biggest user of project resources, whether it be manpower, computer time, or management judgement is the test phase. It is the phase of greatest risk in terms of dollars and schedule. It occurs at the latest point in the schedule when backup alternatives are least available, if at all. The previous three recommendations to design the program before beginning analysis and coding, to document it completely, and to build a pilot model are all aimed at uncovering and solving problems before entering the test phase. However even after doing these things there is still a test phase and there are still important things to be done." (Royce 1970)

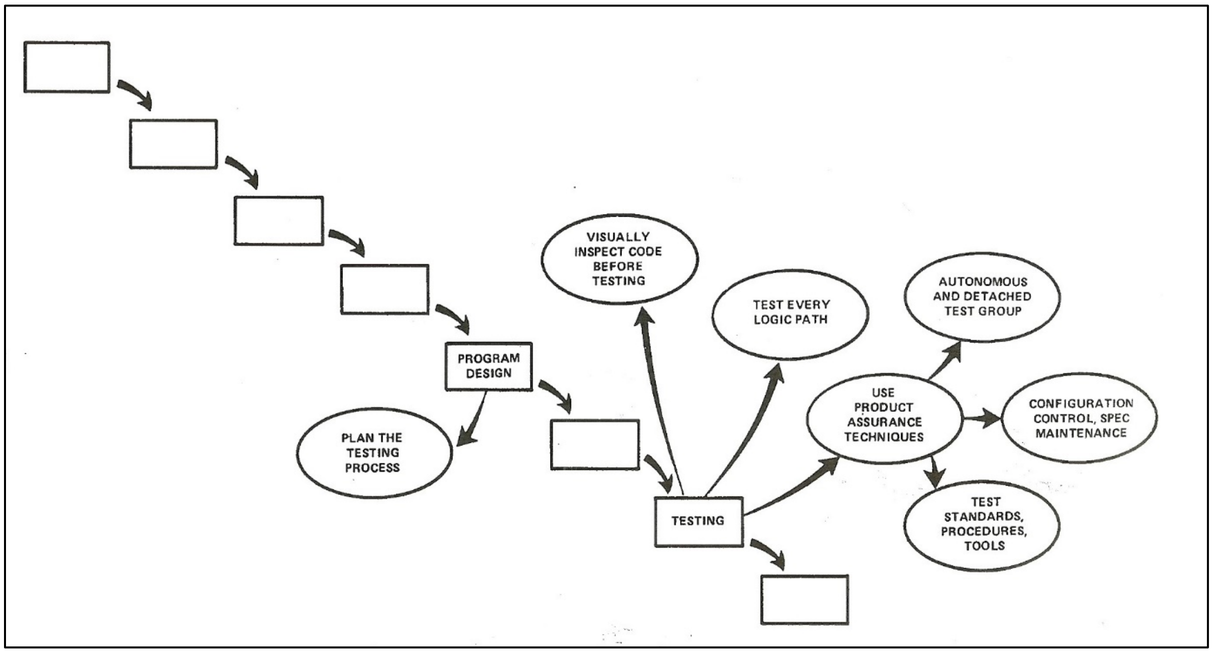

PROGRAM DESIGN
PLAN THE TESTING PROCESS
VISUALLY INSPECT CODE BEFORE TESTING
TEST EVERY LOGIC PATH
TESTING
USE PRODUCT ASSURANCE TECHNIQUES
AUTONOMOUS AND DETACHED TEST GROUP
CONFIGURATION CONTROL, SPEC MAINTENANCE
TEST STANDARDS, PROCEDURES, TOOLS

Figure 8: Recommendations for Planning, Controlling and Monitoring Testing (Royce 1970)

Many agile methods, such as Test-Driven Development (TDD) focus on writing the test first, before actually writing the software. As Royce states:

> "Many parts of the test process are best handled by test specialists who did not necessarily contribute to the original design. If it is argued that only the designer can perform a thorough test because only he understands the area he built, this is a sure sign of failure to document properly. With good documentation it is feasible to use specialists in software product assurance who will, in my judgment, do a better job of testing than the designer." (Royce 1970)

Some agile methods include the concept of Pair Programming which has one person writing the code while another person is sitting by the coder and reviewing his work and enhancing it through an interactive dialog. Then periodically, the two software engineers switch roles, but both continue writing and reviewing the software as a team. Royce writes:

> "Most errors are of an obvious nature that can be easily spotted by visual inspection. Every bit of the analysis and every bit of code should be subjected to a simple visual scan by a second party" (Royce 1970)

Royce also recommends to "test every logic path in the computer program at least once with some kind of numerical check." In 1970 when his paper was written, this was a very difficult and time-consuming recommendation, but today's software development tools provide mechanisms for tracking code coverage during the execution of a test procedure. Royce also states "while this (recommendation) sounds simple, for a large, complex computer program it is relatively difficult to plow through every logic path with controlled values of input. In fact some would argue that is every nearly impossible." (Royce 1970) Today's automated testing tools support the low cost, re-execution of test procedures to ensure that existing software is not broken while new software is being developed.

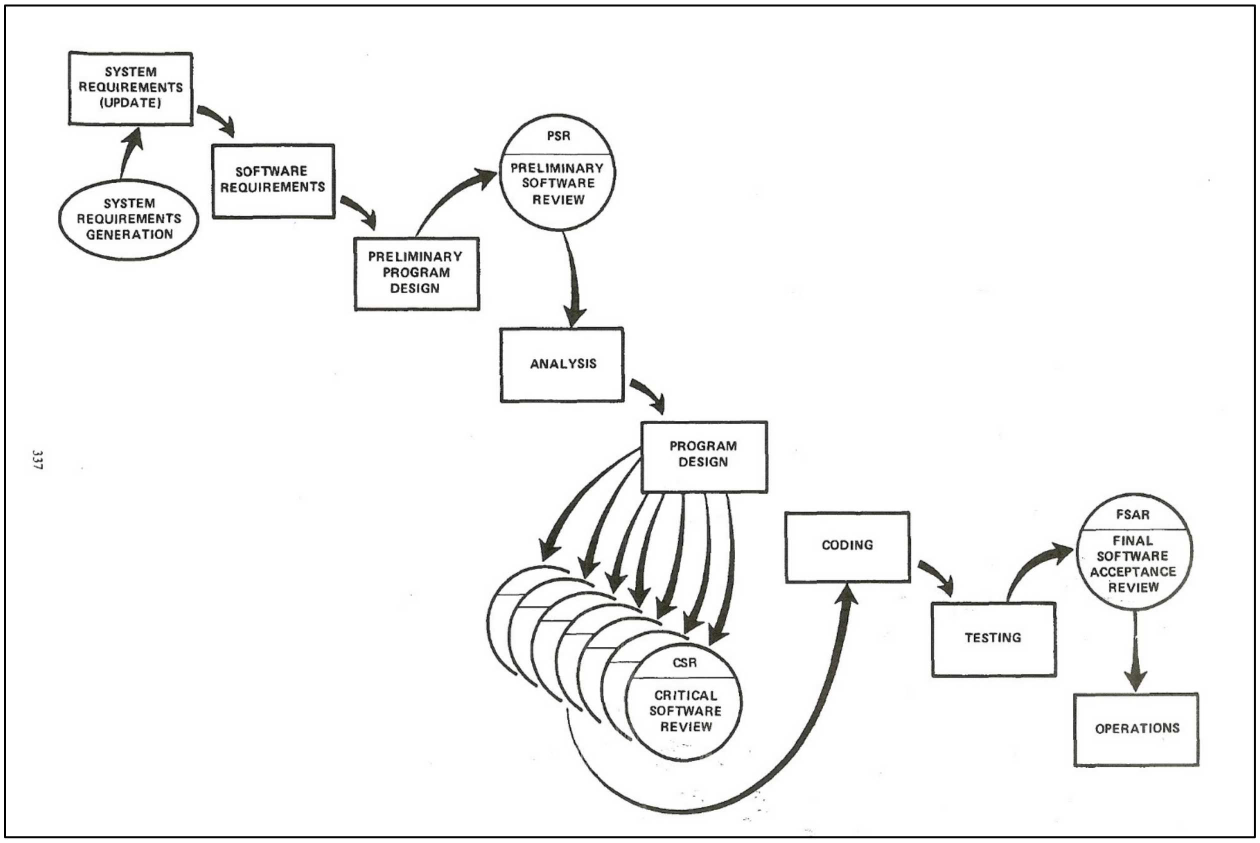
STEP 5: INVOLVE THE CUSTOMER

Royce's final recommendation is to involve the customer.   He states:

> "For some reason what a software design is going to do is subject to interpretation even after previous agreement. It is important to involve the customer in a formal way so the customer has committed himself at earlier points before final delivery. To give the contractor free rein between requirement definition and operation is inviting trouble." (Royce 1970)

As figure 9 depicts, Royce provides multiple points in the software development life cycle where the insights and judgement of the customer can strengthen the development effort.

Today's agile methods typically include a Product Owner as an active member within each agile team who provides the voice of the customer to team.   As a member of the agile team, the Product Owner participates in all agile team activities.



SYSTEM REQUIREMENTS (UPDATE)
SYSTEM REQUIREMENTS GENERATION
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
337
CSR
CRITICAL SOFTWARE REVIEW
CODING
TESTING
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS

Figure 9: Involve the Customer at Multiple Points in the Development Effort (Royce 1970)

CONCLUSIONS

As previously stated, nearly everyone that publishes agile software development practices refer to the concepts identified in Figure 1 of this paper. Figure 10 shows the last figure provided in Royce's 1970 paper. It has many more steps and interactions and is based on implementing the five recommendations that significantly reduce the development risk inherent in a waterfall development process. Today's agile practices incorporate many of these recommendations to some degree. But irrespective of the software development process that a project manager and his development team selects, the following attributes need to be included within the software development methodology:

- Invoke a method of baselining and versioning the project and execute a change process that can receive, assess and execute changes from multiple sources into the baseline in a timely and responsive manner.
- Recognize the interactive relationship of the entire software development process and that previous steps may need to be re-executed based on new information obtained while executing a downstream activity.
- Establish a high-level architecture (program design) for the system, write it down, and ensure that all parties understand this architecture.
- There are elements of the system that need to be documented. Make a conscious decision of what documentation, actually information transfer, will be required and use whatever forms of expressions are necessary to setup and execute this information transfer.

- Be willing to model, simulate, and prototype high risk areas of the system to obtain a thorough understanding and resolution of key technical areas early within the design of the system. Incorporate the lessons learned from these activities into the design and software development process for the system
- Testing should commence early in the software development process and become a continuous activity throughout the software development process.
- Keep your customer involved and committed throughout the entire development process.

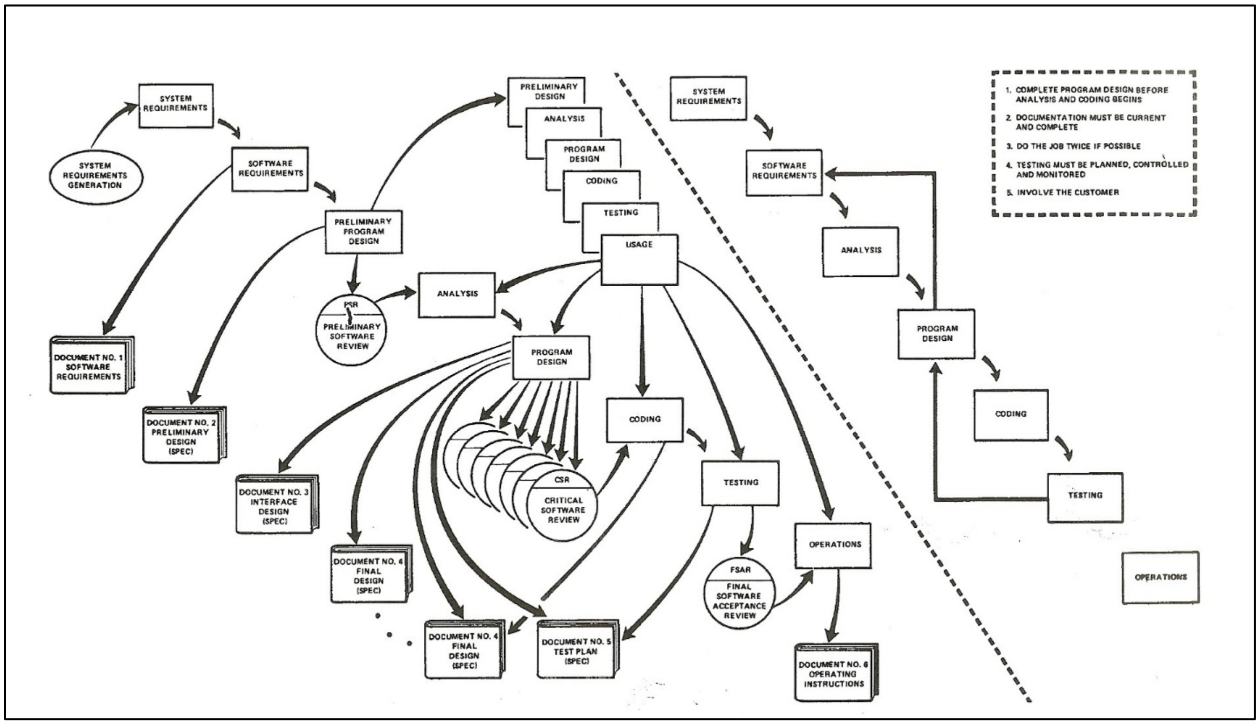

SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
CODING
TESTING
OPERATIONS
CSR
CRITICAL SOFTWARE REVIEW
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 5 TEST PLAN (SPEC)
DOCUMENT NO. 6 OPERATING INSTRUCTIONS
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
1. COMPLETE PROGRAM DESIGN BEFORE ANALYSIS AND CODING BEGINS
2. DOCUMENTATION MUST BE CURRENT AND COMPLETE
3. DO THE JOB TWICE IF POSSIBLE
4. TESTING MUST BE PLANNED, CONTROLLED AND MONITORED
5. INVOLVE THE CUSTOMER

Figure 10: Adding Royce's Recommendations to the Waterfall Development Methodology (Royce 1970)

When written down, these attributes appear to be common-sense, but in 1970 Dr. Royce felt the need to present a paper that described how to invoke a few common-sense recommendations into a step-wise software development approach. Then in 2001, a group of highly experienced software developers felt the need to write down a Manifesto for Agile Software Development that again described a set of common-sense recommendations into a software development approach. Since that time, many agile practice books and guidelines have been published that demonstrate now to implement these common-sense recommendations.

BIBLIOGRAPHY

"Manifesto for Agile Software Development.", http://agilemanifesto.org.

"Winston W. Royce.", accessed 3/17, 2018, https://en.wikipedia.org/wiki/Winston_W._Royce.

Abrahamsson, Pekka, Muhammad Ali Babar, and Philippe Kruchten. 2010. "Agility and Architecture: Can they Coexist?" *IEEE Software* 27 (2): 16-22.

doi:http://dx.doi.org/10.1109/MS.2010.36. http://search.proquest.com.proxygwa.wrlc.org/docview/215836934?accountid=33473.

Booch, Grady. 2011. "The Architect's Journey." *IEEE Software* 28 (3): 10-11. http://search.proquest.com.proxygwa.wrlc.org/docview/862912221?accountid=33473.

Cockburn, Alistair. 2007. *Agile Software Development; the Cooperative Game*. Second Edition ed. Boston, MA: Pearson Education, Inc.

Erdogmus, Hakan. 2009. "Architecture Meets Agility." *IEEE Software* 26 (5): 2-4. doi:http://dx.doi.org/10.1109/MS.2009.121. http://search.proquest.com.proxygwa.wrlc.org/docview/215837170?accountid=33473.

Girvan, Lynda and Debra Paul. 2017. *Agile and Business Analysis - Practical Guidance for IT Professionals*. Swindon, UK: BCS Learning & Development Ltd.

Layton, Mark C. and Steven J. Ostermiller. 2017. *Agile Project Management for Dummies*. Hoboken, NJ: John Wiley and Sons, Inc.

Leffingwell, Dean, Alex Yakyma, Richard Knaster, Drew Jemilo, and Inbar Oren. 2017. *SAFe Reference Guide: Scaled Agile Framework for Lean Software and Systems Engineering*. Vol. 2017. Willard, Ohio: Pearson, Education, Inc.

Royce, Winston W. 1970. "Managing the Development of Large Software Systems." *Proceedings, IEEE WESCON.*

Schuh, Peter. 2006. *Integrating Agile Development in the Real World*. Hingham, Massachusetts: Charles River Media, Inc.

Sessions, Roger. 2008. *Simple Architectures for Complex Enterprises*. Redmond, WA: Microsoft Press.