PROJECT MANAGEMENT
CENTER FOR EXCELLENCE

A.J. CLARK SCHOOL OF ENGINEERING
Civil & Environmental Engineering Department

APPLYING 1970 WATERFALL LESSONS LEARNED WITHIN TODAY'S AGILE DEVELOPMENT PROCESS

Johnny D. Morgan, PhD

2018 Project Management Symposium

Overview of the Presentation

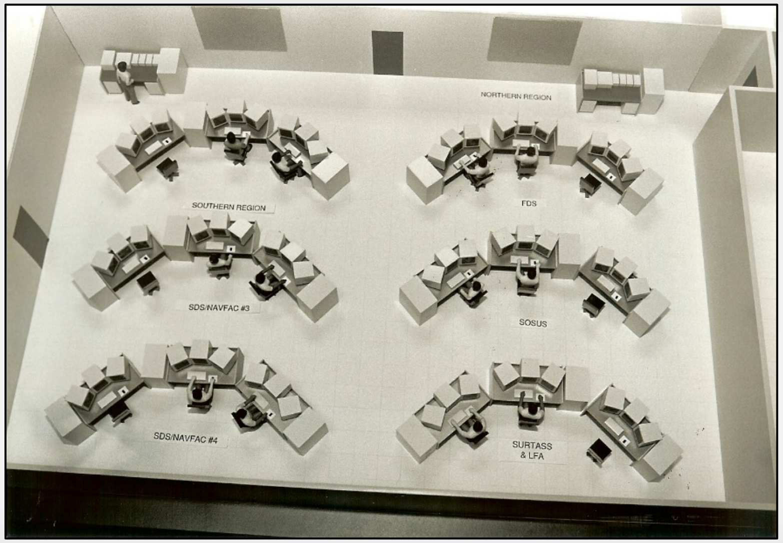
- Purpose: This presentation will review a paper published in 1970 that describes 5 steps to reduce development risks associated with waterfall development activities and describe how these steps are incorporated into today's agile development practices

- Companion Symposium Paper is Available

My Experiences That Influenced Developing This Presentation

- **Geosituational Demonstration System**
  - Intelligence Analyst System
  - IBM PC Based
  - 8 Person Team
  - Approximately 100 KSLOC in size
  - Monthly Code Deliveries for Evaluation
  - Developed from 1988 through 1990

- **Navy Undersea Surveillance System**
  - Large Scale, Wide-Area System
  - Multiple Interconnected Sites
  - 100+ Processing Cabinets
  - 200+ Engineers/Developers
  - 2000+ KSLOC in size
  - 3+ Years of Development to Final Delivery
  - Developed from 1990 through 1994



Managing The Development of Large Software Systems

- **Meet Winston W. Royce**
  - American Computer Scientist
  - Received his PhD in 1959
  - Started as a Project Manager at TRW in 1961
  - Became a Director at Lockheed's Software Technology Center in the 1980s
  - Retired in 1994 and died at his home Clifton, VA in 1995

- **In 1970, he published a paper entitled "Managing the Development of Large Software Systems" (Royce 1970)**
  - Described his personal views about managing large software projects based on assignments during his past 9 years developing software packages for spacecraft mission planning, commanding, and post mission analysis
  - He admits to becoming prejudiced by his experiences and the paper relates some of these prejudices

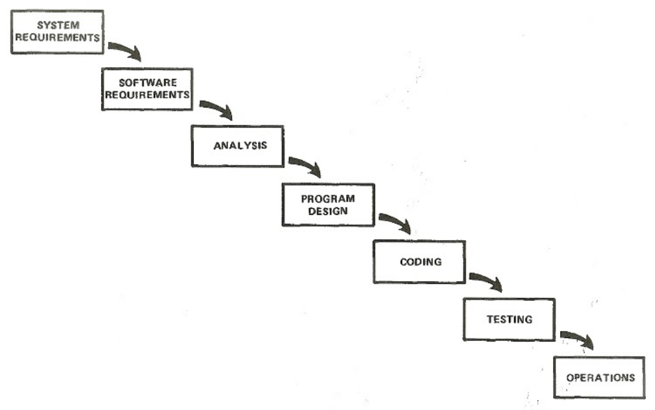


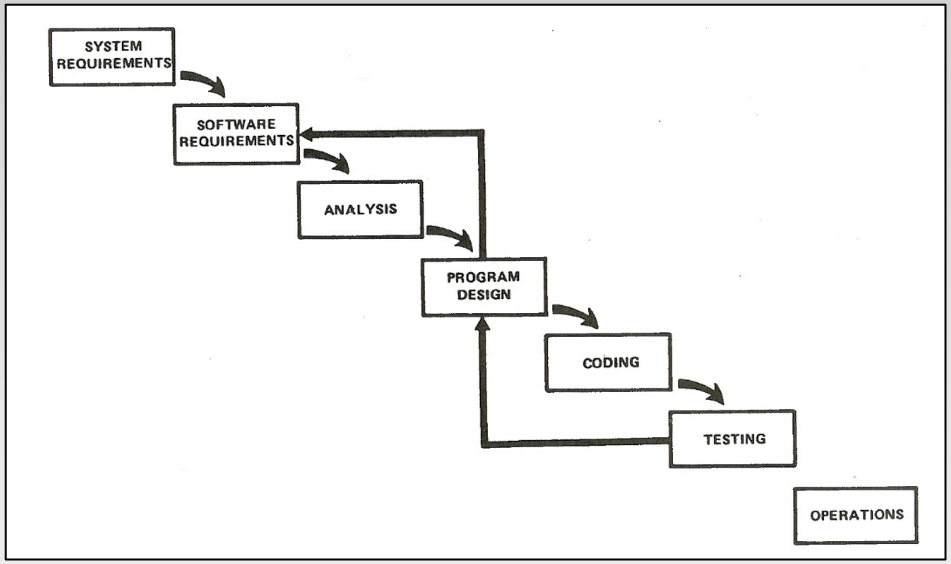
Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Most people remember this diagram from his paper!

Belief in the Concept, But

- Royce states “I believe in the concept but the implementation is risky and invites failure.”

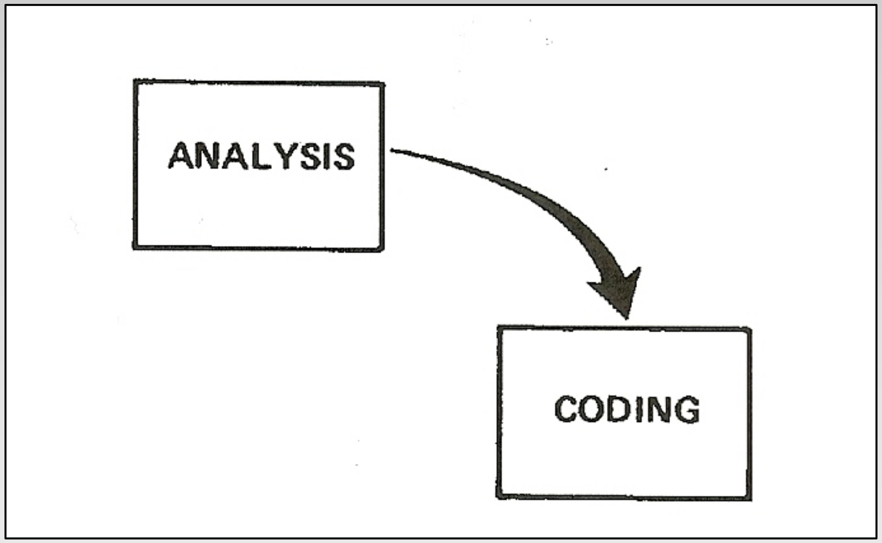
- If “the testing phase….is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed…the required design changes are likely to be so disruptive….one can expect up to a 100 percent overrun in schedule and/or costs”



- “I believe the illustrated approach is fundamentally sound. The remainder of the discussion presents five additional features that must be added to this basic approach to eliminate most of the development risk.”

Royce Was A Lean Thinker

- "There are two essential steps to all computer program developments….there is the analysis step, followed by the coding step."
- "Plan(s) to manufacture larger software systems, and keyed only to these steps, however, is doomed to failure"



- "Many additional development steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development costs…..Customer personnel typically would rather not pay for them and development personnel would rather not implement them."

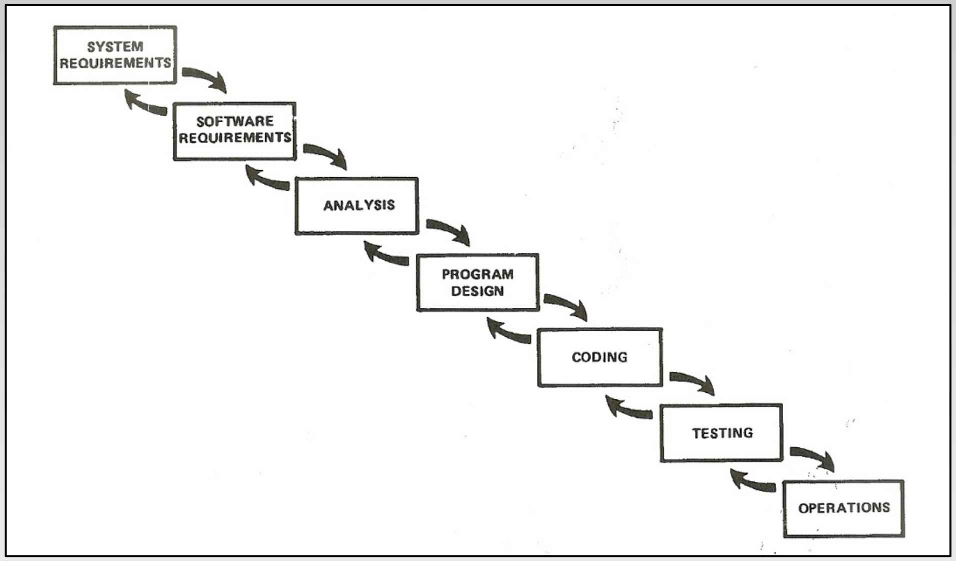
"The prime function of management is to sell these concepts to both groups and then enforce compliance on the part of development personnel."

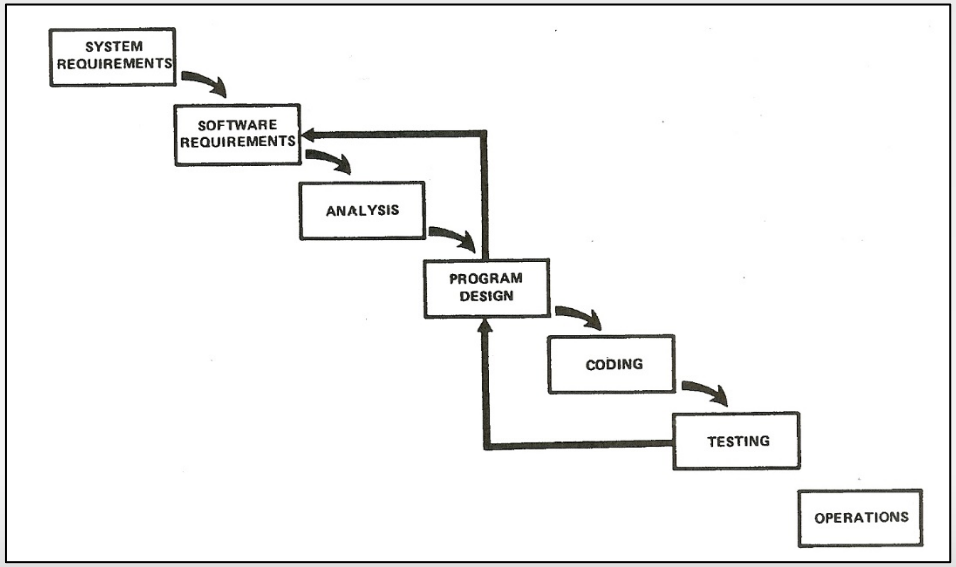
Iterative Intersection Between the Various Phases

- **Royce discussed the interactive relationship between successive development phases.**

- **He champions a change process and a moving baseline to which to return in the event of unforeseen design difficulties**
  - **Changing requirements**
  - **Results from modeling and prototyping**
  - **Inability to translate concepts into working software**

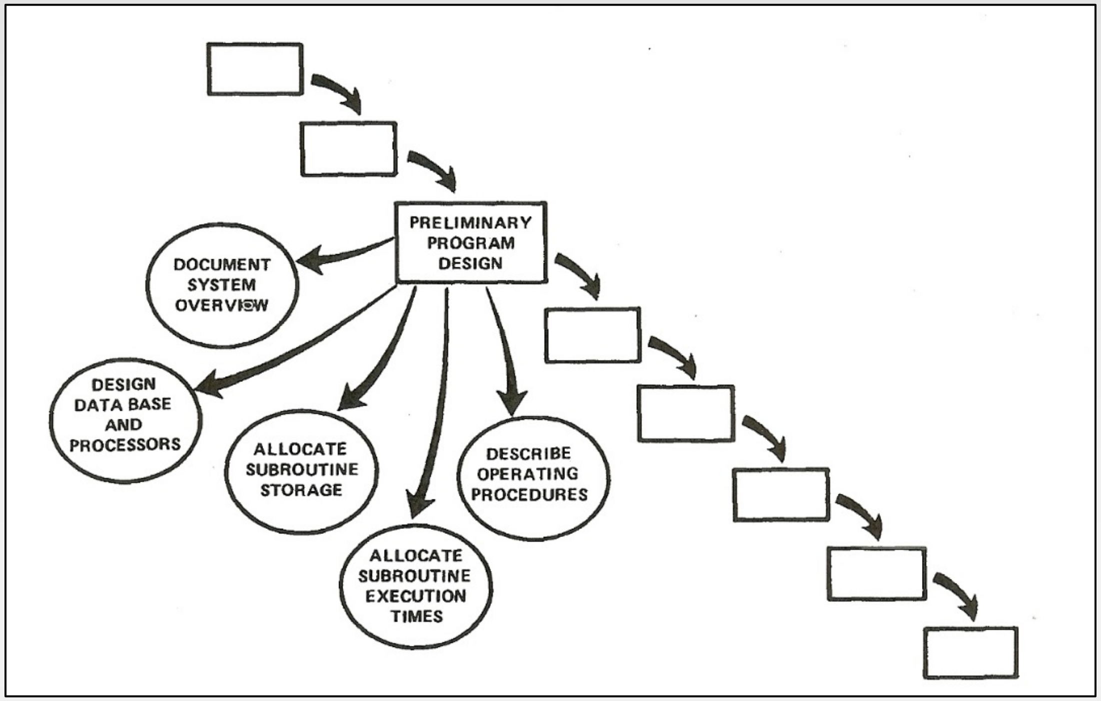
I have found that an effective change process is key to any organization and supporting development process being agile or responsive.





Step 1: Program Design Comes First

- Begin the design process with program designers, not analysts or programmers
- Design, define, and allocate the data processing modes
- Write an overview document that is understandable, informative, and current


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

Royce refers to this as a Preliminary Design and it will impose constraints on the program designer to ensure that his software fits into enterprise!

Agility versus Architecture

Agile Perspective:

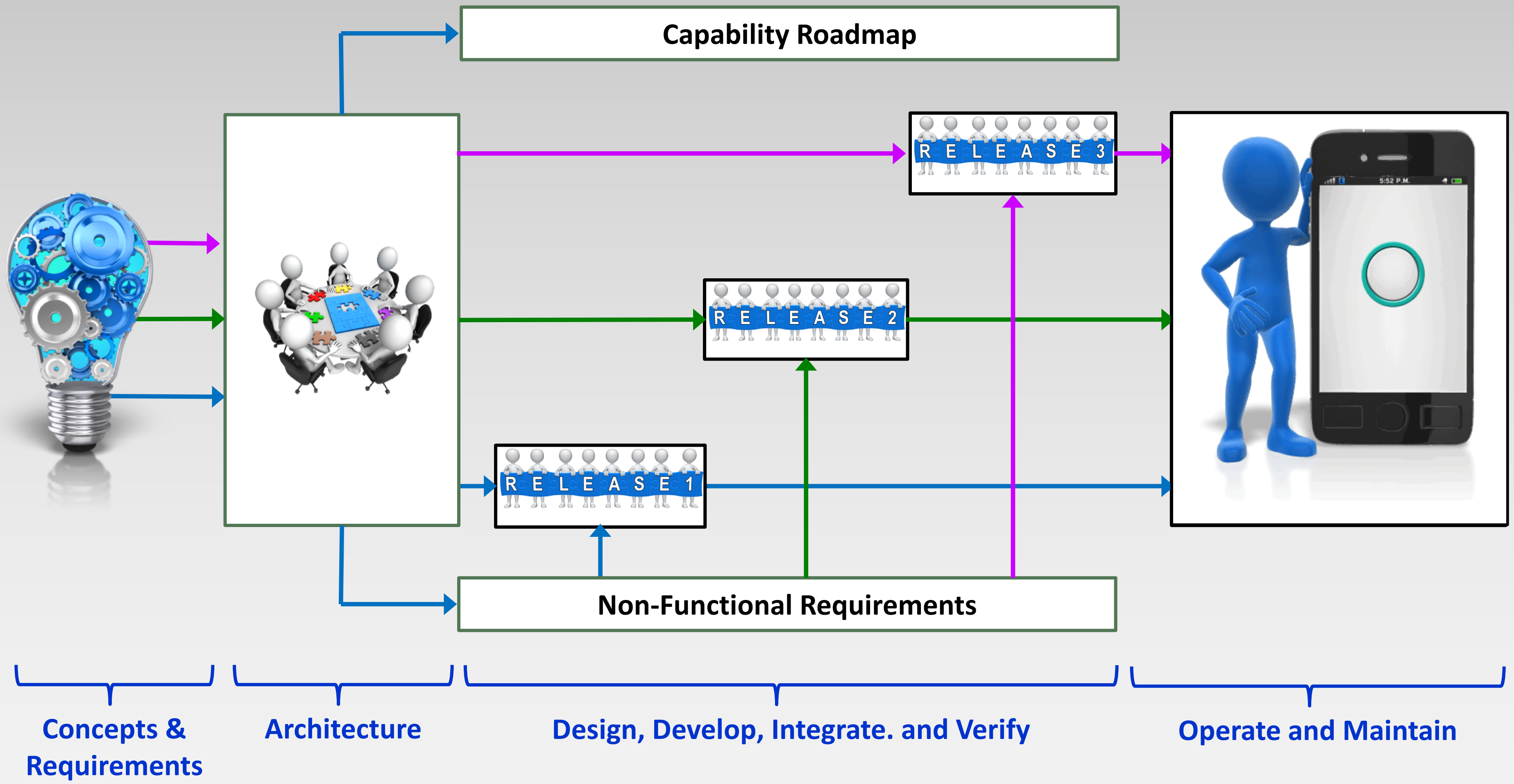
- Architectures emerge from self-organizing teams
- Be resolutely adaptive: respond only when changes occur
- Perceive architecture as Big Up-Front Design (BUFD) with You Ain't Gonna Need it (YAGNI) features
- Focused on functional requirements

Architecture Perspective:

- Architectures naturally seek the maximum level of complexity (Law of Entropy)
- Anticipate change, plan for them, and isolate their impact
- Small successive refactoring through each sprint is insufficient
- Focused on both functional, and non-functional requirements (Reliability, Availability, Maintainability, Performance, Information Security, etc.)

- **The Scaled Agile Framework (SAFe) introduces the concept of Architectural Runway**
  - ➢ **It provides the necessary technical basis for developing business initiatives and implementing new features and capabilities**
  - ➢ **It 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 et al., 2017)**

Defining Architecture

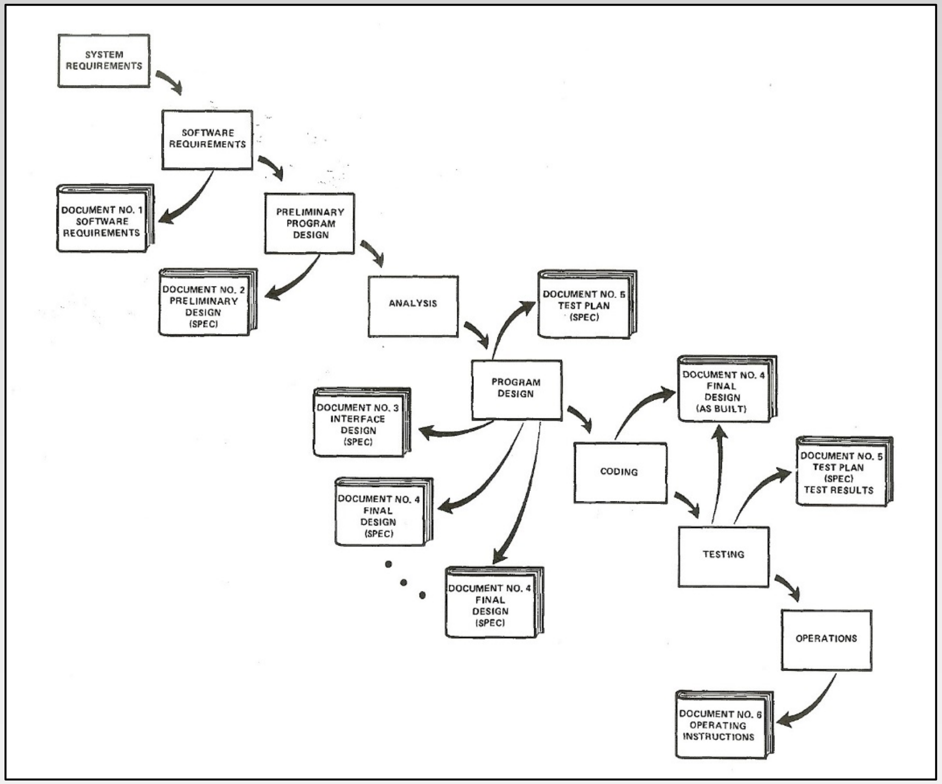


Architecture: Encompasses the set of significant decisions about the structure and behavior of the system. These decisions prove to be the hardest to undo, change and refactor (Abrahamsson et al., 2010)

Try to Interleave architectural stories and functional stories

Step 2: Document the Design

- "The real value of good documentation begins downstream…during the testing phase and continues through operations and redesign"
- Without documentation:
  - ➢ It is difficult to find and correct mistakes
  - ➢ The people who wrote the code are generally the people who then have to maintain the code
  - ➢ After initial operations, to add new features to the software may require reverse engineering of the existing code



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)
DOCUMENT NO. 4 FINAL DESIGN (AS BUILT)
CODING
DOCUMENT NO. 5 TEST PLAN (SPEC) TEST RESULTS
TESTING
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS

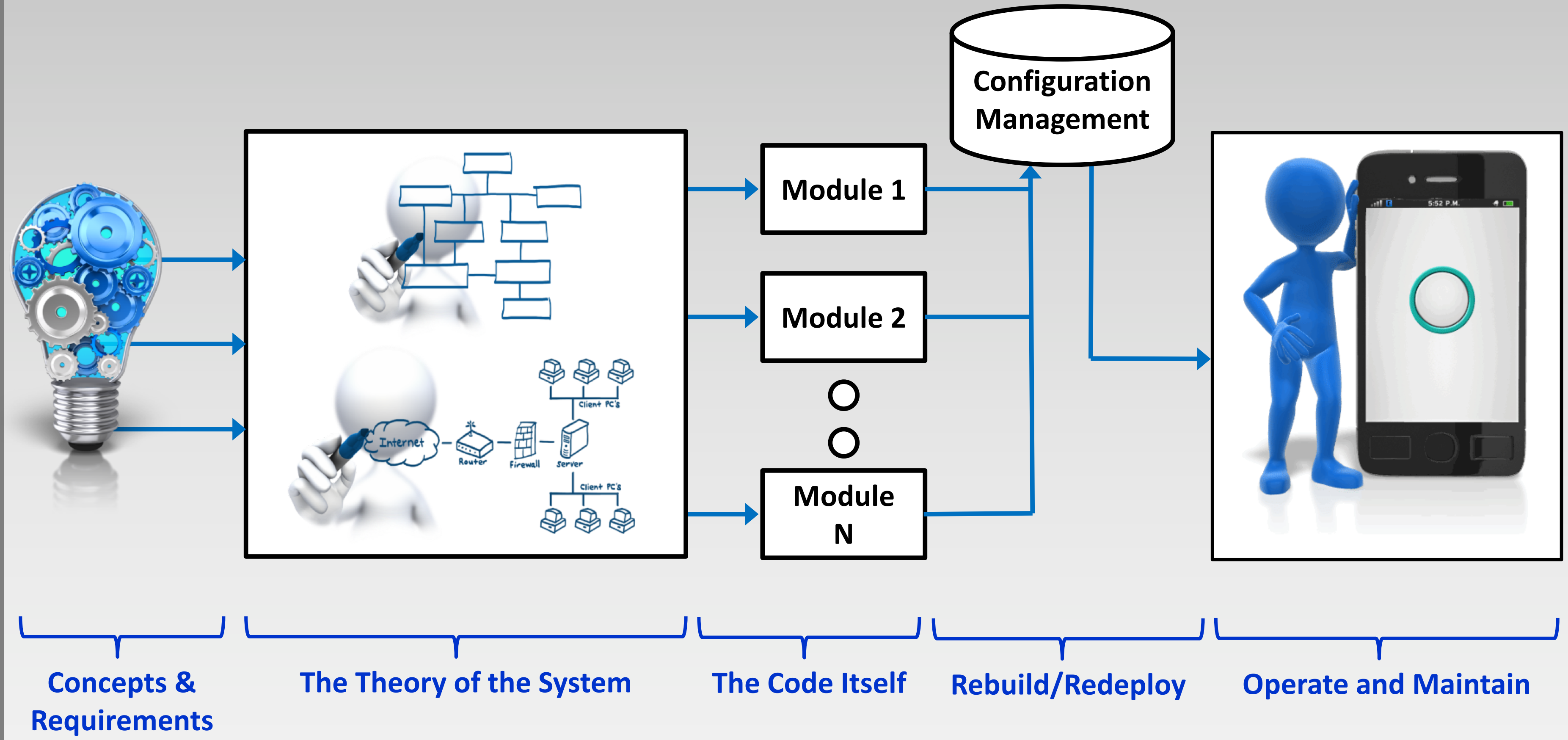
"A verbal record is too intangible to provide an adequate basis for an interface or management decision….If the documentation does not exist there is yet no design, only people thinking and talking about the design which is of some value, but not much"

Thoughts on Documentation

- **The code is the truth, but not the whole truth.**
- **Large architectural decisions can not be discerned from the code itself**
  - The results of these decisions are scattered throughout the code
  - Their meaning and presence are in the heads of the code's creators and not easily evident by staring at the code (Booch, 2011)
- **In agile software development, the code changes often, and thus keeping the lowest level documentation in-line with the code is difficult (almost futile)**
- **"The architect's and designer's job is not to pass along 'the design' but to pass along 'the theories' driving the design….**
  - This latter goal is more useful and more appropriate….
  - Knowledge of the theory is tacit in the owning, so passing along the theory requires passing both explicit and tacit knowledge" (Cockburn, 2007)

Document those items that helps the next programmer build an adequate theory of the system/program

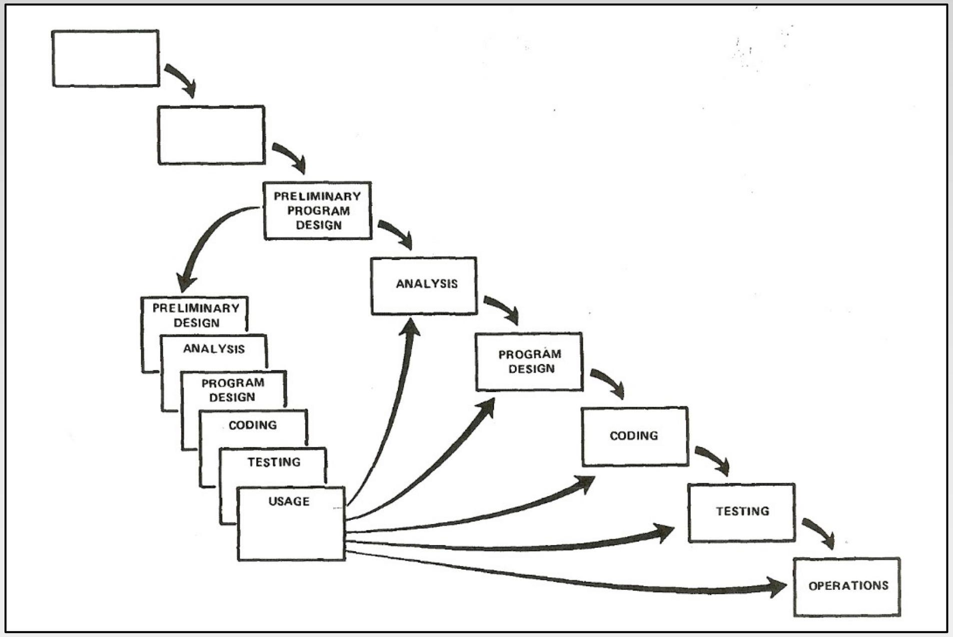
Recommendations on ~~Documentation~~ Artifacts



- **Change the mindset from "document" to "artifact"**
  - A collection of knowledge, irrespective of the media that contains it
- **Collect and maintain the following:**
  - Knowledge that defines to users, operators, and maintainers how to operate and maintain the system
  - Knowledge that describes how to rebuild and redeploy the system should a problem or disaster occur.
  - Knowledge that allows future personnel to modify the system over it's life cycle (The theory of the system)

Step 3: Do It Twice

- "If the computer program in question is being developed for the first time, arrange matters so that the version delivered to the customer…..is actually the second version insofar as critical design/operations areas are concerned."
- "They must quickly sense the trouble spots in the design, model them, model their alternatives…"
- "At least perform experimental tests of some key hypotheses"

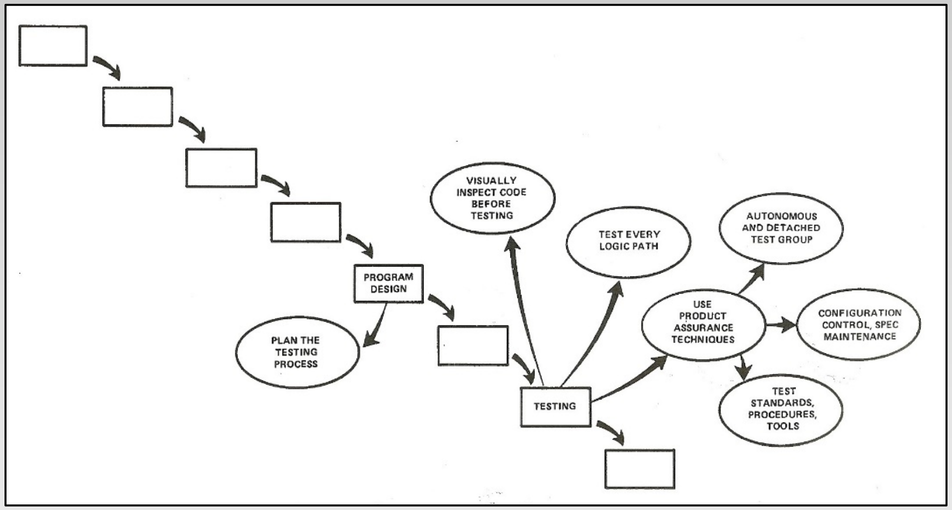


Many People Now Say "Fail Fast!"

Build a Minimum Viable Product (MPV)

Step 4: Plan, Control and Monitor Testing

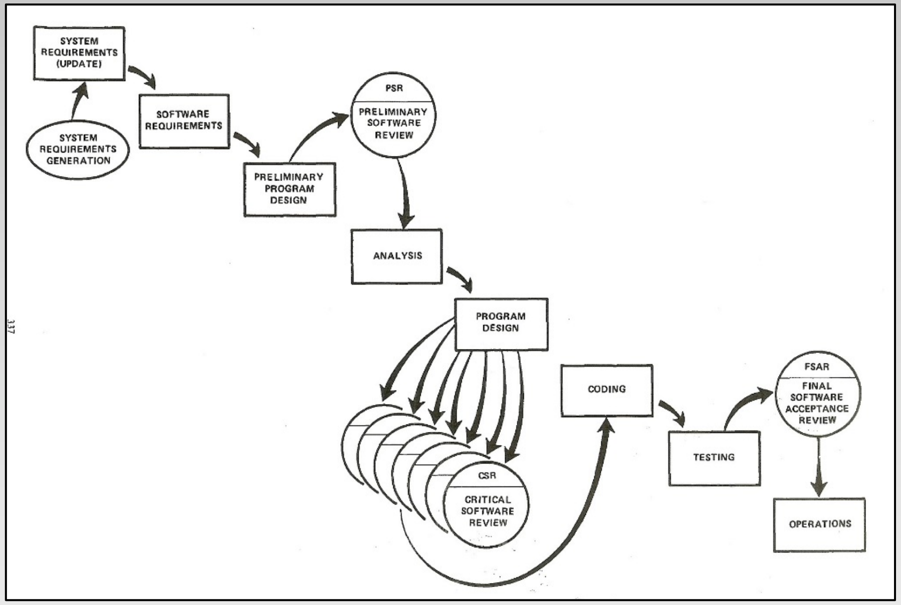
- "Most errors are of an obvious nature that can be easily spotted by visual inspected…Every bit of code should be subjected to a visual scan by a second party…"
- "Test every logic path in the computer program at least once…this step will uncover the majority of coding errors.
- "After the simple errors are removed, then it is time to turn over the software to the test area for checkout purposes"


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

Modern Software Practices Now Include:
- Test Driven Development (TDD)
- Pair Programming
- Code Reviews
- Separate Development, Test, and Production Environments
- System Test Teams
- Code Coverage Tools
- Automated Testing Tools

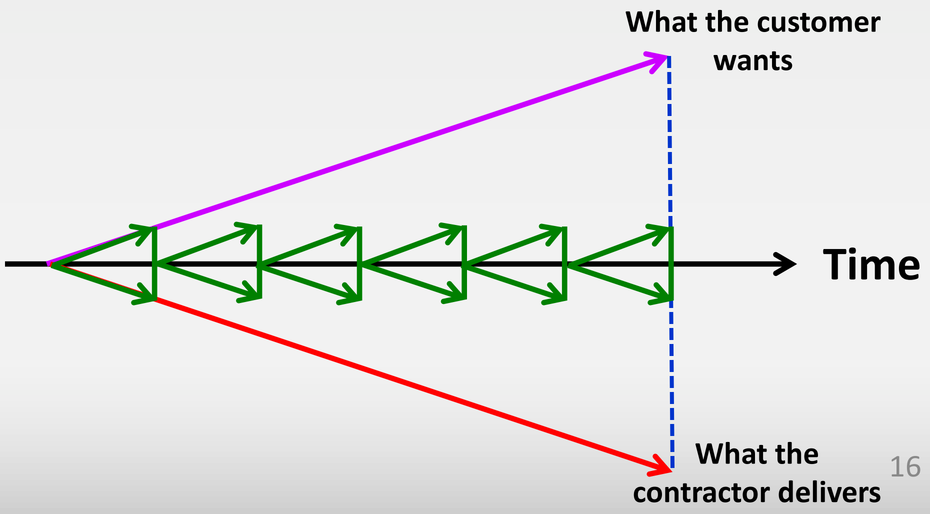
Step 5: Involve the Customer

- "For some reason what a software design is going to do is subject to wide interpretation even after previous agreement.
- It is important to involve the customer in a formal way so that he is has committed himself at earlier points before final delivery.
- To give the contractor free rein between requirements definition and operation is inviting trouble.
- (Add points where)…Insight, judgement and commitment of the customer can bolster the development effort"



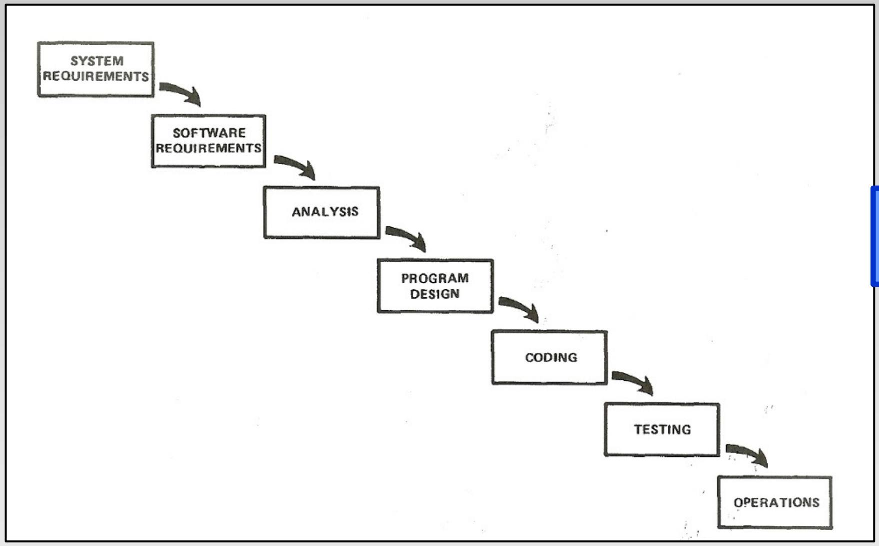
SYSTEM REQUIREMENTS (UPDATE)
SYSTEM REQUIREMENTS GENERATION
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
CSR
CRITICAL SOFTWARE REVIEW
CODING
TESTING
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS

Agile Practices now include a Product Owner as a Member of the Agile Team



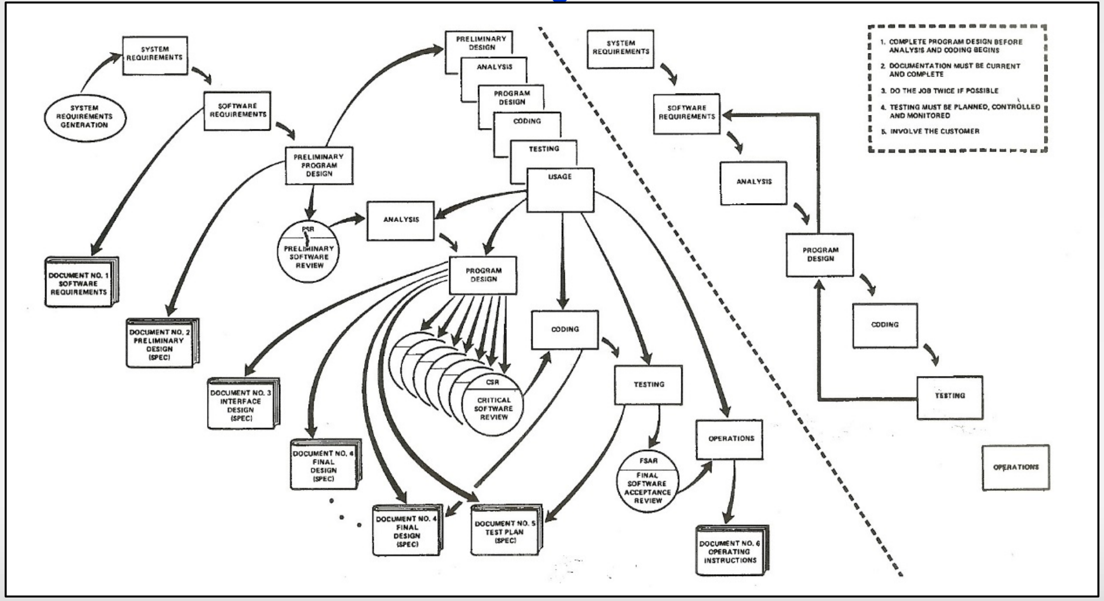
Very Different Perspectives

Everyone Knows This (Page 2)



Five Steps to Reduce Risk:

1. Program Design Comes First
2. Document the Design
3. Do It Twice
4. Plan, Control and Monitor Testing
5. Involve the Customer


SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
PSR PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
CSR CRITICAL SOFTWARE REVIEW
FSAR FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS
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

Now you Know This (Page 11)

Contact Information

Johnny D. Morgan, PhD
INCOSE ESEP, AWS CSA & CD
PMI PMP, SAFe Agilist
General Dynamics Information Technology
johnny.morgan@gdit.com

References

Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and architecture: Can they coexist? *IEEE Software, 27*(2), 16-22.

Booch, G. (2011). The architect's journey. *IEEE Software, 28*(3), 10-11.

Cockburn, A. (2007). *Agile software development; the cooperative game* (Second Edition). Boston, MA: Pearson Education, 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*.