

ASQ Section 1313 Talk - Feb 2007 - Software Lifecycle Models

Feature Topic: Software Lifecycle Models

Feature Speaker: Arnold Miller, Openwave Systems Inc. Broomfield, CO

E-mail Speaker: arnold.miller@openwave.com

Bio: Arnold Miller is a Principle Software Quality Engineer with Openwave Systems in Broomfield Colorado leading test efforts on mobile phone location service software. He has over 20 years in the Software field including Software Developer, Software Maintainer and Quality Engineer. Mr. Miller is an ASQ senior member of the Boulder ASQ Section. He has taught the Section's CSQE exam review course. He has presented papers and tutorials on Software Quality and ISO 9000 at ASQ Section Meetings and Rocky Mountain Quality Conferences.

Abstract:

All Software product lifecycle and process models have three generic phases: definition, development and maintenance. The generic definition phase focuses on What the product will do via customer contact, project planning and requirements analysis. The generic development phase focuses on How to design, code and testing the software to demonstrate the What. The generic maintenance phase focuses on Changes to the software because of errors, adaptations and enhancements. (Quality Council of Indiana: CSQE Study Guide, 2002)
This presentation will discuss and compare how the Waterfall, Prototyping, Spiral, Incremental and Agile models implement these generic phases.

Talks:

Slide 01: Topic talk side

Slide 02:

There are many different Software Product Lifecycle Models like Waterfall (slide 8 to 11), Prototyping (slide 12 to 13), Incremental (slide 14 to 15), Spiral (slide 17 to 21), and Agile (slide 22 to 25)

Each model implements the generic definition, development, and maintenance phases (slide 3 to 5) and the supporting activities (slide 6 to 7 with different emphasis and occurrences).

Slide 03: Generic definition phase focuses on "What will the product do" by answer questions:

- What Information is to be processed?
- What Functions and performances are desired?
- What System behavior can be expected?
- What Interfaces are to be established?
- What Design constraints exist?
- What Validation criteria are required?

Via Steps

- Customer contact - each element defined and focused on role will have in implementation
- Project planning - Risks analysis, Resource allocation, Cost estimates, Work tasks, and define schedules
- Requirements analysis - Technical detail on information, behavior, functions

Slide 04: Generic development phase focuses on "How to produce product" via

- Designing Software architecture
- Design Data structures
- Implement Procedural details
- Translate Design into executable code
- Performing tests validation and verification activities

Via Steps

- Design - Translates requirements into representations describing the architecture, data structures, algorithms and human computer Interfaces
- Coding - Translate Design into programming language for machine executions
- Testing - validations demonstrate functional, logic and implementation works or has errors

Slide 05: Generic maintenance phase focuses on "Changes to existing product" because of

- Error correction and prevention
- Adaptation as environment evolves
- Enhancements based on changing requirements

Changes groups

- Correction - fixes problems
- Adaptation - changes to external operation environment
- Enhancement - additional features that improves system
- Re-engineering - Improvements to internal workings for better understanding, maintenance, performance

Slide 06: Supporting Lifecycle activities across all phases:

- Quality Assurance - Reviews to ensure maintaining quality in each phase.
- Configuration Management - Uniquely identify and control information created and used in each phase.
- Project Monitoring - Processes to ensure schedule and cost under control in each phase
- Measurement - Processes and products measures that determine quality and productivity
- Documentation - Internal and External documents to support the life cycle and customers
- Reusability - Leveraging other internal and external products, systems, components
- Risk Management - Understanding, Analysis, Mitigating problems before they occur

Slide 07: Problem Solving Loop

- Status Quo - Current State of affairs
- Problem Definition - Identifies and describes what is to be solved
- Technical Development - Solves the problem through apply technology
- Solution Integration - Delivers results (documentation, programs, data, products) to those that requested the solution

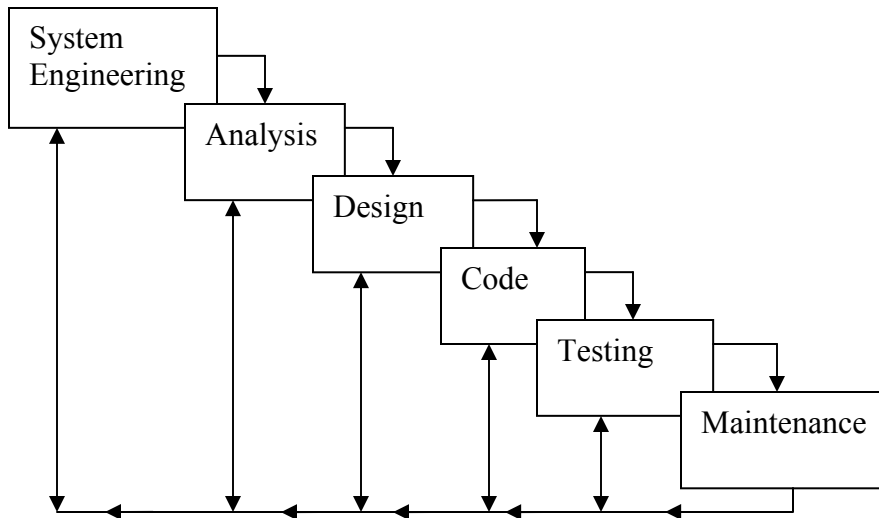
Applicable at many levels:

- Macro level - entire application
- Middle level - single and multiple components
- Bottom level - sub, single and multiple functions

Slide 08: Waterfall Model

Linear progression through the following activities, with feedback to previous activities

- System Engineering - (definition) Complete system requirements and specific software requirements
- Analysis - (definition) Understand information domain, required functions, behavior, performance and interfacing characteristics
- Design - (development) Architecture, Data Structures, Procedure details
- Code - (development) translate design to code human readable form. Verified and under configuration control before testing
- Testing - (development) focus on logical internals and functional external interactions to uncover errors and show that defined inputs generates required results
- Maintenance - (maintenance) Changes after release because of errors found, improvements requested and environmental changes. Reapplies preceding life cycles to existing products



Slide 10: Waterfall Pluses:

- Reasonable approach when requirements are well understood
- Gating checks to get to next activity - User buy-in (Quality and Monitoring)
- Templates for methods of Analysis, Design, Code, Testing
- Provides concrete, testable requirements and specifications
- Follows standard manufacturing product flow

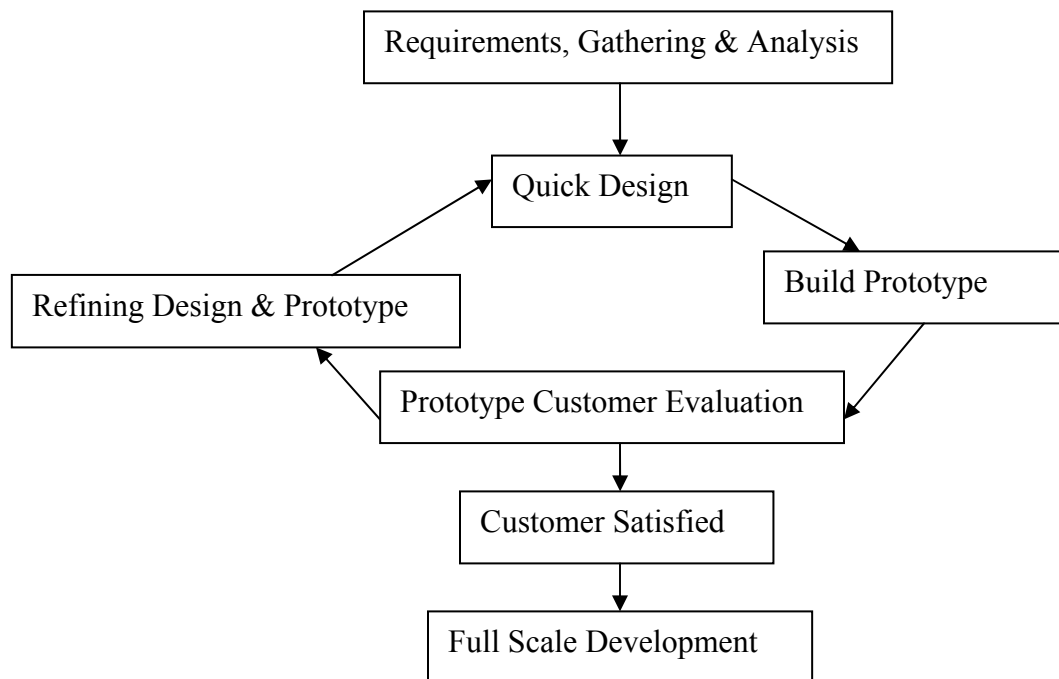
Minuses

- Real Projects rarely follow sequence flow
- Indirectly accommodate iterations
- Difficult for customer to state all requirements explicitly
- Working version late in process
- Potential blocking states to get to next activity
- Requirement changes can have significant negative impacts

Slide 11: Prototyping Model

Help Solve the working version late in process problem:

- Create model of software that is built in an evolutionary manner.
- Can be build on the target environment or on a convenient platform.
- Customer/user evaluation to refine requirements and future prototypes
- Better understanding for requirements, designs and validations



Pluses:

- Early incomplete working versions
- Repeat incomplete working versions
- Identifying requirements
- Customer buy-in via direct participation

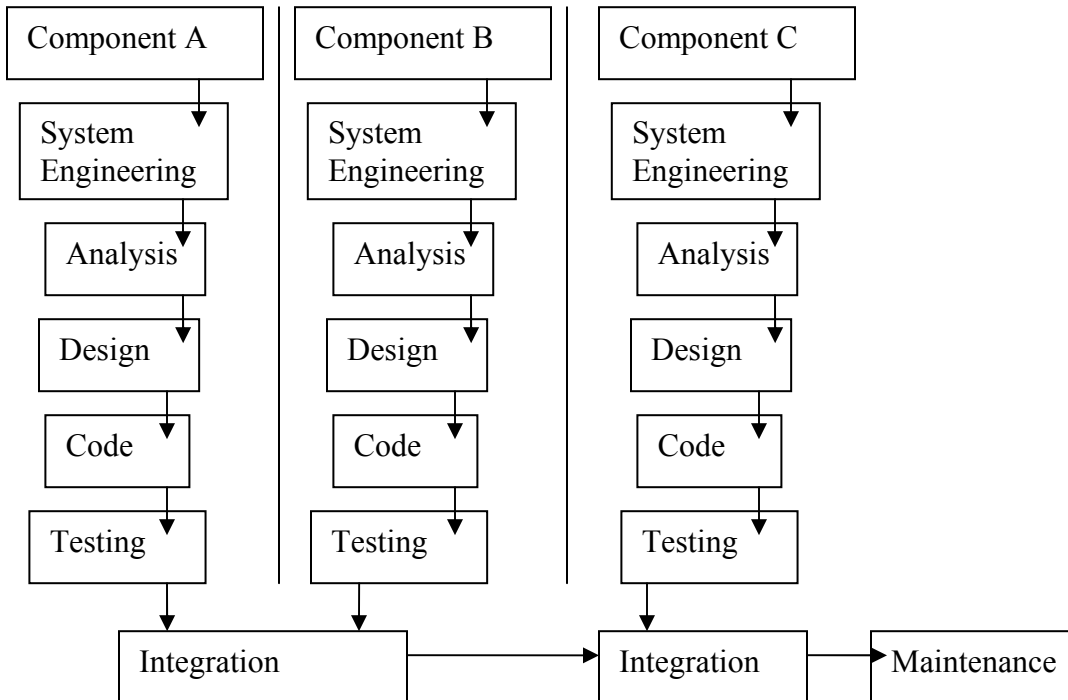
Minuses

- False impression of a complete working system
- Too much time doing prototypes (analysis paralysis)
- Limited focus and Not on correct platform
- Continued changes can reduce quality
- Still need other Model to implement complete solution

Slide 13: Incremental Model

Divides large products/projects into smaller manageable components

- Each component is done in isolation using predetermined method up to Component testing
- Integrate Components together and into final Integration testing
- Combined Component Maintenances



Pluses:

- Specialists focus on their expertise area
- Supports parallel and over lapping pipe-line Component development
- Sub-divides Project into workable components

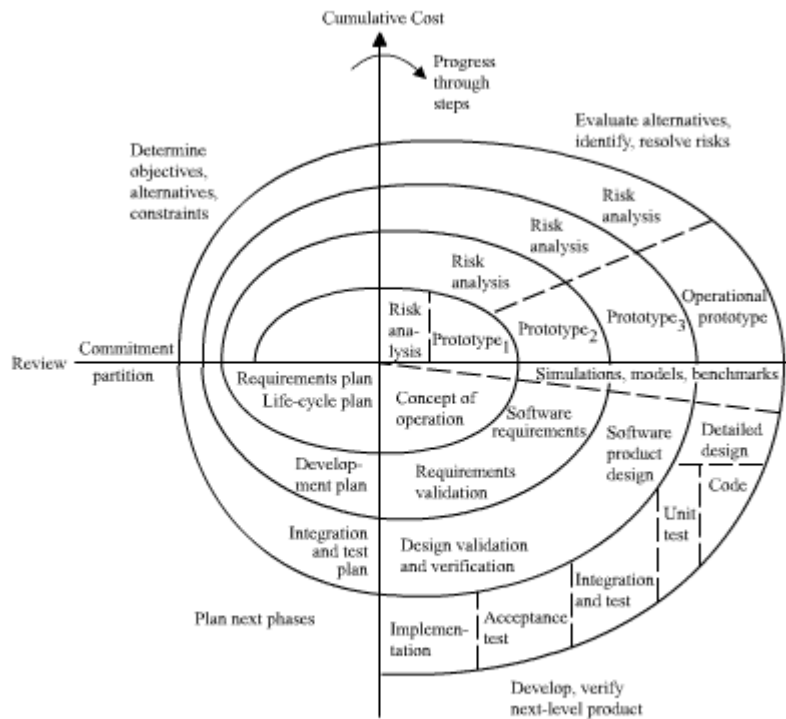
Minuses

- Multiple projects to manage and coordinate together
- Good coordination for integration
- Increase Requirement management changes
- Potential duplication and missing efforts

Slide 15: Spiral Model

Repetitive cyclical activities that build on the previous cycle (iteration) to generate more complete versions

- Planning (definition)
 - Determine objectives, alternatives and constraints
 - Initially and customer feedback from previous iteration
- Risk Analysis (definition)
 - Alternative analysis, understanding, resolutions, mitigations
 - Initially and customer feedback from previous iteration
 - Continue or Stop decision
- Engineering (development)
 - Design next working version based on previous version
 - Leveraging towards final solution
- Customer Evaluation (development)
 - Customer review on current working version
 - Comments to improve and perfect working version



The Spiral Model

Slide 17: Spiral Pluses:

- Early risk analysis, resolution and mitigation
- Periodic customer feedback, buy-in, improvement
- Partial working system at each iteration complete
- Combines Waterfall and Prototyping Pluses with limited iterations

Minuses

- Project scope and schedule can changes with each iteration
- Improper risk understand

Slide 18: Plan-Driven Environment

Goals: Predicable, Stable, High Assurance

Size: Larger Teams and Projects

Environment: Stable, Low Change, Project/Organization Focus

Customer: As-need interactions, contract focus

Plan & Control: Documented, quantitative control

Communication: Explicit documented knowledge

Requirements: Formalized project, capability, interface quality, foreseeable evolution

Development: Extensive design, longer increments, expensive refactoring

Test: Documented plans and procedures

Personnel: 50% level 3 early, 10% throughout; 30% level 1B workable (Cockburn)

Culture: Policies and Procedures (thrive on order)

Early: Definition and Design

Level 3: Senior people

Level 2: Middle people

Level 1a: Junior people

Level 1b: Entry people

Level -1: Negative people

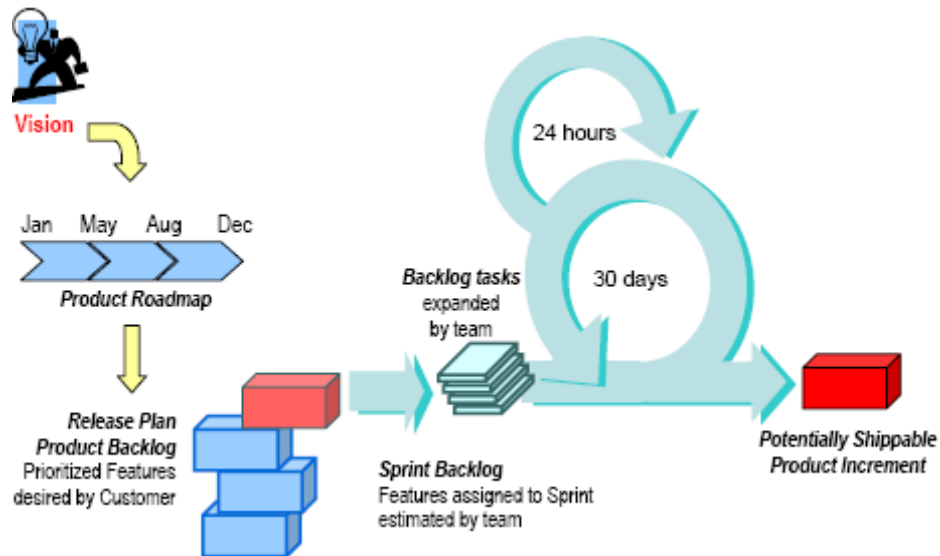
Slide 19: Agile Model

Formal Incremental and Iterative model via "Manifesto"

- **Individuals and interactions** valued over process and tools
- **Working software** valued over comprehensive documentation
- **Customer collaboration** valued over contract negotiation
- **Responding to change** valued over following the plan

Many versions: Scrum, Lean, and Extreme Programming

Slide 20: Extended Scrum Framework



Slide 21: Common Practices

- Planning Related
 - Quick scope each iteration using customer business priorities
 - Requirements expressed in user-developed test validation
 - Standard work week, overtime suggests failure
 - Fix Cost and Schedule (Value/Vision) drives Features
 - Release planning initial features into which iterations
 - Revise plan for next iteration based on completed and upcoming work

Slide 22:

- Iterative Life Cycle Related
 - Standard Iterative cycle every few weeks (e.g. 2 weeks, 30 days)
 - Update project scope (requirements), costs and schedules
 - Prioritized Backlog items into current iteration - time box estimates
 - Daily (Scrum) stand-up reports (did, will did, issues)
 - Burn down Completes chart feature items and task sub-items
 - Ending Retrospective - Went well, need improvements, could do difference (process feed back loop)
 - Concentrated effort during iteration

Slide 23:

- Design Approach Related
 - Maintain simple design, remove complexity, redesign for improvements
 - Real customers available for full time requirement questions
- Coding Practices Related
 - Define and follow coding standards
 - Code is the key communication medium
- Testing Related
 - Write unit test before code
 - Daily code builds, integration, and validation
 - Customer write acceptance tests

Slide 24:

Pluses:

- Project facing new technology
- Handles Risk that cannot be identified up-front
- Partial working system at each iteration complete
- Quickness to change directions and plans
- Value/Vision-driven schedules
- Feature and task completion metrics and measurements
- Cross Functional Team buy-in (Dev, Test, Doc, Project, Product, etc)
- High Customer interaction and acceptance

Minuses

- Signification Specification Documentation
- Few Project plan changes
- Contracts with fixed prices, date and scope
- Plan-driven schedules

Slide 25: Value-Driven Environment

Goals: Rapid value, responding to change

Size: Smaller Teams and Projects

Environment: Turbulent, high change, project focus

Customer: Dedicated on-site, prioritized increments focus

Plan & Control: Internalized plans, qualitative control

Communication: Tactical interpersonal knowledge

Requirements: Prioritized informal stories/test cases, undergo unforeseeable changes

Development: Simple design, short increments, inexpensive refactoring

Test: Executable test cases define requirements; testing early, often, always

Personnel: 30%+ level 2 and 3 experts; 0% Level 1B and Level -1s (Cockburn levels)

Culture: Many degrees of freedom (thrive on chaos)

Level 3: Senior people; Level 2: Middle people; Level 1a: Junior people;

Level 1b: Entry people; Level -1: Negative people

Slide 26: References:

Boehm, Barry and Richard Turner, "Using Risk to Balance Agile and Plan-Driven Methods", IEEE Software magazine, IEEE Computer Society, June 2003

Cohn, Mike and Doris Ford, "Introducing an Agile Process to an Organization", IEEE Software magazine, IEEE Computer Society, June 2003

Certified Scrum Master Training, Rally Software Development, CSM Training v12.73, 2006

CSQE Primer: V. Software Engineering Processes - A. Environmental Conditions - 1. Life Cycles, Quality Council of Indiana, 2002

Slide 27: References

Duncan, Scott P, "Agility and Discipline: Coexistence or Combat?", Software Quality magazine, ASQ Software Division, Number 2 Spring 2004

Lycett, Mark, et al, "Migrating Agile Methods to Standardized Development Practices", IEEE Software magazine, IEEE Computer Society, June 2003

Pressman, Roger S., Software Engineering - A Practitioner's Approach - Part One: The Product and Process - Chapter 2: The Process, McGraw Hill, 2001, Fifth Ed.

The Spiral Model, Software Engineer Lessons, University of Vermont
<http://courses.cs.vt.edu/csonline/SE/Lessons/Spiral/> 1999, (last visited 19 Feb 2007)

Williams, Laurie and Alistar Cockburn, "Agile Software Development: It's about Feedback and Change", IEEE Software magazine, IEEE Computer Society, June 2003