Software Engineering Notes

Patrick Lindsay

January 24, 2013

Contents

Part I

Notes

Chapter 1

January 15, 2012

1.1 Overview

Software Engineering - The application of sound engineering practices to software creation and maintenance.

1.1.1 Software development Life Cycle (Traditional Approach)

- Requirements Phase
- Analysis or Specification Phase
- Design Phase
- Implementation/Integration Phase
- Maintenance Phase
- Retirement

Requirements Phase

- Determining the NEEDS and WANTS of the client or customer.
- Determining the constraints of the system.

Analysis or Specification Phase

- After analyzing the requirements, construct a *specification document* which explicitly describes what the product is to do, and the constraints under which it must operate.
- This includes the description of the input, output, actions, and UI.

- The specification document can be used as part of a contract with the client.

Problems with the Spec Document

1. Ambiguity - one sentence may have more than one interpretation.
2. Incompleteness - relevant fact or requirement is left out.
3. Contradiction - two places in the spec document are in conflict.

Design Phase

- Construct an *Architectural Design*.
- Construct a *Detailed Design*.
- Test for *traceability*.

1. Architectural Design - Description of the product in terms of modules.
2. Detailed Design - Description of each module.
3. Traceability - each part of the design can be traced to a statement in the specification document.

Implementation Phase

- Code each module from the detailed design.
- Programmer tests his/her own code separately.
- Modules are combined and tested by developers.
- Product is tested by SQA group. This is called product testing.
- Project is given to the client for acceptance testing.

Maintenance Phase

- Corrective Maintenance - bug squashing
- Enhancement Maintenance - Updates
  - Perfective - client makes new demands
  - Adaptive - changes in the environment of the product requires changes in the software.
- Perform regression testing - insuring that changes have not affected already working functionality.

Retirement Phase

- Determining if desired changes are too costly.
- Determining if a product is obsolete.

1.1.2 Four Components of the Software Engineering Enterprise

The four P's

1. Process
2. Project
3. People
4. Product

Process

- The process is sometimes called the life-cycle model or development sequence.
  - Waterfall
  - Spiral
  - Incremental Build
- Makes use of several process frameworks.
  - Personal Software Process (PSP)
  - Team Software Process (TSP)
  - Capability Maturity Model (CMM)
- Documentation Standards
  - IEEE
  - ANSI

Project

- The set of activities needed to produce the required product.
- Project management is extremely important.
- Many projects are not about developing new products, but maintaining already existing *legacy* systems.

People

- Team Organization
- Team Management
- Relationship with customer or client
- Relationship with end users
- Communication with upper management

Product

Includes

- Requirement Specification Document
- Design Document
- Source Code
- Executable
- User Manuals

Chapter 2

January 17, 2013

2.1 Traditional Software Engineering Process

2.1.1 Historical Influences

- Structured programming (Edsger Dijkstra's letter calling "GOTOs" harmful) uses sequence control, iteration, invoking functions
- Object Oriented paradigm: the use of objects with data and functionality which can represent real-world entities.

Note:
Silver Bullet ca. 1980s
Likened the software crisis to a werewolf. Object Oriented Paradigm was the Silver Bullet.
It did not work as expected.

- Design Patterns: stock of reusable design elements (templates)

2.1.2 Component Reuse

A component as defined by Meyer is "a program element satisfying:"

1. The element may be used by other program elements. (Clients)
2. The clients and their authors do not need to be known to the element's author

2.1.3 Key Expectations of Software Engineering

1. Decide in advance what the specific quality measures are to be for the project and product.
   Predetermine quantitative quality goals.

2. Gather data on all projects to form a basis for estimating future projects.

3. All requirements, designs, code and test materials should be freely and easily available to all members of the team.
   Source code should always be available to all team members in an easily accessible and interpreted way.
   Git, Mercurial, etc.

4. A process should be followed by all team members. *Uniformity*

   (a) Design only against requirements.

   (b) Program only against design.

   (c) Test only against requirements and design.
   ALWAYS FOLLOW THE RECIPE!

5. Measure and achieve quality goals.

2.2 Methods

Be able to draw and discuss these.

2.2.1 Waterfall Method

SEE DIAGRAM 52.9 ON PG 53.

- First described by William*(?)* Royce in 1970.
- No phase is complete until documentation for that phase has been completed and approved by the SQA group.
  Very orderly; heavy on documentation.
- Has been used with great success on a variety of products.
- Feedback loops permits modifications to be made to the previous phase.

Advantages

1. Enforced disciplined approach
2. Requirement that documentation be provided at each phase.
3. All products of the phase must be checked by SQA.
4. Inherent aspect of each phase is testing.

Disadvantages

- The resulting specification document may not be able to be understood by the client.
- It can lead to the construction of product that does not meet the client's needs.

2.2.2 Rapid Prototype Model

See diagram on pg 55. Construction of a functional subset of the desired product in order to allow the client and the developer to interact.
Keyword is rapid. This is a thrown-together, proof-of-concept type project; a mock-up.

Advantages

- The process is linear and possibly faster than the Waterfall Model
- Increases interaction between client and developer.

Disadvantages

- Client may inaccurately think the product is almost complete when viewing the prototype.
- Developer may attempt to use the prototype as part of the final product.

2.2.3 Waterfall-Rapid Prototype Hybrid

May form a hybrid model using the rapid prototype as the first phase in the Waterfall Model in order to increase interaction but allow for feedback loops within the development of the product.

2.2.4 Incremental Model

Software is implemented, integrated, and tested as a series of incremental builds.
Code pieces providing specific functions.

Advantages

1. Results in builds which can be developed in weeks, not months or years.
2. End user need not learn the entire product at one time.
3. Client need not pay for the entire product at one time.
4. Developer gets paid earlier.
   (At each build delivery)

5. Open-ended design makes maintenance easier.

6. Easier to make changes during development.

Disadvantages

1. Each new build must fit in without destroying existing builds.
   Regression Testing

2. Requires more careful to design to make it open to additions.

3. Can degenerate to a build and fix product if broken into too few builds.

2.2.5 Spiral Model

See Figure 2.12 on pg 63 and Figure 2.13 on pg 65.

- A Waterfall Model with each phase preceded by risk analysis in an attempt to control or resolve risk.
- Each phase is 360.
- The measure of the radius is the cumulative cost to date.
- The measure of the angle is the progress measure.
  Each phase is 360.
  Requires a very experienced engineer.

Advantages

1. The emphasis on alternatives and constraints supports the reuse of existing software.

2. The incorporation of software quality as a specific objective.

3. Answers the question of how much testing should be performed in terms of risks.

4. Maintenance is simply another cycle of the spiral, the same as development.

Disadvantages

1. Intended exclusively for internal development.
   Client and developer are members of the same organization.

2. *Applicable only to large-scale projects.*

3. Must have developers who are skilled at pinpointing the possible risks.

2.3 Agile Methods

2.3.1 General

According to the Agile Manifesto, they value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responsiveness to change over following a plan.
  YOU DON'T ALWAYS HAVE TO FOLLOW THE RECIPE!

Traits

- Highly iterative
- Pair programming with a focus on teamwork and ego-less programming.
  THIS IS MANDATORY.
- Early and planned testing.
- Story cards *Similar to storyboards in movies.*
- Refactoring *Turning working code into better code.*
- Feedback

Principles Behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive disadvantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity - the art of maximizing the amount of work not done - is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Chapter 3

January 22, 2013

3.1 Refactoring

- Reduce software complexity
- Improve internal structure while preserving the behavior of the code.
  Prettifying
- Improve software quality
  Readability, execution efficiency, size efficiency, etc.
- Performed in response to "bad smells" in the code, undesirable characteristics
  Martin Fowler
- Automated or manual

3.2 Teams

Team - a group of professionals organized in order to complete the task of creating a large software project.

3.2.1 Team Structure

Project Factors Related to Structure of the Team

1. Difficulty of the problem
2. Size of the program in LOC or Function Points
   LOC stands for Lines of Code
3. Time the team will stay together

4. Degree of modularity for program
5. Required quality and reliability
6. Rigidity of delivery date
7. Degree of communication required

Jelled Team

- A group of people which are so tightly knit that the attitude is that the whole is greater than the sum of the parts
- Egos are forgotten and the team becomes important
- Exhibits cohesiveness, team spirit, common definition of success
- Generally more productive, more motivated, and happier

Why Don't All Teams Jell?

- A frenzied work atmosphere
- High frustration causing friction among team members
- A "fragmented or poorly coordinated" software process
- An unclear definition on the team
- "Continuous and repeated exposure to failure" - M. Jackman *Homeopathic Remedies for Team Toxicity*

Necessary Team Traits

Personality

- Openness - intellectual curiosity
- Conscientiousness - self-discipline, pushing toward goals
- Extroversion - energy, emotions, seek company of others
  Being a people person
- Agreeableness - compassionate and cooperative
- Neuroticism - how a person responds to stress, emotional stability

How to Make Personality Traits Work

1. Recognize people have different types of personalities
2. Assemble a diverse team covering a range of personalities
3. Create an open, honest, tolerant atmosphere in team meetings

3.2.2 Roles

- Team Leader - Responsible for overseeing all aspects of the team project; holds the tie-breaking vote.
- Technical Lead - Expert on all technical aspects of the project, in particular the hardware and software used for development.
- Designer - Designs the project and breaks the project into smaller pieces (*Modules*) for the programmers.
- Lead Programmer - Needs to have an understanding of the project as a whole; organizes all of the other programmers.
- Technical Writer - Writes all documentation for the project (*Team meeting minutes, Specification Document, Users' Manual*)
- Configuration Management - Maintains the code base for the project; could include CVS responsibilities.
- Quality Assurance - Writes, maintains, and conducts all testing associated with the project.

3.2.3 Organizational Structures

- Democratic Team
- Hierarchical or Chief Programmer
- Team Manager/Team Leader
- Synchronize-and-Stabilize Team
- Agile Team

Democratic Team

- Group of up to 10 programmers
- Equal partnership with egoless programming
- Works well if the group is small, highly competent
- Problem with who is in charge
- Positive attitude about finding fault
- Good in research environment with difficult problem

Hierarchical or Chief Programmer

- One overall manager (*Chief Programmer*)
- Everyone understands the lines of authority
  One boss.
- Team members tend to participate less in decisions; decisions are handed down from above
- May have a Programming Secretary and a Backup Programmer for the Chief Programmer
- Difficult to find one person adept at both managing and programming.

Team Manager/Team Leader

See Figure 4.4, pg. 114

- Split the responsibilities of Chief Programmer into Team Manager and Team Leader
- The Team Manager handles the nontechnical management
- The Team Leader deals with the technical issues of the project
- Results in programmers having two bosses
- May be difficult to determine if an issue is technical or nontechnical

Technical Organizational Structure for Large Projects

- One Project Leader oversees several team leaders
- Each team leader has several programmers for which he/she is responsible
- Clear lines of communication
  Two level structure

Synchronize-and-Stabilize Team

Has been used by Microsoft

- Small team led by a manager and having three to eight developers and three to eight testers working one-to-one with the developers
- Developers are given freedom to design and implement their portions as they wish
- Each day, the partial components are tested and debugged.
- Encourages creativity and innovation yet the daily synchronization keeps the project on track.

Agile Team

- Work in pairs (MANDATORY)
- Provide instant review
- Create test cases which are used for daily testing
- Remove the problem if one developer leaves, the knowledge does not disappear about a portion of the project

Chapter 4

January 24, 2013

4.1 Tools for Software Engineers

- Analytic (Theoretical) Tools:
  1. Stepwise Refinement
  2. Cost-Benefit Analysis
  3. Divide-and-Conquer
  4. Separation of Concerns
  5. Software Metrics
- Software Tools (CASE: Computer-Aided Software Engineering)

4.1.1 Analytic Tools

Stepwise Refinement

- Process whereby a project is successively decomposed into more detailed instructions.
- In each step, a given task is written as a set of subtasks.
- Term was first coined by Niklaus Wirth in 1971.
- Helps to concentrate on relevant aspects of the current development phase and ignore details that need not be considered.
- A postponement of decisions on details until as late as possible.
- Critical to object-oriented paradigm.

Cost-Benefit Analysis

- Comparing estimated future benefits against future costs for a certain decision.
- Problems occur in that intangible benefits may be hard to quantify.
- May use past experience to project the estimates for benefits or costs.

Divide-and-Conquer

- Most agree this is the oldest analytical tool used in Software Engineering.
- Break a large problem into smaller subprograms that should be easier to solve.
- Idea used in the Unified Process.
- Good concept but no details in the how... GET THIS BULLET
- *Key difference from Stepwise Refinement is that Divide-and-Conquer does not necessarily procrastinate details.*

Separation of Concerns

- First introduced by Dijkstra in 1974.
- Process of breaking a software project into components which overlap as little as possible in relationship to functionality.
- Regression faults are minimized.
  If every component does one function (High Cohesion), making changes won't affect a lot of things.
- Components are more reusable.

Software Metrics

Measurements used to indicate:

- Size (LOC - Lines of Code)
- Duration (Months, years)
- Effort (Person-Months)
- Quality (Fault Density - Faults/1000LOC)
- Efficiency (Faults/Unit of time)
- Reliability (Mean time between failures)

Product/Process

4.1.2 CASE Tools

- UpperCASE or front-end tools - used in the requirements, analysis and design workflows
- LowerCASE or back-end tools - used in the implementation and maintenance activities

Types of CASE Tools

- Data Dictionary - computerized list of all data defined within the product (type and location defined)
  Could be Upper or LowerCASE
- Consistency Checker - tool which checks that everything in the design is in the specification document and everything in the specification document is in the design.
- Report Generator - tool which generates code needed for producing a report.
- Screen Generator - tool which generates the code necessary for a data capture screen.
- Structured Editor - a text editor which is designed to understand the structure of a program in a programming language, aiding in syntax fault prevention or early detection.
- Pretty Printer or Formatter - code often included with the structured editor which makes use of the language syntax structure to display the code in a standard manner (indenting, highlighting reserved words or comments)
  Try Sublime Text 2!
- On-line Interface Checker - editor know every subprogram declared within the product and their parameter lists.
- Operating System Front End - tool which allows the programmer to give commands to the operating system from within the editor.
- Source Level Debugger *LowerCASE*
- Interactive Source Level Debugger *LowerCASE*
- Version Control Tool - keeps detailed record of each version of the project.
  Check out Git with GitHub or BitBucket!
- Configuration Control Tool - manages multiple variations. *LowerCASE*

Grouped CASE Tools

- CASE Workbench - collection of CASE tools that together support one or two activities.
- CASE Environment - collection of CASE tools which support the entire software process.

4.2 Software Versions

- During maintenance, at least two versions of the product will exist: the old version and the new version.
- Revision - what we call the new version.
- Configuration - the specific version of each artifact from which a given version of the complete product is built.

Chapter 5

January 29, 2013

5.1 Testing

Testing - continual process carried on during the entire life cycle of software.

V and V

- Verification - determining wheter a phase has been correctly carried out (at the end of the phase)
- Validation - testing just before a product is delivered to the client to determine if it satisfies the specifications

Use of the Term Testing We use the term testing instead of V and V because they imply that testing can wait until certain points in the process like the end of a phase, when testing must occur throughout the process.