Lesson Proper for Week 4

DEFINITION OF SOFTWARE REQUIREMENTS

Identifying and defining software requirements begin with reviewing the functional or performance requirements developed to identify the constraints on software The system requirement that is allocated to software evaluations determines accuracy, completeness, and applicability of the requirements for work products.

System requirements allocated to software are refined into greater detail to define derived software requirements. Program and project plans that include the capability to acquire reusable software; software requirements are always identified. The tools (i.e., dynamic object-oriented requirements system [DOORS], matrix worksheets, etc.) can be used for the

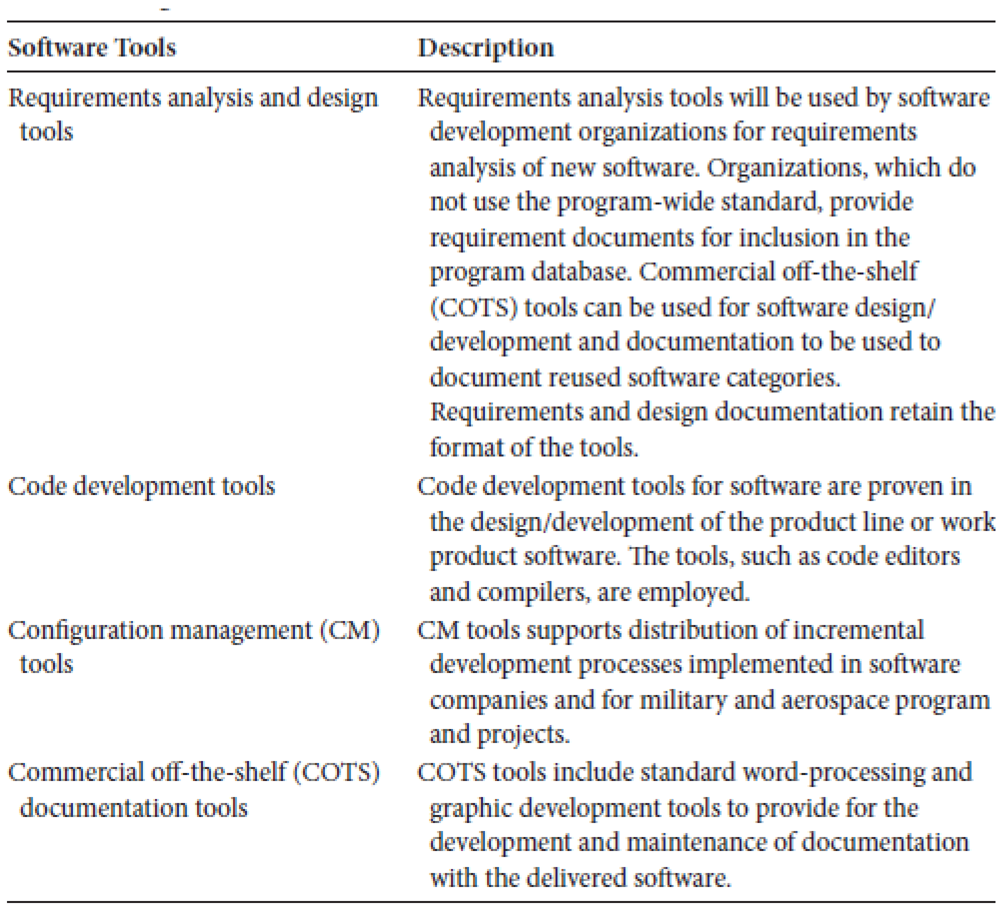
SOFTWARE DESIGN DECISIONS

The establishment of the software architecture definition provides design concepts and decisions for a work product. The software requirements definition and the software operational concepts identify the capabilities and characteristics required for the inputs that are analyzed and integrated to make key design decisions. Many software design tools, as shown in Table 1, benefit the software designer for requirements, code development, configuration management (CM), and software documentation.

Software Requirements Evaluation

The reviews and evaluations of software requirements define that software operational scenarios ensure problems affecting software design are identified, evaluated, and resolved. The software design/development team performs a risk analysis using prototype software to help support early requirements evaluations and design feasibility. Information from these evaluations is fed back into the output of the software requirements development phase if the requirement is proven to be unusable and not to be implemented for use.

TABLE 1: Software Design Tools



Software Tools	Description
Requirements analysis and design tools	Requirements analysis tools will be used by software development organizations for requirements analysis of new software. Organizations, which do not use the program-wide standard, provide requirement documents for inclusion in the program database. Commercial off-the-shelf (COTS) tools can be used for software design/development and documentation to be used to document reused software categories. Requirements and design documentation retain the format of the tools.
Code development tools	Code development tools for software are proven in the design/development of the product line or work product software. The tools, such as code editors and compilers, are employed.
Configuration management (CM) tools	CM tools supports distribution of incremental development processes implemented in software companies and for military and aerospace program and projects.
Commercial off-the-shelf (COTS) documentation tools	COTS tools include standard word-processing and graphic development tools to provide for the development and maintenance of documentation with the delivered software.

Software Reuse

The reuse of software components identifies evaluations by software architecture definitions on how to decide on the incorporation of components into the software design. Opportunities for software reuse support numerous software product developments in state and international markets. The reuse criteria are identified in defined software plans to determine if the program and project reuse library or existing software work products can be used for near-term software design activities.

PEER REVIEWS

Software processes require design engineers to conduct and perform peer reviews to find and correct as many errors as possible before test team integration or customers find problems during delivery. The peer review starts with requirements, design models, and uninterrupted code and unit tests for the software designer. These reviews are applied at various stages during the software design/development life cycle to create clean software work products and provide the assurance that issues or errors are discovered and resolved.

If errors are found early in the development life cycle, time is saved, and the cost is not a concern. Minor software errors that are not fixed or resolved become major errors later in a program and projects. Software design engineers always make mistakes in their code development, so early peer reviews reduce the amount of rework and are not required late in the program and project.

As stated in CMMI® for Development (version 1.3) (CMMI stands for Capability Maturity Model Integration), peer reviews are an important part of verification and a proven mechanism for effective defect removal. An important corollary is to develop a better understanding of the work products and the processes that produced them so defects can be prevented and process improvement opportunities identified. Peer reviews involve a methodical examination of work products by the producers' peers to identify defects and other changes that are needed.

Examples of peer review methods include the following:

- Inspections
- Structured walk-throughs
- Deliberate refactoring
- Pair programming

The peer review verification methods identify software bugs, errors, and defects for removal with recommendations to improve code development as shown in Figure 1.

The peer review method is applied to software work products developed by programs and projects, but it can also be applied to other work products, such as documentation and training, which are typically developed by other software teams. Preparation for peer reviews includes identifying affected teams or groups to participate in the peer review of affected software work products. The criteria for conducting peer reviews are as follows:

- Schedule the peer review at a convenient time
- Assign reviewers (i.e., teams)
- Prepare or update materials
- Provide peer review checklists
- Introduce training materials
- Select software work products
- Provide entry and exit criteria (i.e., minutes, action items, etc.)

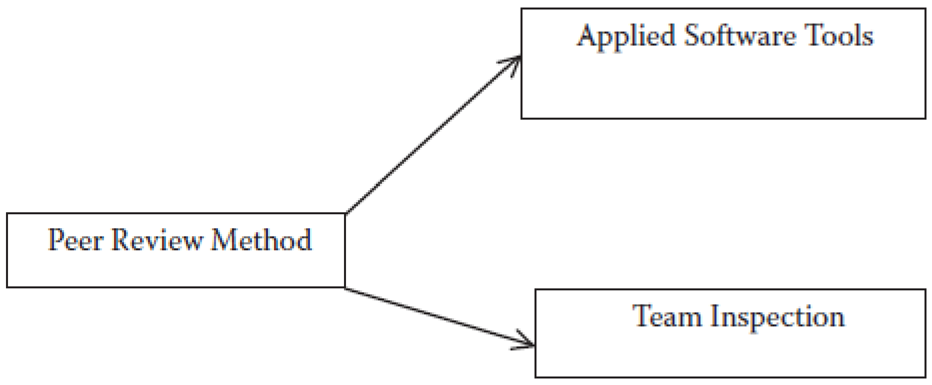


FIGURE 1: Peer review method.

To ensure you have a successful peer review, make sure you have selected the right reviewers to be involved and guidelines are understood from the start. If peer reviews are conducted and performed correctly, the peer review was performed and done right.

SOFTWARE DESIGN/DEVELOPMENT SUGGESTIONS

I suggest we look at two software design/development methods. One method is concurrent software design/development, which is a technique to reduce the time to improve productivity through the simultaneous performance of activities and processing of information. The concurrent software design/development method refers to tasks that are performed simultaneously by different teams or groups that support a team approach to development. The second method is Lean software design/development. According to this method, it is far more effective to have small working teams across the boundaries of informational handoffs, reduce paperwork loads, and maximize strong communication.

Concurrent Software/Design Development

Concurrent software design/development activities require software designers who have enough expertise to anticipate where the defined design is going. When starting software design/development, only partial requirements are known and developed in short iterations to provide feedback for systems to emerge. The use of the concurrent software design/development method does make it possible to delay commitment until the last moment when failure to make a decision eliminates an important alternative or decision.

Lean Software Design/Development

The Lean software design/development objective is to move as many changes as possible from the top curve to the bottom curve as shown in Figure 2.

Lean software design/development delays the freezing of all design decisions as long as possible because it is easier to change a decision that has not been made. This type of software design/development emphasizes

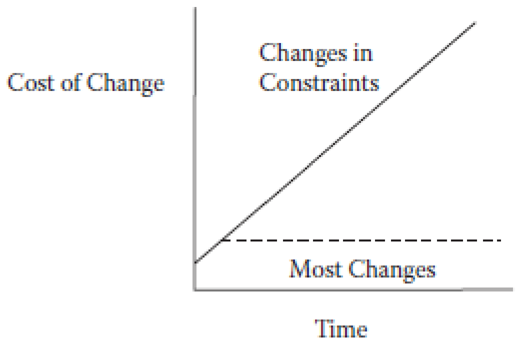


FIGURE 2: Cost curves.

Designing and managing changes throughout the life cycle. Better understanding of software engineering and quick delivery to customers benefits the concepts to improve processes and increases quality according to the following principals:

- Early software product development
- Elimination of wasted time
- Understanding and working to software requirements
- Meeting customer expectations and deliveries on time
- Achieving and implementing team goals
- Shortened design and test software life cycles

Lean Software Configuration Management

The traditional software configuration management (SCM) practice involves the identification of systems and software design/development and providing configuration control. Selected work products and the descriptions to maintain traceability of those configurations are key points throughout the software life cycle. The Lean concept is the process to compare common information with Agile software development.

AGILE SOFTWARE PROCESSES

The implementation of Agile software processes, principles, practices, and software design/development deliveries of work products to customers does provide fewer defects. Application of Agile methodologies supports numerous initiatives and provides a program and project with a manager’s approach to emphasize short-term program and project planning. The adaptability to changing requirements as well as close collaboration with customers and affected teams show accountability. The Agile management model consistently depicts processes as shown in Figure 3.

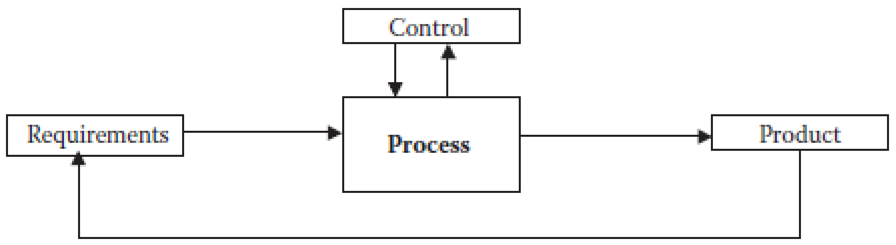


FIGURE 3: Agile management model.

The Agile model adopts values that are consistently making decisions that may cause a rejection of a software design. Agile models are more effective than the traditional (i.e., waterfall, spiral, etc.) models due to perfection versus good-enough concepts for software design practices. The software design engineer using Agile concepts has the capability to understand information first before jumping into software design/development activities.

The current state of the economy changes each day. We must resolve the software engineering approach to adapt to Lean processes and meet the needs of programs and projects. The four key elements for Agile software engineering are:

- The team has control of work assignments
- Communication with team members and customers is needed
- Change is good: “Think outside the box”
- Customer satisfaction and expectations are achieved

The Agile process method for team efforts reflects how a team of software people work together. An Agile process continually improves processes that are not working or are causing major delays in the software design/development environment. Internal program and project managers try to keep the team together by allowing decisions, expectations, and a commitment to show results. When the Agile team working its own processes at times does discover problems, the team will stay the course to solve problems that could have an impact on these processes.

The Agile method is also about continuous incremental delivery of products such as software and systems to other program and project team members and the customer. The Agile team explores and evaluates work product needs and requirements. The planning and analyzing of what to build and defining acceptance are an advantage of testing software and coordinating efforts that feed from one team member to another. Whichever Agile or Lean framework, method, or techniques are used, they employ such things as:

- Data models
- Rules of engagement
- Guides or maps

- Agile team rules

These items can be helpful as teams explore the designs and builds that are prepared for software and systems integration tests to be conducted and performed.

Agile provides team interactions that deal with processes and tools. Performance through team members boosts accountability for results and shared responsibilities for team effectiveness. Strategies, processes, and practices improve effectiveness and reliability. A successful Agile team stays alert to change and will adjust strategies and practices to match.

◀ Preliminary Activity for Week 4

Jump to...

Analysis, Application, and Exploration for Week 4 ▶

Navigation

- Home
- Dashboard
  - Site pages
  - My courses
    - Capstone Project 1
    - Network Attacks: Detection, Analysis & Counter...
    - Ojt/Practicum 1
    - Social And Professional Issues
    - System Integration And Architecture 2
      - Participants
      - General
      - 01 Introduction To Software And Systems Integration
      - 02 Program And Project Planning
      - 03 System Design
      - 04 System Design (Cont.)
      - Preliminary Activity for Week 4
      - **Lesson Proper for Week 4**
      - Analysis, Application, and Exploration for Week 4
      - Generalization for Week 4
      - Evaluation for Week 4
      - Assignment for Week 4
  - Courses

Fair Warning

NOTICE: Please be reminded that it has come to the attention of the Publishing Team of eLearning Commons that learning materials published and intended for ***free use only by students and faculty members within the eLearning Commons network were UNLAWFULLY uploaded in other sites without due and proper permission***.

PROSECUTION: Under Philippine law (Republic Act No. 8293), copyright infringement is punishable by the following: Imprisonment of between 1 to 3 years and a fine of between 50,000 to 150,000 pesos for the first offense. Imprisonment of 3 years and 1 day to six years plus a fine of between 150,000 to 500,000 pesos for the second offense.

COURSE OF ACTION: Whoever has maliciously uploaded these concerned materials are hereby given an ultimatum to take it down within 24-hours. Beyond the 24-hour grace period, our Legal Department shall initiate the proceedings in coordination with the National Bureau of Investigation for IP Address tracking, account owner identification, and filing of cases for prosecution.

Activities

- Assignments
- Forums
- Quizzes
- Resources