Applied Software Project Management

Design and Programming

Review the Design

- Once the SRS has been approved, implementation begins. Programming teams have many options:
  - The programmers can simply start building the code and create the objects and user interface elements.
  - Designers can build a user interface prototype to demonstrate to the users, stakeholders and the rest of the team. Any code used to develop the prototype is typically thrown away once the design has been finalized.
  - Pictures, flow charts, data flow diagrams, database design diagrams and other visual tools can be used to determine aspects of the design and architecture.
  - An object model can be developed on paper, either using code, simple class diagrams or Unified Modeling Language (UML) diagrams.
  - A written design specification may be created, which includes some or all of the tools above.

Review the Design

- Design tasks should always include reviews, even when there is no written design specification.
- Any written documentation should be reviewed and, if possible, inspected.
  - It is important that the reviews and inspections reach the correct audience.
  - Many users who have important input for the user interface may be uninterested or confused by object models and UML diagrams.

Version Control

- A *version control system* allows programmers to keep track of every revision of all source code files
  - The main element of the version control system is the *repository*, a database or directory which contains each of the files contained in the system.
  - A programmer can pick a point at any time in the history of the project and see exactly what those files looked like at the time.
  - It is always possible to find the latest version of any file by retrieving it from the repository.
  - Changing a file will not unexpectedly overwrite any previous changes to that file; any change can be rolled back, so no work will accidentally be overwritten.

Version Control

- There are two common models for version control systems
  - In a copy-modify-merge system, multiple people can work on a single file at a time.
    - When a programmer wants to update the repository with his changes, he retrieves all changes which have occurred to the checked out files and reconciles any of them which conflict with changes he made before updating the repository.
  - In a lock-modify-unlock system, only one person can work on any file at a time.
    - A programmer must check a file out of the repository before it can be modified. The system prevents anyone else from modifying any file until it is checked back in.
    - On large projects, the team can run into delays because one programmer is often stuck waiting for a file to be available.

Refactoring

- *Refactoring* is a programming technique in which the design of the software is improved without changing its behavior.
  - There are many choices that programmers make which do not affect the behavior of the software but which can have an enormous impact on how easy the code is to read and understand.
  - Refactoring works especially well during code reviews.
  - Because refactoring is a change to the design, it may impact the design review process. If previously reviewed code is refactored, changes to that should be distributed to the review team.

Unit Testing

- Before a build is delivered, the person or program building the software should execute *unit tests* to verify that each unit functions properly.
  - All code is made up of a set of objects, functions, modules or other non-trivial *units*. The purpose of unit testing is to create a set of tests for each unit to verify that it performs its function correctly.
  - Programmers create *suites* of unit tests, where each test is a small block of code which exercises a specific behavior of one unit.
  - The most common (and effective) way for programmers to do unit testing is to use a *framework*, a piece of software that automatically runs the tests and reports the results.

Everyone is responsible for quality

- There are different kinds of testing that serve different purposes.
  - Programmers use unit tests is to verify that the software works exactly as the programmer intended.
  - Software testers are responsible for verifying that the software meets its requirements (in the SRS) and the needs of the users and stakeholders (in the Vision and Scope Document).
    - Many defects arise when a programmer delivers software that worked as he intended, but did not meet the needs of the users. Software testers can catch these problems.

Everyone is responsible for quality

- Many programmers are confused about exactly what it is that software testers do.
  - All they know is that they deliver a build to the QA team. The QA people run the program and find bugs, which the programmers fix.
  - It is often hard for them to figure out where unit testing ends and functional testing begins.
  - The project manager should watch for this confusion, and help to clarify it by making sure that the programmers understand what kinds of testing are expected of them.

Project Automation

- Many quality problems happen because the team does not build the software consistently, and loses track of the health of the code.
- Effective project automation reduces these errors. The team can adopt a tool which:
  - Retrieves the latest build from the version control system, builds it, copies it to a folder, and reports any build warnings or errors
  - Runs unit tests, generating a test report and reporting critical failures
  - Runs automated code review tools, reporting any warnings or rule violations
  - Lists any changes which have been committed and by whom, including links to code listings for the changes