SOFTWARE ENGINEERING AND PROJECT MANAGEMENT

Question: Difference between Verification and Validation?
Answer: Verification ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, walkthroughs and inspection meetings. Validation ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after verifications are completed.

Question: Difference between Verification and Validation:
Answer: Verification takes place before validation, and not vice versa. Verification evaluates documents, plans, code, requirements, and specifications. Validation, on the other hand, evaluates the product itself. The inputs of verification are checklists, issues lists, walkthroughs and inspection meetings, reviews and meetings. The input of validation, on the other hand, is the actual testing of an actual product. The output of verification is a nearly perfect set of documents, plans, specifications, and requirements document. The output of validation, on the other hand, is a nearly perfect, actual product.

Question: What is SCM?
Answer:
Supply chain management (SCM) is the oversight of materials, information, and finances as they move in a process from supplier to manufacturer to wholesaler to retailer to consumer. Supply chain management involves coordinating and integrating these flows both within and among companies. It is said that the ultimate goal of any effective supply chain management system is to reduce inventory (with the assumption that products are available when needed). As a solution for successful supply chain management, sophisticated software systems with Web interfaces are competing with Web-based application service providers (ASP) who promise to provide part or all of the SCM service for companies who rent their service.

Question: What is Software Matrix?
Answer:
Software-Matrix.com was created to help find and download new software faster, and easier. We list free, freeware, shareware, and commercial titles from our own archive and from other search engines as well. Our unique screen layout enables you to scan through three to four times more programs on each screen, drastically speeding up software searches. A minimalist approach has been taken, removing all unnecessary clutter from the search result pages, again with the goal of speed, and ease of use in mind.

Question: Discuss the terms Quality and consistency of Software.
Answer:

Quality: In the context of software engineering, software quality refers to two related but distinct notions that exist wherever quality is defined in a business context:

- **Software functional quality** reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product\(^1\).
- **Software structural quality** refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, the degree to which the software was produced correctly.

Structural quality is evaluated through the analysis of the software inner structure, its source code, in effect how its architecture adheres to sound principles of software architecture. In contrast, functional quality is typically enforced and measured through software testing.

Consistency: When our inner systems (beliefs, attitudes, values, etc.) all support one another and when these are also supported by external evidence, then we have a comfortable state of affairs. The discomfort of cognitive dissonance occurs when things fall out of alignment, which leads us to try to achieve a maximum practical level of consistency in our world.

We also have a very strong need to believe we are being consistent with social norms. When there is conflict between behaviors that are consistent with inner systems and behaviors that are consistent with social norms, the potential threat of social exclusion often sways us towards the latter, even though it may cause significant inner dissonance.

Question No. What is the Software Engineering Problem?

Answer: Here is a simple software engineering problem that I have encountered the last few days. It's not something dramatic, but one that has made me stop development in search of elegant solutions. I am posting this here because I would like to see how functional programming languages would solve this problem.

Here is the problem: I am writing a GUI toolkit that reuses the Win32 API wrapped up in a set of C++ classes (the reason is that there is no GUI library that completely hides Win32 will reusing it - existing libraries either follow Win32 logic (ala WxWidgets) or provide their own implementation (ala Qt/Swing)). As you may know, Win32 is not object-oriented, nor does it have a well thought out/consistent interface. For example, although the menu bar is a screen object, it is not a window: all there is is a bunch of functions for creating a menu, redrawing it, adding menu items etc. But I want to have menus and other non-window Win32 items as widgets in the toolkit, for consistency reasons.
The problem lies in the organization of classes. I have 4 types of widgets:

1. widgets with no children and no window: for example, menu items
2. widgets with no children and window: for example, buttons
3. widgets with children and no window: for example, menus (which have menu items as children)
4. widgets with children and window: for example, forms

The object-oriented design solutions would be:

1. make non-window classes unrelated to widgets; something I don’t desire as it is a non-consistent solution (after all, a menu IS a widget)
2. make a generic Widget class that has children and all window properties and ignore those properties for non-window widgets; a nasty solution.
3. have 5 separate classes: Component, Container (extends Component), Window (extends Component), WindowedComponent (extends Component, Window), WindowedContainer (extends Container, Window), using multiple virtual inheritance or aggregation. I also don’t like this solution because of namespace pollution and because it will be seen as complex by the library’s users.
4. have 3 separate classes: Component, Container (extends Component), and template class Window that can be parameterized by the type of superclass (Container or Component). I also don’t like this solution, because a) templates are slow in compilation, b) bloat the size of code, c) all code has to be in the header file.

So what I am asking is how functional programming languages solve an issue like the above, which is an issue of code organization/clarity/reuse/taxonomy. I can’t seem to find a good object-oriented solution to the problem, nor any of my colleagues/friends can. So I am asking if other programming paradigms have a better solution for this problem.

**Question** Discuss the contents of SRS documentation and it’s importance.

**Answer:**

An SRS is basically an organization’s understanding (in writing) of a customer or potential client’s system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It’s a two-way insurance policy that assures that both the client and the organization understand the other’s requirements.
requirements from that perspective at a given point in time.
The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

It’s important to note that an SRS contains functional and nonfunctional requirements only; it doesn’t offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer’s system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer’s assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document
to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.

- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of “Requirements Development,” which is the initial product development phase in which information is gathered about what requirements are needed—and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client’s current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

**Question:** What are the characteristics of software processes?

**Answer:**

Software process is a method of developing or producing software. The purpose of process:

- **Effectiveness.** - An effective process must help us produce the right product. It doesn’t matter how elegant and well-written the software, nor how quickly we have produced it. If it isn’t what the customer wanted, or required, it’s no good. The process should therefore help us determine what the customer needs, produce what the customer needs, and, crucially, verify that what we have produced is what the customer needs.

- **Maintainability.** - However good the programmer, things will still go wrong with the software. Requirements often change between versions. In any case, we may want to reuse elements of the software in other products.

**Question:** What are software agents?
Answer: In computer science, a software agent is a piece of software that acts for a user or other program in a relationship of agency, which derives from the Latin agere (to do): an agreement to act on one's behalf. Such "action on behalf of" implies the authority to decide which (and if) action is appropriate.

Question: Discuss Cyclomatic Complexity.

Answer: Cyclomatic complexity (or conditional complexity) is a software metric (measurement). It was developed by Thomas J. McCabe, Sr. in 1976 and is used to indicate the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. The concept, although not the method, is somewhat similar to that of general text complexity measured by the Flesch-Kincaid Readability Test.

Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

One testing strategy, called Basis Path Testing by McCabe who first proposed it, is to test each linearly independent path through the program; in this case, the number of test cases will equal the cyclomatic complexity of the program.

Question: What is Software Maintenance?

Answer: Software Maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.[1] A common perception of maintenance is that it merely involves fixing defects. However, one study indicated that the majority, over 80%, of the maintenance effort is used for non-corrective actions (Pigosky 1997). This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the system.

Software maintenance and evolution of systems was first addressed by Meir M. Lehman in 1969. Over a period of twenty years, his research led to the formulation of Lehman's Laws (Lehman 1997). Key findings of his research include that maintenance is really evolutionary development and that maintenance decisions are aided by understanding what happens to systems (and software) over time. Lehman demonstrated that systems continue to evolve over time. As they evolve, they grow more complex unless some action such as code refactoring is taken to reduce the complexity.