ST SUNDAY TIMES
EDUCATION TIMES
THE WISDOM UPGRADE

Education is the guardian genius of democracy.It is the only dictator that free men recognize, and the only ruler that free men require.

-Mirabeau Buonaparte Lamar

Software Architecture and Engineering – from Dog Houses to Skyscrapers

Chandika Mendis- Director Technology Virtusa Corporation Sri Lanka

During my interviews of young newcomers into the IT field, I notice that many of them have trouble differentiating "coding" from the discipline of "software engineering." This is unfortunate, for there is a hugely important difference between the two that young software engineers must appreciate. This article discusses some of these software engineering aspects, with learning's from Virtusa's Technology Office.

A statement by Booch is very relevant to the topic at hand: "Dog houses have been built. You can't build a skyscraper the way you build a dog house." The "engineering" part of software engineering becomes really important-neigh, critical-for large-scale complex projects-the software equivalent of skyscrapers. So what are the important engineering aspects of building the software equivalent of skyscrapers?

First, it's important to understand what attributes are expected from the skyscrapers of the software world. Besides accurately and fully supporting immediate business requirements, the software also needs to support future expansion and scale. Future expansion translates to both increasing business volume and changing business requirements. The software also needs to adhere to industry best practices that helps future maintainability of the code. The use of the proper software engineering practices can result in software that adheres to the above attributes, while at the same time facilitating the productive use of a large team to improve time-to-result and enhance productivity of the software development process itself.

Requirements

Requirements gathering is usually the starting point of the software engineering cycle. Verbal and informal methods of gathering requirements, while quick and useful in small projects, are insufficient for large-scale projects. At the same time, the traditional waterfall method of first gathering all the requirements before proceeding to design and development tends to also fail in a world where time-to-result is becoming a prime differentiator. Instead, modern agile development techniques tend to engage customers earlier and more frequently and also provide for shorter iterations, enabling customers to see the product earlier and decide implementation priorities. In our experience, agile methods are growing in acceptance and momentum, and are even becoming the approach of choice in offshore development projects.

Metrics, such as the Requirements Clarity Index and Requirements Stability Index, are useful measures of the requirements gathering and management processes. One interesting tool I've come across is an Automated Requirements Measurement tool from NASA. The tool can scan a requirements document and capture many requirements-related metrics; for example, the usage of weak English phrases that lead to ambiguities, structural inconsistencies, etc. The output of this tool can be very useful to optimize manual requirements reviews.

Requirements gathering includes both business requirements and technical requirements. The project architect is responsible for visualizing the technical solution, and must therefore ensure that any technical requirements and constraints are fully captured during the requirements process.

Architecture and Design

For large projects to be delivered on-time, it is important to effectively engage a large team, and parallelize as much work as possible. Software architecture provides the means to decompose a large project into smaller sub-systems or sub-projects. Decomposition involves identifying the functional modules, architectural layers and cross-cutting concerns. Architectural layers focus on specific functions, such as the user interface, business logic or access to database/external systems. By organizing these layers such that, for example, the user interface layer can only access the business logic layer, and the business logic layer can only access the persistence layer, layered architecture reduces the interconnections between components and reduces the complexity of the resulting software. Cross-cutting concerns deal with common technical requirements that go across layers, such as security, error handling, transactions, etc. A technical requirements checklist of the right questions to ask in these areas can help avoid expensive requirements gaps in new development engagements.

Defining the architecture requires broad and deep knowledge of technology choices and best practices. The team should leverage industry standards and best practices associated with its technology selections. These standards should not be limited to the main programming language, but should also address user interface (HTML, accessibility, XHTML standards, etc.), database (database object naming conventions, SQL standards, etc.), and any other specific technologies in use. Virtusa's Technology Office provides a portal with all these specific standards and best practices, from which teams can select the subset they require for each project.

In many industries, existing blueprints are used to create new applications. So rather than architecting from scratch, it is often better to use established architecture blueprints as a starting point. Creating an architecture blueprint in code during the early stages of the design process is a highly-recommended best practice that results in better quality code aligned with the architectural vision. This approach has the advantage of mitigating technical risks early in the project and describing the architecture in developers' language. This thin slice of code representing all the architectural layers can also be run through profiling tools to capture potential performance and scalability bottlenecks with the architecture.

Please turn to page 6