Discipline by Discipline: Design (article)

Discipline by Discipline: Design

One of the most important aspects of the Unified Process and what I emphasize when managing complex software development projects is to focus on an '*architecture-centric*' approach to building a system. The best way to accomplish this from my perspective is to ensure a *prioritization* of user stories/use cases based on not only business value but *architectural significance*. It's also important to remember that building architecturally significant components can still demonstrate business value at the completion of an iteration. A simple UI showing how data can be entered, stored and retrieved (even if it's not editable) shows a business stakeholder that the 'tiers' and 'layers' that exist behind the scenes are adding value. One way to ensure a balance between business needs and architectural requirements is to establish a **troika** of the Product Owner, Architect and Project Manager to review and assess the Release Plan and its prioritization.

The Design Discipline

- **Balancing the amount of effort applied to formal Design: Agile Modeling (AM)**

> *In large corporations or projects, or in safety-critical domains, a Scrum/XP approach to documentation is too lightweight. However, even under these circumstances, it is important to protect projects from having to produce unnecessary obligations such as "shelfware" artifacts. Scott Ambler's Agile Modeling approach in which artifacts are only produced if they really add value, and maintenance of intermediate artifacts is minimized, is a useful framework to manage this balance. A good reference model for determining how much 'ceremony' is required for a project can be found in an excellent book by Barry Boehm and Richard Turner Balancing Agility and Discipline: A Guide for the Perplexed.*

- **Designing for Future Change: Service-oriented Architectures (SOA)**

A Service-oriented Architecture is one in which all functions, or services, are defined using a description language and have invokable interfaces that are called to perform steps in business processes. Each such service must have unambiguous semantics (see Design By Contract), and should preferably be idempotent (repeatable without side effects). Larman's expedited UP leads smoothly to the definition of such architectures since it promotes the definition of system or component-level operations with well-defined semantics (contracts). Although this requires a degree of maturity within a development organization its benefits are significant.

- **Realizing the Full Benefits of Reuse: Product Line Architectures**

Adopting some form of Product Line Process is indispensable to the achievement of true asset reuse across a range of projects and products. The earlier model of "if we build a component repository, they will come" simply does not work. To realize reuse benefits the enterprise must combine an application engineering process with a domain engineering process. A standard recommendation is to adopt this two-level approach wherever and whenever feasible. Using commonality/variability analysis together with a full repertoire of variation point management techniques (model management and refactoring, model-driven architecture, management of variable configurations, design patterns, framework development, polymorphism and AOSD) in order to achieve this well-defined relationship between the domain engineering and application engineering processes. (Of course, one of the reasons software development has moved towards OO technologies is that they contribute importantly to this repertoire.).

- **Design by Contract (DBC)**

  Design by Contract is a simple but powerful discipline for writing specifications of use cases, services, system operations, component operations and class methods. It uses the concepts of pre conditions, post conditions and invariants to achieve a full separation of "what" from "how". The benefits of so doing are enormous, and include:

    - *Precision,*

- *Lack of ambiguity,*
- *Testability, and*
- *The ability to write short specifications even for complex implementations.*