Definitions

Architecture

Application Architecture (GC EARB) Application Architecture consists of the interaction of applications with each other and with users. It focuses less on internal mechanics and specific programming and more on overall design on how data is consumed and created by the system. It views the interactions between applications, databases, middleware to ensure scalability, reliability, availability and manageability.

Application Architecture (RedHat) An application architecture describes the patterns and techniques used to design and build an application. The architecture gives you a roadmap and best practices to follow when building an application, so that you end up with a well-structured app. [Redhat - CNA - What is Application Architecture].

Application Architecture (TOGAF) A description of the structure and interaction of the applications as groups of capabilities that provide key business functions and manage the data assets. [Application Architect - Wikipedia.]

Architecture Quotes : - *Architecture is the decisions that you wish you could get right early in a project, product or project lifecycle* - Ralph Johnson & Martin Fowler - *Architecture is about the important stuff, whatever that is.* - Ralph Johnson & Martin Fowler - *Architecture is the stuff you can't Google.* - Mark Richards

Architecture Style (TOGAF) The combination of distinctive features related to the specific context within which architecture is performed or expressed; a collection of principles and characteristics that steer or constrain how an architecture is formed.

Types of Architecture The overall architecture of an enterprise can be described by integrated sub-architecture domains. These are:

- Business Architecture
- Application Architecture
- Information Architecture
- Technology Architecture
- Security Architecture
- Privacy Architecture and
- Data Architecture

Architecture Characteristics:

- Architecture characteristics are the aspects the system must do that is not directly related to the domain functionality. These are often called non-functional requirements but should be considered as Quality Requirements.

- An architectural characteristics meets three criteria:
  1. specifies a non-domain (*non-functional*) consideration,
  2. influences some aspect of the design, and
  3. is critical/important to the application's success. A few are listed below
- Examples of architectural characteristics:
  - operational characteristics: availability, business continuity, performance, recoverability, robustness, scalability, elasticity.
  - structural characteristics: configurability, extensibility, installability, reusability, localization, maintainabilty, portability, supportability, upgradeability.
  - cross-cutting: authentication, authorization, legal, privacy, security, supportabilty, usability, achievability, compatibilty, accessibility, interoperability.
  - See *Neal Ford's Presentation with List of Quality Attributes* for more information

Governance:

- Governance, derived from the Greek word kubernan "*to steer*" is an important responsibility of the architect role. As the name implies, the scope of architecture governance covers any aspect of the software development process that architects (including roles like enterprise architects) want to exert an influence upon. For example, ensuring software quality within an organization falls under the heading of architectural governance because it falls within the scope of architecture, and negligence can lead to disastrous quality problems.

Technical Debt

Technical debt is somewhat misunderstood in within IT and our department. Technical debt goes beyond having aging end-of-life applications. Below are some common definitions used to describe technical debt:

- Technical debt (also known as design debt or code debt, but can be also related to other technical endeavours) is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer. [Technical Debt - Wikipedia].
- Technical Debt is a term coined thirty years ago by Ward Cunningham:

  Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the *cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.* [Ward Cunningham - 1992].

Note: Ward Cunningham is one of the authors of the Agile Manifesto

- The sum of time and effort one has to pay to keep up with the tools one is 'married' to.
- Changes required that are not completed are considered debt, and until paid, will incur interest on top of interest, making it cumbersome to build a project. As a change is started on a codebase, there is often the need to make other coordinated changes in other parts of the codebase, system, solution or documentation. Although the term is used in software development primarily, it can also be applied to other professions.
- Technical debt is a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution. - *Technical Debt - Technopedia*
- Technical debt is commonly associated with extreme programming, especially in the context of refactoring. That is, it implies that restructuring existing code (refactoring) is required as part of the development process. Under this line of thinking refactoring is not only a result of poorly written code, but is also done based on an evolving understanding of a problem and the best way to solve that problem.
- Technical debt may also be known as design debt.
- When taking short cuts and delivering code that is not quite right for the programming task of the moment, a development team incurs Technical Debt. This debt decreases productivity. This loss of productivity is the interest of the Technical Debt. - *Technical Debt Metaphor - Agile Alliance*