Why Software Architecture is Important?

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
CSIS, Work Integrated Learning Programs

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SEZG651/ SSZG653 Software Architectures CS01/02A

Why is Software Architecture Important?

1. An architecture will inhibit or enable a system's driving quality attributes.
2. The decisions made in an architecture allow you to reason about and manage change as the system evolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances communication among stakeholders.
5. The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. An architecture defines a set of constraints on subsequent implementation.
7. The architecture dictates the structure of an organization, or vice versa.
8. An architecture can provide the basis for evolutionary prototyping.
9. An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.
10. An architecture can be created as a transferable, reusable model that form the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.
12. By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.
13. An architecture can be the foundation for training a new team member.

Inhibiting or Enabling a System's Quality Attributes

Whether a system will be able to exhibit its desired (or required) quality attributes is substantially determined by its architecture.

This is the most important message of this course!

- Performance: You must manage the time-based behavior of elements, their use of shared resources, and the frequency and volume of inter-element communication.
- Modifiability: Assign responsibilities to elements so that the majority of changes to the system will affect a small number of those elements.
- Security: Manage and protect inter-element communication and control which elements are allowed to access which information; you may also need to introduce specialized elements (such as an authorization mechanism).
- Scalability: Localize the use of resources to facilitate introduction of higher-capacity replacements, and you must avoid hardcoding in resource assumptions or limits.
- Incremental subset delivery: Manage inter-component usage.
- Reusability: Restrict inter-element coupling, so that when you extract an element, it does not come out with too many attachments to its current environment.

Reasoning About and Managing Change

innovate achieve lead

- About 80 percent of a typical software system's total cost occurs after initial deployment
  - accommodate new features
  - adapt to new environments,
  - fix bugs, and so forth.
- Every architecture partitions possible changes into three categories
  - A *local* change can be accomplished by modifying a single element.
  - A *nonlocal* change requires multiple element modifications but leaves the underlying architectural approach intact.
  - An *architectural* change affects the fundamental ways in which the elements interact with each other and will probably require changes all over the system.
- Obviously, local changes are the most desirable
- A good architecture is one in which the most common changes are local, and hence easy to make.

Predicting System Qualities

- If we know that certain kinds of architectural decisions lead to certain quality attributes in a system, we can make those decisions and rightly expect to be rewarded with the associated quality attributes.
- When we examine an architecture we can look to see if those decisions have been made, and confidently predict that the architecture will exhibit the associated qualities.
- The earlier you can find a problem in your design, the cheaper, easier, and less disruptive it will be to fix.

Enhancing Communication Among Stakeholders

innovate achieve lead

- Software architecture represents a common abstraction of a system that most, if not all, of the system's stakeholders can use as a basis for creating mutual understanding, negotiating, forming consensus, and communicating with each other.
- The architecture—or at least parts of it—is sufficiently abstract that most nontechnical people can understand it adequately.
- Each stakeholder of a software system—customer, user, project manager, coder, tester, and so on—is concerned with different characteristics of the system that are affected by its architecture. For example:
  - The user is concerned that the system is fast, reliable, and available when needed.
  - The customer is concerned that the architecture can be implemented on schedule and according to budget.
  - The manager is worried (in addition to concerns about cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways.
  - The architect is worried about strategies to achieve all of those goals.
- Architecture provides a common language in which different concerns can be expressed, negotiated, and resolved at a level that is intellectually manageable even for large, complex systems.

Earliest Design Decisions

innovate achieve lead

- Software architecture is a manifestation of the earliest design decisions about a system.
- These early bindings carry enormous weight with respect to the system's remaining development, its deployment, and its maintenance life.
- Each decision constrains the many decisions that follow.
- What are these early design decisions embodied by software architecture?
  - Will the system run on one processor or be distributed across multiple processors?
  - Will the software be layered? If so, how many layers will there be? What will each one do?
  - Will components communicate synchronously or asynchronously? Will they interact by transferring control or data or both?
  - Will the system depend on specific features of the operating system or hardware?
  - Will the information that flows through the system be encrypted or not?
  - What communication protocol will we choose?
- Imagine the nightmare of having to change any of these decisions.

Defining Constraints on an Implementation

innovate achieve lead

- An implementation exhibits an architecture if it conforms to the design decisions prescribed by the architecture.
  - The implementation must be implemented as the set of prescribed elements
  - These elements must interact with each other in the prescribed fashion
  - Each element must fulfill its responsibility to the other elements as dictated by the architecture.
- Each of these prescriptions is a constraint on the implementer.
- Element builders may not be aware of the architectural tradeoffs—the architecture (or architect) simply constrains them in such a way as to meet the tradeoffs.
  - Example: an architect assigns performance budget to the pieces of software involved in some larger piece of functionality.
  - If each software unit stays within its budget, the overall transaction will meet its performance requirement.
  - Implementers of each of the constituent pieces may not know the overall budget, only their own.

Influencing the Organizational Structure

- Architecture prescribes the structure of the system being developed.
- That structure becomes engraved in the structure of the development project (and sometimes the structure of the entire organization).
- The architecture is typically used as the basis for the work-breakdown structure.
- The work-breakdown structure in turn dictates
  - units of planning, scheduling, and budget
  - interteam communication channels
  - configuration control and file-system organization
  - integration and test plans and procedures;
  - much more
- The maintenance activity will also reflect the software structure, with teams formed to maintain specific structural elements from the architecture.
- If these responsibilities have been formalized in a contractual relationship, changing responsibilities could become expensive or even litigious.

Enabling Evolutionary Prototyping

innovate achieve lead

- Once an architecture has been defined, it can be analyzed and prototyped as a skeletal system.
  - A skeletal system is one in which at least some of the infrastructure— how the elements initialize, communicate, share data, access resources, report errors, log activity, and so forth—is built before much of the system's functionality has been created.
- This approach aids the development process because the system is executable early in the product's life cycle.
- The fidelity of the system increases as stubs are instantiated, or prototype parts are replaced with complete versions of these parts of the software.
- This approach allows potential performance problems to be identified early in the product's life cycle.
- These benefits reduce the potential risk in the project.

Improving Cost and Schedule Estimates

- One of the duties of an architect is to help the project manager create cost and schedule estimates early in the project life cycle.
- Top-down estimates are useful for setting goals and apportioning budgets.
- Cost estimations that are based on a bottom-up understanding of the system's pieces are typically more accurate than those that are based purely on top-down system knowledge.
  - Each team or individual responsible for a work item will be able to make more-accurate estimates for their piece than a project manager and will feel more ownership in making the estimates come true.
- The best cost and schedule estimates will typically emerge from a consensus between the top-down estimates (created by the architect and project manager) and the bottom-up estimates (created by the developers).

Transferable, Reusable Model

- Reuse of architectures provides tremendous leverage for systems with similar requirements.
  - Not only can code be reused, but so can the requirements that led to the architecture in the first place, as well as the experience and infrastructure gained in building the reused architecture.
  - When architectural decisions can be reused across multiple systems, all of the early-decision consequences are also transferred.
- A software product line or family is a set of software systems that are all built using the same set of reusable assets.
  - Chief among these assets is the architecture that was designed to handle the needs of the entire family.
  - The architecture defines what is fixed for all members of the product line and what is variable.
  - The architecture for a product line becomes a capital investment by the organization.

Using Independently Developed Components

- Architecture-based development often focuses on components that are likely to have been developed separately, even independently, from each other.
- The architecture defines the elements that can be incorporated into the system.
- Commercial off-the-shelf components, open source software, publicly available apps, and networked services are example of interchangeable software components.
- The payoff can be
  - Decreased time to market
  - Increased reliability (widely used software should have its bugs ironed out already)
  - Lower cost (the software supplier can amortize development cost across their customer base)
  - Flexibility (if the component you want to buy is not terribly specialpurpose, it's likely to be available from several sources, thus increasing your buying leverage)

Restricting Design Vocabulary

- As useful architectural patterns are collected, we see the benefit in voluntarily restricting ourselves to a relatively small number of choices of elements and their interactions.
  - We minimize the design complexity of the system we are building.
  - Enhanced reuse
  - More regular and simpler designs that are more easily understood and communicated
  - More capable analysis
  - Shorter selection time
  - Greater interoperability.
- Architectural patterns guide the architect and focus the architect on the quality attributes of interest in large part by restricting the vocabulary of design.
  - Properties of software design follow from the choice of an architectural pattern.

Basis for Training

- The architecture can serve as the first introduction to the system for new project members.
- Module views are excellent for showing someone the structure of a project
  - Who does what, which teams are assigned to which parts of the system, and so forth.
- Component-and-connector views are excellent for explaining how the system is expected to work and accomplish its job.

Summary

1. An architecture will inhibit or enable a system's driving quality attributes.
2. The decisions made in an architecture allow you to reason about and manage change as the system evolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances communication among stakeholders.
5. The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. An architecture defines a set of constraints on subsequent implementation.
7. The architecture dictates the structure of an organization, or vice versa.

Summary

8. An architecture can provide the basis for evolutionary prototyping.
9. An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.
10. An architecture can be created as a transferable, reusable model that form the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.
12. By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.
13. An architecture can be the foundation for training a new team member.