SOFE 3650

Software Design and Architectures

Introduction

Dr. Ramiro Liscano

SIRC 3360

Ramiro.Liscano@ontariotechu.ca

OntarioTech UNIVERSITY

Learning Objectives

- To review “good” software design principles
  - Overlaps with what you learned in SOFE 2720
- To get an overview of what Software Architecture consists of.
- To understand why Software Architecture Matters

Software Design

- Jack Mostow[1] suggests that the purpose of software design is to construct a solution that:
  - Satisfies a given purpose
  - Conforms to limitations of the target medium
  - Meets implicit or explicit requirements on performance and resource usage
  - Satisfies implicit or explicit design criteria on the form of the artifact
  - Satisfies restrictions on the design process itself, such as its length or cost, or the tools available for doing the design.

OntarioTech UNIVERSITY

[1]J. Mostow, Towards better models of the design process, AI Magazine, 6(1), 1985.

Software Design Trade-offs

- Since these goals are very often in conflict, it's clear that the designer is faced with an optimization problem in which no option is clearly better than any other.
- These tradeoffs force the designer to choose between various options at multiple points along the way.
  - The goal of software engineering design is finding the best solution to the problem given the goals and constraints of the project.

Designing is an Experiment

- Designing and writing a piece of software is an experiment.
- Engineering is thus a succession of hypotheses, each built upon the failure of all prior hypotheses.
  - Each iteration is tested or analyzed in an effort to understand how the design will behave under real or imagined conditions.
  - If the new design fails, the failures are found, components redesigned, and the analysis is performed again.

Engineering is not “anti-agile”

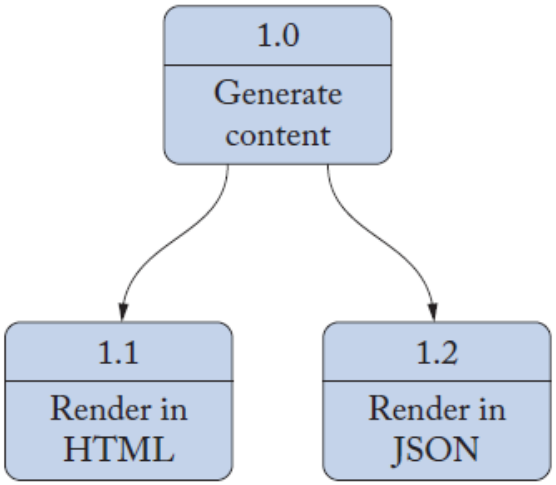
- In today’s agile software development environment there is a tendency to consider that software design is not part of the agile process.
  - **But it is!!!**
  - **Design happens whether you want it to or not!!**
- The agile process simply tries to emphasize that design and coding go hand in hand.

Key Software Engineering Principles

- Martin[2] published five "key" SE principles.

1. SINGLE RESPONSIBILITY

A class should have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.



[2]R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, p. 58, Prentice Hall, New York, 2018.

OntarioTech UNIVERSITY

Key Software Engineering Principles

2. OPEN-CLOSED

A software artifact should be open to extension and closed for modification.

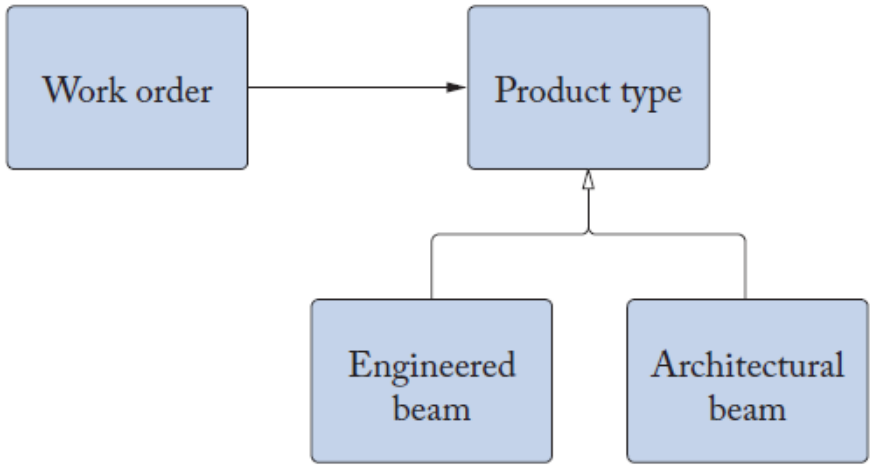
- Should be able to modify the behavior of an artefact without having to modify it.
- Seems rather challenging but can be done through extensions of the original artefact.
  - In a language like Java this would correspond to one Class extending another.

Key Software Engineering Principles

3. LISKOV SUBSTITUTION

*If for each object **o1** of type **S** there is an object **o2** of type **T** such that for all programs **P** defined in terms of **T**, the behavior of **P** is unchanged when **o1** is substituted for **o2**, then **S** is a subtype of **T**.*

- In OO-languages this principal is the basis of polymorphism in inheritance.
- In the example, Work order uses the Product Type abstraction.
- Any sub-type (Engineered or Architectural Beam) will work with Work order without modifying it.



OntarioTech UNIVERSITY

Key Software Engineering Principles

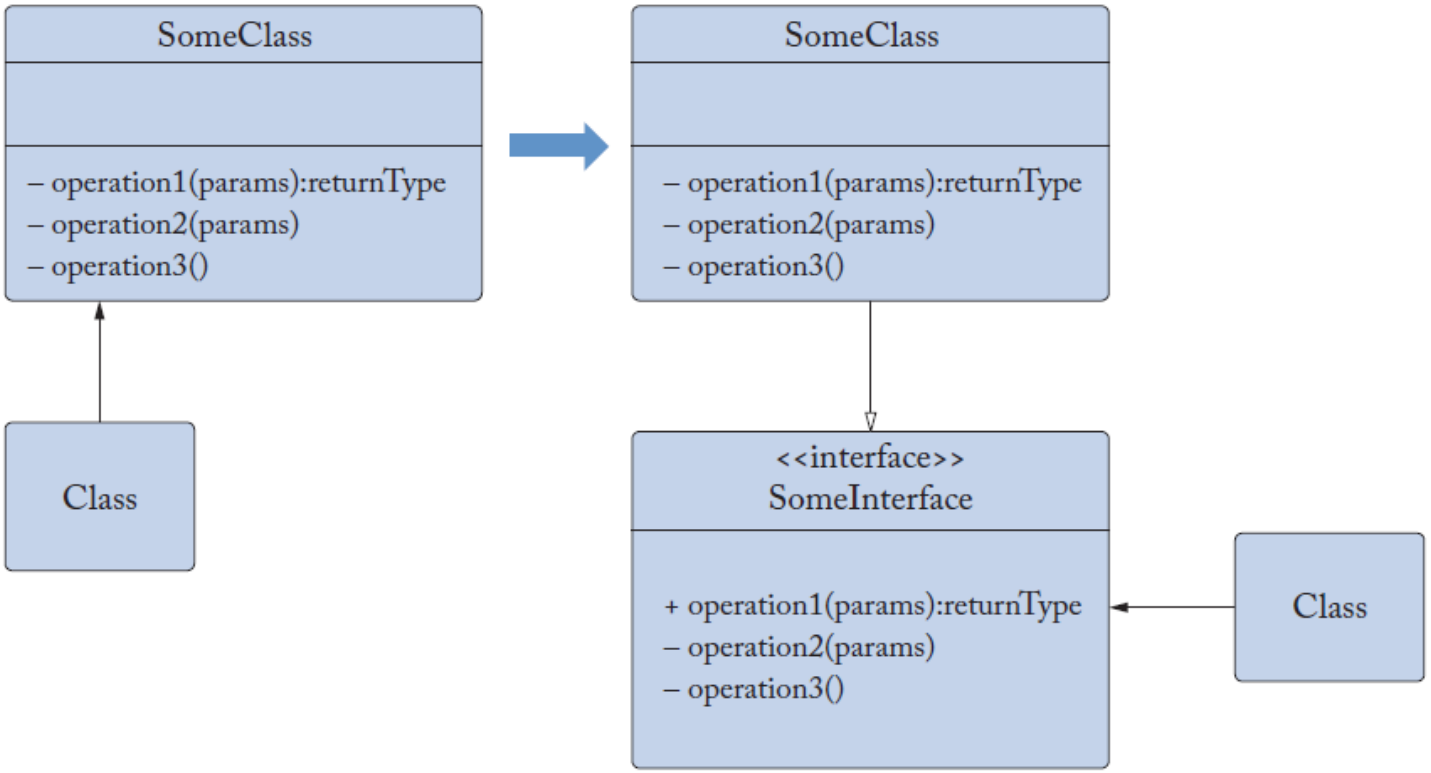
4. INTERFACE SEGREGATION

Implementation of an abstraction should be separated from the interface to that abstraction

- The goal is to decouple the users of an abstraction from changes to the implementation of that abstraction, leaving them vulnerable only to changes in the interface (which can't be helped and happen far less often).
- If you as the architect focus on the interfaces between your components and merely describe the components as black boxes, you can limit the degree to which the builders of those components can break other components.
- **Violating this principle can have real-world costs!!**

Key Software Engineering Principles

4. INTERFACE SEGREGATION



OntarioTech UNIVERSITY

Key Software Engineering Principles

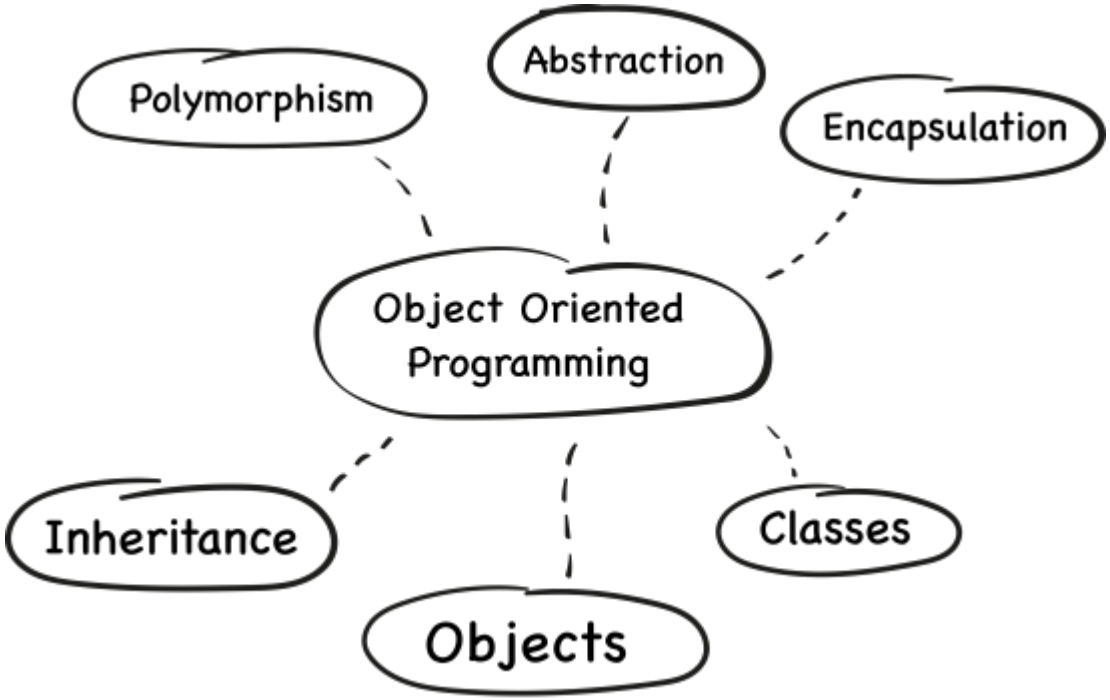
5. DEPENDENCY INVERSION

The most flexible designs are those in which source code dependencies refer only to abstractions.

- Modules that make use of code in another module should refer only to the definition of the abstract interface.
- Consequently, a module should only import abstract interfaces (*include* in C and C++, *import* in Python, and *use* in Java) of the modules it uses.
- This is typically a difficult principle to adhere to especially when 3rd party libraries are leveraged.

Key Software Engineering Principles

- These 5 particular properties are supported in OO programming languages through encapsulation, abstraction, polymorphism, and inheritance



OntarioTech UNIVERSITY

Languages that Support OOP

- Some top programming languages that support OO are: Java, C#, Ruby, Python, TypeScript, and PHP.
- All of these languages have their own syntax but the principles are the same.
- In non-OOP languages, OO properties such as encapsulation, abstraction, polymorphism, and inheritance can be implemented but it is not simple.
  - See Does procedural language have design patterns? - Stack Overflow

Key Data Related Principles

- Along with Martin's programming principles there are also 2 very important data focused principles.
  - One copy of data
  - Keep business logic in its place.

Key Data Related Principles

1. ONE COPY OF DATA

Keep only a single copy of each data element in your data store.

- If you have more than one copy of something in your data, one of them will be wrong.
- **Data consistency is a common problem in many projects !!**

OntarioTech UNIVERSITY

Key Data Related Principles

2. KEEP BUSINESS LOGIC IN ITS PLACE

Business logic belongs in the code, specifically in the controllers and models that support the views.

- Keeping your layers separate will make your life easier in the long run.
- Example: Some DB have the ability to define procedures and triggers in the DB.
- Putting business logic in the database violates the Layers pattern and eventually leads to complex code.

Summary of Software Design Principles

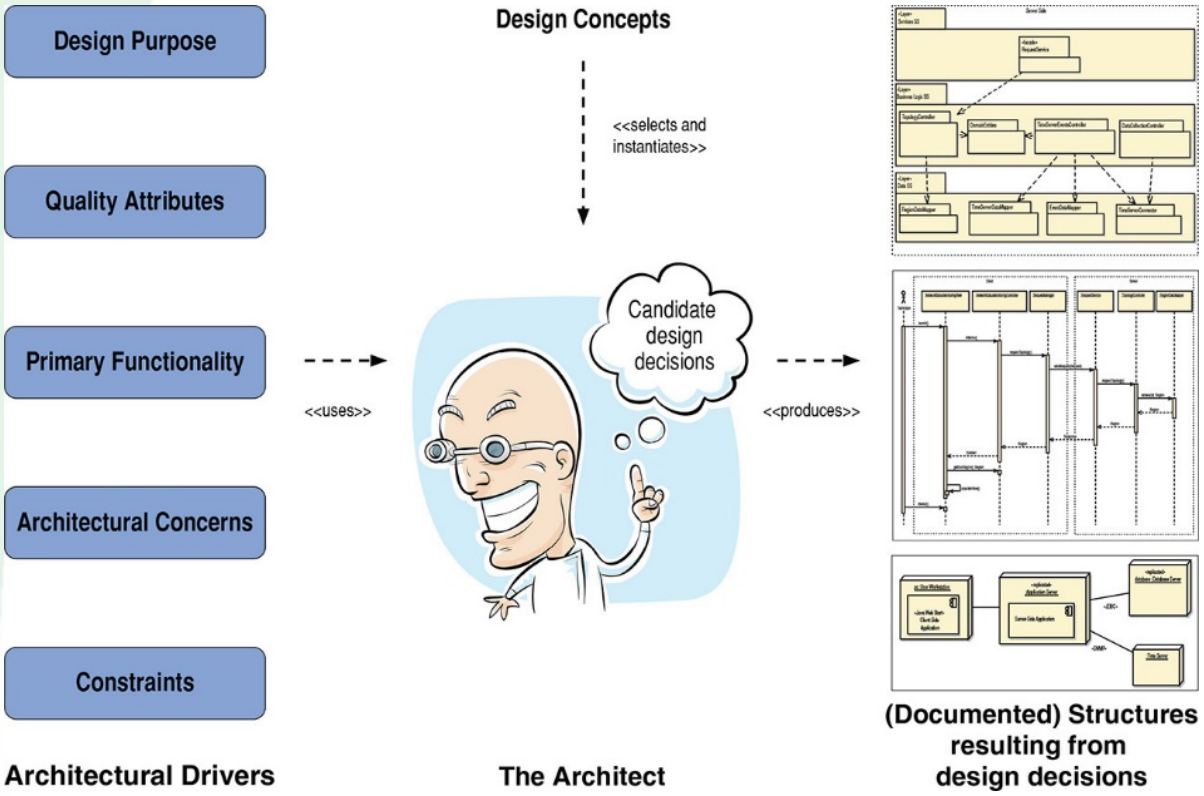
- "Good" software design principles are important for achieving maintainable and affordable software.
- We reviewed the following 7 SE principles:
  - Single Responsibility
  - Open-Closed
  - Liskov Substitution
  - Interface Segregation
  - Dependency Inversion
  - One Copy of Data
  - Keep Business Logic in its Place
- Architectural design tries to follow these principles.

What is Software Architecture

OntarioTech UNIVERSITY

Software Architecture Design

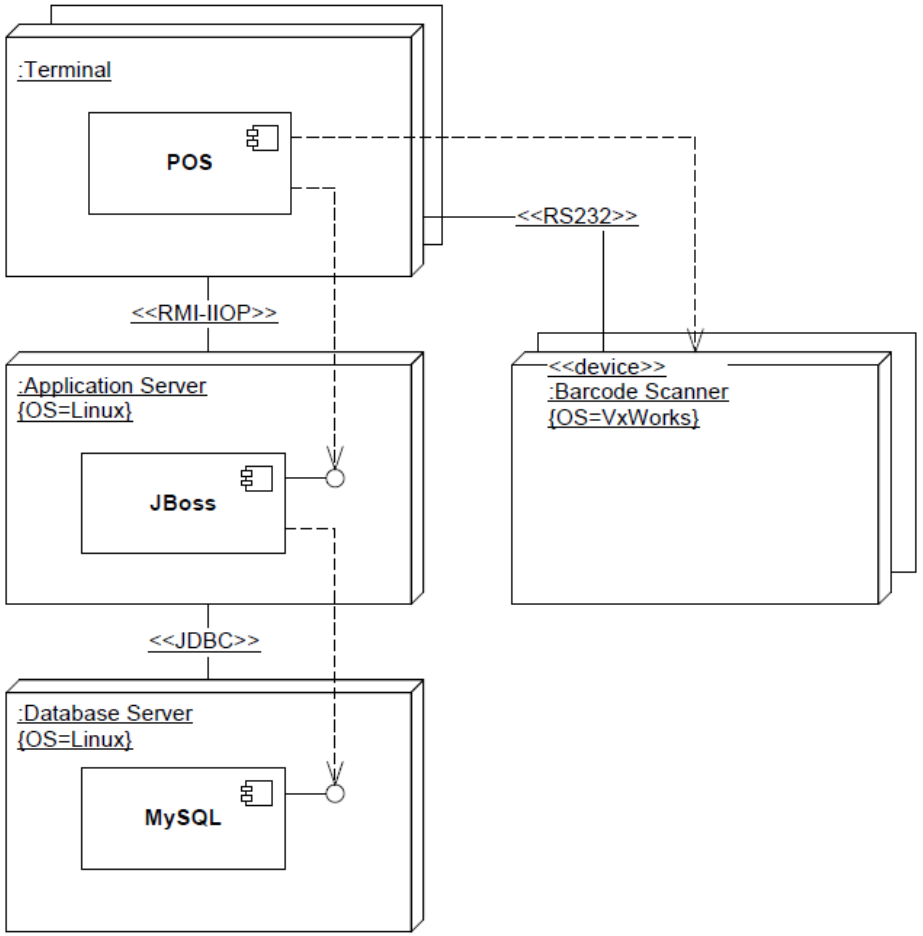
- Software architecture design consist of a process that results is a set of artifacts that helps a software developer understand the overall software applications.



OntarioTech UNIVERSITY

Software Architecture Artifacts

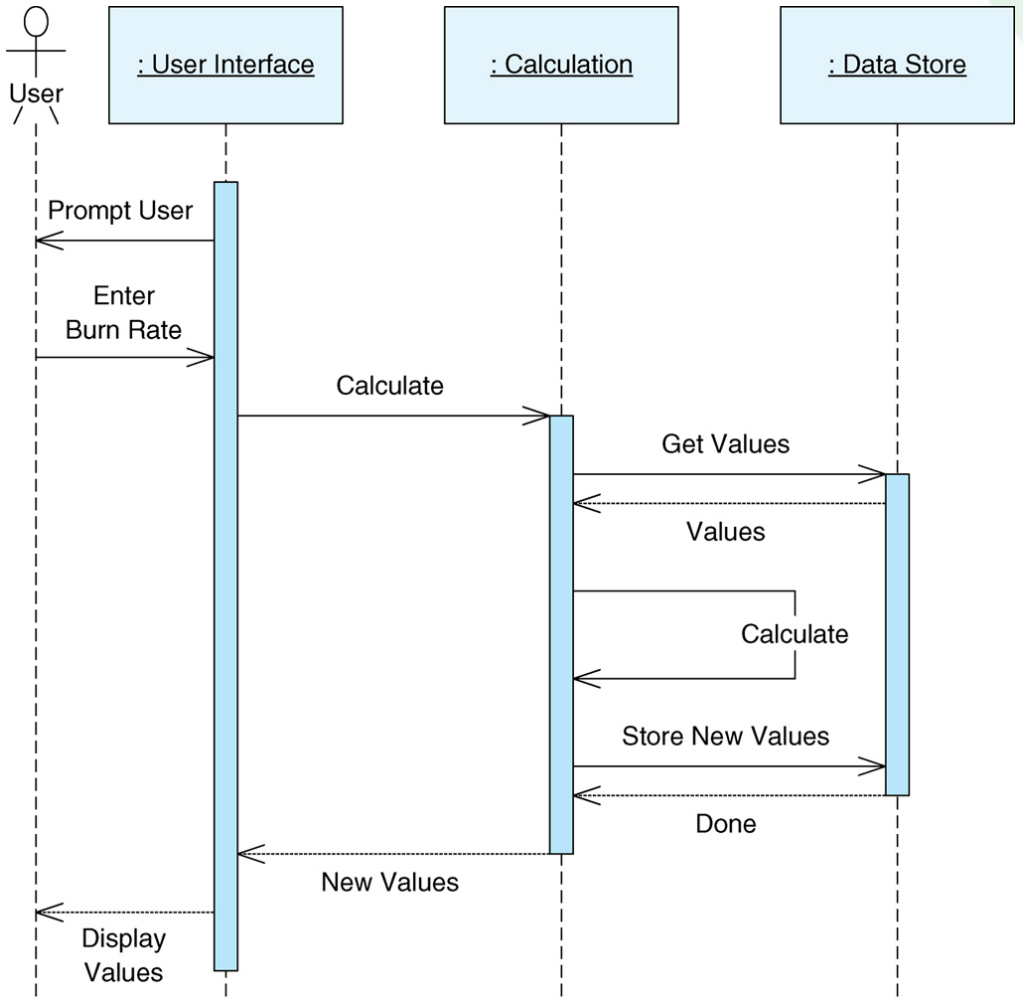
- Documentation consisting of:
  - Diagrams
  - Component descriptions.
- The diagrams capture different high-level views of the structure of the application related to components (single responsibility) and their associations (interfaces)



OntarioTech UNIVERSITY

Software Architecture Artifacts

- The diagrams can also capture behavior among components as UML sequence diagrams



OntarioTech UNIVERSITY

Software Architecture Artifacts

- The components responsibilities are documented.

Component	Responsibility	Properties
NetworkDeviceConnector	Communicate with network devices and isolate the rest of the system from specific protocol	
NetworkDeviceEventController	Process events that are received from the network devices	
Topology Controller	Provides access to the network topology information and changes in it	Type = stateless
Region Data Mapper	Manage persistence of Regions	Framework = Hibernate
NetworkStatusMonitoringView	Display the network topology and events that occur on the devices	Framework = Swing

Software Architecture Artifacts

- The components Interfaces are documented.

NetworkStatusMonitoringContoller

Method name	Description
boolean initialize()	Opens up the network representation so that users can interact with it.
Region getRootRegion()	Returns a reference to the root region

RequestService:

Method name	Description
Response sendRequest(Request)	This method receives a request. Just this method is exposed in the service interface. This simplifies the addition of other functionality in the future without having to modify the existing service interface.

Summary of Software Architecture Design

- Software Architecture Design is a process resulting in the specification of software artifacts such as components (responsibilities and interfaces) and their dependencies, coordination among the components, and deployment of the components.
- This course presents the foundations of software architecture design.

Architecture Matters

All architecture is design; not all design is architecture. The difference? If you have to start over to fix it, it's architecture. (Scott A. Whitmire)

Architecture Design Process

- There is no one way to design an architecture and there are many views on this.
- Most books on the subject, like Designing Software Architectures: A Practical Approach (Cervantes et al.), present a how to *organize the practice* of architectural design while others like, Engineer Your Software (Whitmire), want to identify the best architecture to solve the problem.

Common Design Views

- For either of these approaches there is an agreement that:
  - The problem drives the process by specifying both functional and non-functional requirements.
    - Non-functional quality requirements are key!!
  - They both take advantage of architectural patterns.
  - There are a set of problem types that can take advantage of particular frameworks.
  - There is a lot of components that can be re-used
  - Component deployment affects architecture
  - Programming style significantly influences the way you approach architectural design.

Reasons why Software Architecture is Important

- The Software Architecture in Practice[3] book lists 13 reasons why software architecture in important.
- In order to appreciate these one can relate these to any major software project such as the 4th year capstone project.
  - The 2nd report for the capstone is an architecture design report.

OntarioTech UNIVERSITY

3 Software Architecture in Practice, Len Bass, Paul Clements, Rick Kazman,

1. Architecture Influences 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.
- An example of quality attributes influenced by architecture are:
  - **Performance**: You must manage the behavior of elements, their use of shared resources, and the frequency and volume of inter-element communication in order to achieve the desired performance.
  - **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.
  - **Scalability**: Localize the use of resources to facilitate introduction of higher-capacity replacements.
  - **Reusability**: Restrict inter-element coupling.

OntarioTech UNIVERSITY

2. Managing Change

- 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.
- A good architecture is one in which the most common changes are local, and hence easy to make.

OntarioTech UNIVERSITY

3. Predicting System Qualities

- Certain kinds of architectural decisions lead to certain quality attributes in a system.
- 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.

OntarioTech UNIVERSITY

4. Enhancing Communication Among Stakeholders

- 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.
- 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.

OntarioTech UNIVERSITY

5. Earliest Design Decisions

- 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.

OntarioTech UNIVERSITY

6. Defining Constraints on the Implementation

- 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.

OntarioTech UNIVERSITY

7. Influencing the Organizational Structure

- 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
  - Inter-team communication channels
  - configuration control and file-system organization
  - integration and test plans and procedures;
  - much more
- If these responsibilities have been formalized in a contractual relationship, changing responsibilities could become expensive or even litigious.

OntarioTech UNIVERSITY

8. Evolutionary Prototyping

- Once an architecture has been defined, it can be analyzed and prototyped as a skeletal system.
- 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.

OntarioTech UNIVERSITY

9. Determining 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.
- Cost estimations that are based on understanding of the system's pieces are typically more accurate than those that are based purely on top-down system knowledge.
- 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).

OntarioTech UNIVERSITY

10. Transferable and Reusable Models

- Reuse of architectures provides tremendous leverage for systems with similar requirements.
  - When architectural decisions can be reused across multiple systems, all of the early-decision consequences are also transferred.
- A software product line of software systems are all built using the same set of reusable assets defined by an architecture.
  - 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.

OntarioTech UNIVERSITY

11. Independently Developed Components

- Architecture-based development often focuses on components that are likely to have been developed separately, even independently, from each other.
  - 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 special purpose, it's likely to be available from several sources, thus increasing your buying leverage)

OntarioTech UNIVERSITY

12. Restricting Components and Interactions

- As useful architectural patterns are collected, we see the benefit in voluntarily restricting ourselves to a relatively small number of choices of components and their interactions.
  - We minimize the design complexity of the system we are building.
  - Enhance 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.
  - Properties of software design follow from the choice of an architectural pattern.

OntarioTech UNIVERSITY

13. 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 and which teams are assigned to which parts of the system.
- Component-and-connector views are excellent for explaining how the system is expected to work and accomplish its job.

OntarioTech UNIVERSITY

Summary of Architecture Matters

- The 13 points presented of why software architecture matters are rather verbose and it is important to understand that without this step there are many aspects of a software project that can go wrong such as:
  - Simplifying the software system
  - Being able to sub-divide the work
  - Achieving the quality required in the product
  - Describing to others the system
  - Etc.