

Software Design and Architecture

Dr. Rodrigo Spínola



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



Agenda

2

- Starting on Software Design
- Software Architecture
- Architectural Design Decision Matrix
- Architectural Degradation
- Architectural Patterns
- Functions of the Software Architect
- Perceived tensions Agility-Architecture



Lecture 19 - Software
Design and Architecture

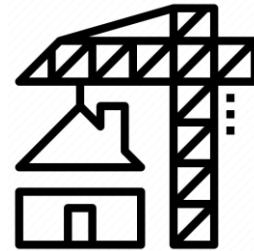
TDresearchteam
Technical Debt Research Team



3

Building Houses

- What does “designing a house” mean?
- What shape does this design take?
 - Is it a necessary step?
- Are there certain properties common to group of houses?
 - e.g. beach house vs. suburban house
- What skills should architects possess?



>

Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team

 **VCU**
Computer Science
College of Engineering

4

Designing Houses vs. Designing Software

- Compare designing houses to software design
 - Do you think software design is necessary?
 - Who are software architects?

>

Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team

 **VCU**
Computer Science
College of Engineering

Software Design

- In the waterfall model, design generally occurs after requirements
 - In agile, design time is limited
- **Different design concerns at different abstraction levels** (e.g., class vs. module vs. entire system)
- Today we'll talk about the concept of software architecture
 - Highest level of design, closest to requirements



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



Software Architecture

The set of **principal design decisions** made during the development of the system and its subsequent evolution.

- Purposes:
 - Sustainability of the developed system
 - E.g., architect for performance, reliability, resiliency, availability, etc.
 - Effective project communication
 - Adequate basis for reuse



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



7

Software Architecture

Architecture is the fundamental **organization of a system** embodied in its **components**, their **relationships** to each other and to the environment, and the **principles guiding** its design and evolution. (IEEE 1471-2000 Standard)

- Software architecture encompasses the set of principal decisions about:
 - The **organization** of a software system
 - The selection of **structural** elements and their **interfaces** by which the system is composed together with their **behavior** as specified in the collaboration among those elements



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



8

Software Architecture

- Software architecture is **not only** concerned with structure and behavior, but also with usage, functionality, performance, reuse, comprehensibility, economic and technological constraints and tradeoffs



Lecture 19 - Software
Design and Architecture

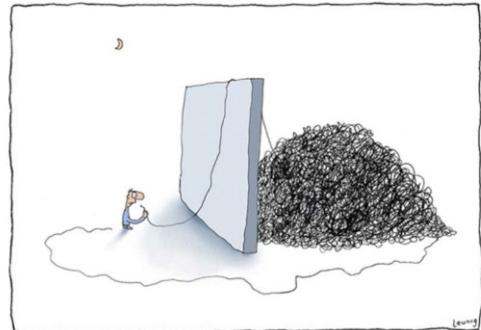
TDresearchteam
Technical Debt Research Team



Software Architecture

- ...is part of Design
 - **But not all design is architecture**
- ... includes structure and much more
 - Behavior, style, tools and language
- ... includes infrastructure and much more
- ... is part of system architecture

I'll just change this one thing...



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



Example System Architecture

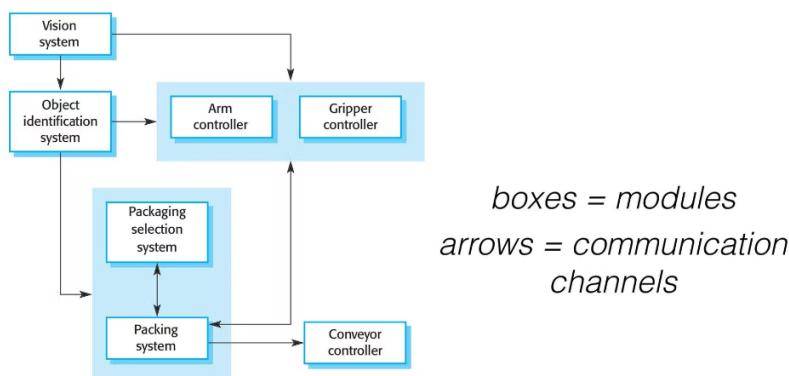


image credit: Sommerville, 10th edition



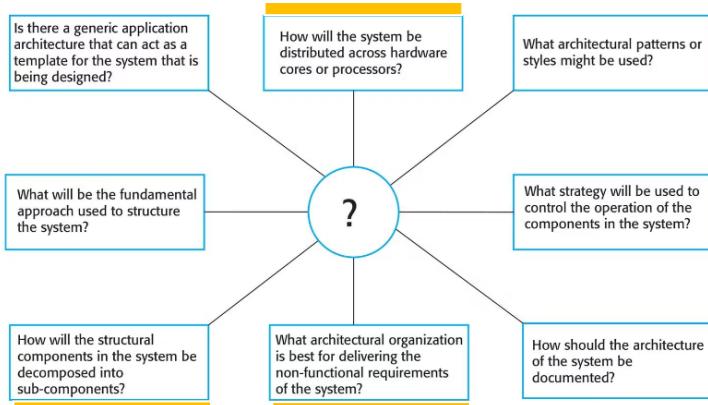
Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



11

Architectural Design Decision Matrix



Lecture 19 - Software Design and Architecture

TDresearchteam
Technical Debt Research Team



12

What happens to architecture during implementation and maintenance?

- Prescriptive – What we plan to build?
 - Usually, some sort of a model or document
- Descriptive – What we end up with?
 - As time goes on, prescriptive turns into descriptive
 - Usually, the structure of the code



Lecture 19 - Software Design and Architecture

TDresearchteam
Technical Debt Research Team



13

Architectural Degradation

- Architectural drift
 - Introduction of **new** principal design decisions into the descriptive architecture that were **not included** in the prescriptive architecture
- Architectural erosion
 - Introduction of new principal design decisions into the descriptive architecture that **violate** its prescriptive architecture

I'll just change this one thing...



Be careful, it can lead to **Technical Debt!**

>

Lecture 19 - Software Design and Architecture

TDresearchteam
Technical Debt Research Team

 **VCU**
Computer Science
College of Engineering

14

Architectural Design Patterns

- Patterns are means of representing, sharing and reusing knowledge
 - An architectural pattern is a description of good design practice, which has been tried and tested in different environments
 - The grand tool: **refined experience**
 - Learn from success and from failure
 - There is no virtue in reinventing the wheel

>

Lecture 19 - Software Design and Architecture

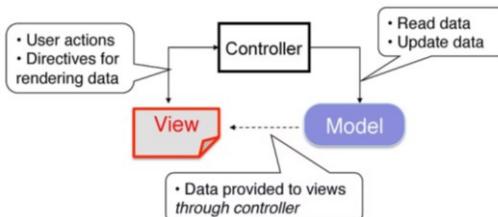
TDresearchteam
Technical Debt Research Team

 **VCU**
Computer Science
College of Engineering

Architectural Pattern: Model-View-Controller (MVC)

15

- **Goal:** separate organization of **data (model)** from **UI and presentation (view)** by introducing **controller**
 - Mediates user actions requesting access to data
 - Presents data for rendering by the view



Architectural Pattern: Model-View-Controller (MVC)

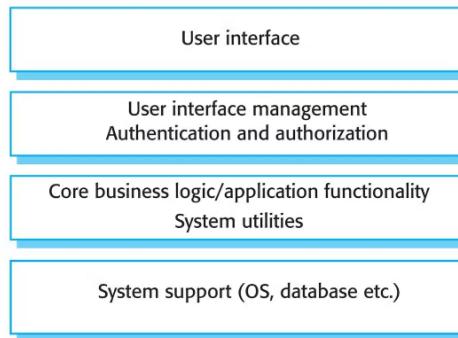
16

- Used in many places
 - Generally, in applications with a UI and data to show in that UI
 - E.g., web applications
- Benefits
 - Separation of concerns
- Disadvantages
 - Extra code



Architectural Pattern #2 Layered Design

18



19

Layered Architectural Pattern

- Also used in many places
 - E.g., operating systems
- Benefits
 - Separation of concerns
 - Abstract away complexity into each layer
- Disadvantages
 - Requires good interfaces
 - Performance can be an issue when many layers

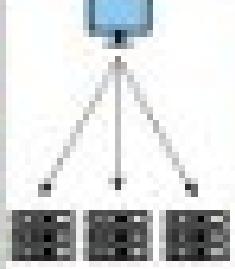


Distributed Layer

- Distributed – assumes that each layer can exist on a separate machine
 - Client-server
 - Two layers:
 - client = top layer
 - server = bottom layer
 - Client issues request to server, gets response
 - Usually assumes that the number of clients \geq number of servers
 - Peer-to-peer
 - Solves problem of “single point of failure/congestion” in client-server model
 - Adds layers of complexity in discovery, availability



| Top 5 Most-Used Architecture Patterns

Layered architecture	Event-driven architecture	Microkernel architecture	Microservices	Monolithic architecture
				

22

Perceived Tensions Architecture-Agility

Architecture

- Architecture = big up-front design
- Architecture = massive documentation
- Architects dictate from their ivory tower

Agility

- Low perceived or visible value of architecture
- Loss of rigor, focus on details
- Quality attributes not reducible to stories

Anticipation vs. Adaptation



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



23

Functions of a Software Architect



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



Functions of a Software Architect

Definition of the architecture

- Architecture definition
- Technology selection
- Architectural evaluation
- Management of non-functional requirements

Delivery of the architecture

- Ownership of the big Picture
- Leadership
- Coaching and mentoring
- Design, development, and testing
- Quality assurance



Lecture 19 - Software
Design and Architecture

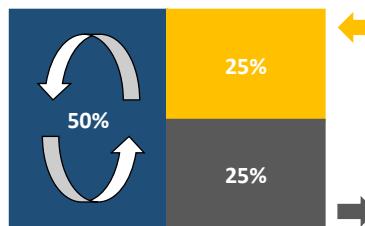
TDresearchteam
Technical Debt Research Team



What architects actually do?

Architecting:

- Design
- Validation
- Prototyping
- Documenting
- ...



Getting input:

- User, requirement
- Other architecture
- Technology

Providing information:

- Communicating architecture
- Assisting other stakeholders

Kruchten, P. (2008). What do software architects really do? *Journal of Systems and Software*, 81(12), 2413-2416.



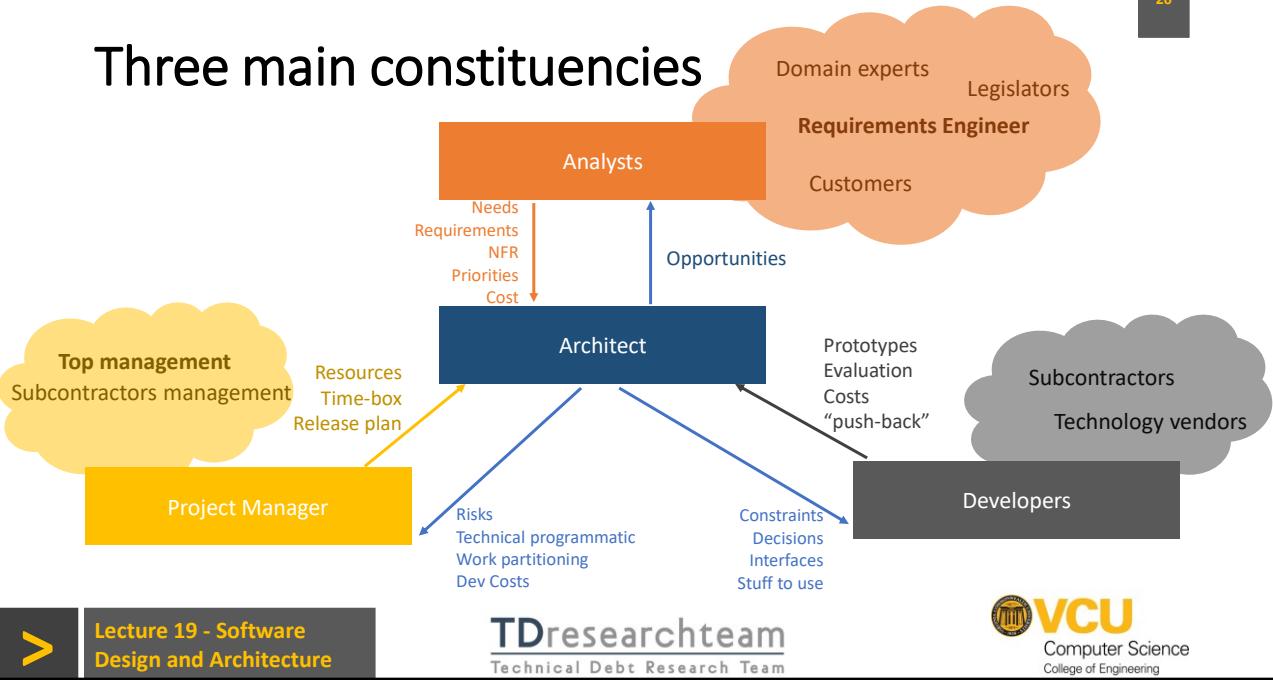
Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



26

Three main constituencies



Lecture 19 - Software
Design and Architecture



Summary

- All software-intensive systems have an architecture
- How much effort should you put into varies greatly
- Most of the time, the architecture is implicit
 - Choice of technology, platform
 - Still need to understand the architecture
- Novel systems
 - Much more effort in creating and validating an architecture
- Key drivers are mostly non-functional
 - Runtime: capacity, performance, scalability, security
 - Non runtime: evolvability, regulatory,...



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



28

Summary

Architecting is Making Decisions!

The life of a software architect is a long (and sometimes painful) succession of suboptimal decisions made partly in the dark.



Lecture 19 - Software Design and Architecture

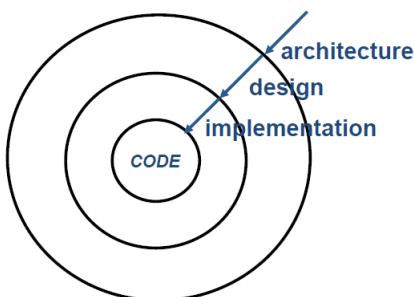
TDresearchteam
Technical Debt Research Team



29

Summary

- Architecture involves a set of strategic design decisions, rules or patterns that constraint design and code



Architecture decisions are the **most fundamental decisions** and changing them will have **significant ripple impacts**.



Lecture 19 - Software Design and Architecture

TDresearchteam
Technical Debt Research Team



Software Modeling and UML

Dr. Rodrigo Spínola



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



31

Why model software?

- Engineers have always modeled things they are planning to build
- Displays an engineered system at a particular level of abstraction
- Helps one think clearly about the system
- Crucial in communicating to others the structure of a system
- Makes working in a team possible



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



32

UML

due to its semantic

UML is a **graphical language** for **visualizing, specifying, constructing, and documenting** the artifacts of a software-intensive system.

- Consists of several different diagram types
- Can be used at different abstraction levels
 - From business process to individual language statements
- **It is a language, not a method or procedure!**



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



33

UML Diagrams

- UML 2 defines 13 diagram types, divided into two general sets:
 - Structural Modeling Diagrams:
 - Used to model “things” that make up a model – the classes, objects, interfaces, and components. In addition, they are used to model the relationships and dependencies between elements
 - Behavioral Modeling Diagrams:
 - Capture the varieties of interaction and instantaneous states within a model as it “executes” over time; tracking how the system will act in a real-world environment, and observing the effects of an operation or event, including its results



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



34

Structural UML

- **Class diagrams** define the basic building blocks of a model: the types, classes, and general materials used to construct a full model
- **Object diagrams** show how instances of structural elements are related and used at run-time
- **Component diagrams** are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well-defined interface
- **Deployment diagrams** show the physical disposition of significant artifacts within a real-world setting



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



35

Behavioral UML

- **Use case diagrams** are used to model user/system interactions. They define behavior, requirements and constraints in the form of scripts or scenarios
- **State machine diagrams** are essential to understand the instant-to-instant condition, or “run state” of a model when it executes
- **Communication diagrams** show the network, and sequence, of messages or communications between objects at run-time, during a collaboration instance
- **Sequence diagrams** are closely related to communication diagrams and show the sequence of message passed between objects using a vertical timeline
- **Timing diagrams** fuse sequence and state diagrams to provide a view of an object’s state over time, and messages which modify that state



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



UML

- Using UML models for software maintenance
 - Scaniello et al. "On the impact of UML analysis models on source-code comprehensibility and modifiability". ACM TOSEM 2014
 - *"We carried out a family of experiments to investigate whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code. [...] The results of both the individual experiments and meta-analysis indicate that UML models produced in the requirement analysis process influence neither the comprehensibility of source code nor its modifiability."*
 - **In other words, UML diagrams must be kept up to date to be useful**



Key Ideas in Software Modeling with UML

- Names are important
 - Give good names to classes in class diagram, use cases, etc.
- Provide only essential details
 - **Avoid over-modeling**
- Keep the model up to date
 - Easy for the model lose its usefulness if it's outdated



Tool Support for UML

- Many specialized tools exist
 - Some integrated in IDEs
 - Extensions for Eclipse or other IDEs
 - Visual studio has native support for class and sequence diagrams
 - Standalone (e.g. MS Visio, Visual Paradigm)



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team



Class is
over,
questions?

Software Design and Architecture

Dr. Rodrigo Spínola



Lecture 19 - Software
Design and Architecture

TDresearchteam
Technical Debt Research Team

