Software architecture - Course 2020-21

Study guide – Date: 20 March 2021

- https://arquisoft.github.io/course2021.html

Table of contents

Contenido

1.1 Presentation

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE01_Presentation.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.TE01_Presentacion.pdf

This document contains a list of contents for the Software architecture course (https://arquisoft.github.io/) with some annotations.

We goal of the document is to contain a guide of the contents given in the course which facilitates its search and that contains some references or annotations for further reading.

In each lesson we include a link to the slides in English and Spanish. The web page also contains the video recordings of the lecture classes.

The content that appears in blue color is content that has not been taught in the 2020-21 course.

1.2 Software Architecture – definitions

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE02_Definitions.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te02_Definiciones.pdf

- What is architecture?
- Vitruvius “De architectura”
  Firmitas (Durability)
  Utilitas (Utility)
  Venustas (Elegance/Beauty)
- What is software architecture?
  - Formal definition ISO/IEC/IEEE 42010:2011
  - Popular definition
  - Other definitions
- Buildings vs Software architecture
  - Similarities and differences
- Other disciplines which are similar
  - Civil engineering, mechanical engineering, aeronautics
- Other architectures
  - Business, enterprise, systems, information, data…
  - Common thing: Structure and vision
- Architecture vs design
- Benefits of software architecture
- Challenges of software architecture
- Laws of software architecture
  - 1st law: Everything is a trade-off
  - 2nd law: Why is more important than how
- Agile software architecture
- Architecture drivers (inputs)
  - Design objectives
    - Different systems
  - Functional requirements
  - Quality attributes
  - Constraints
  - Concerns
- Method vs creativity
- Types of systems
  - Greenfield systems in novel domains
  - Greenfield in a mature domains
  - Brownfield

1.3 Communicating software architecture

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE03_Documentation.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te03_Documentacion.pdf

More info: (Clements et al., 2010)

1.3.1.1 Architecture is more than code

1.3.1.2 Goal of documentation

1.3.1.3 Documentation requirements

1.3.1.4 Rules for good documentation

1.3.1.5 Problem vs solution space

1.3.1.6 Views and viewpoints

More info: (Clements et al., 2010)

1.3.1.7 Documenting views

1.3.1.8 Tools for diagrams

More info: (Brown, 2018)

1.3.1.8.1 Sketches

1.3.1.8.2 Drawing tools for diagrams

1.3.1.8.3 Text-based diagramming tools

1.3.1.8.4 Modeling tools

1.3.1.8.5 Reverse engineering the model

1.3.1.8.6 Architecture description languages (ADL)

1.3.1.9 Software architecture templates and methodologies

1.3.1.9.1 Kruchten 4+1

- Logical view
- Development view
- Process view
- Physical view
- Scenarios view

1.3.1.9.2 Views and beyond

More info: (Clements et al., 2010)

1.3.1.9.3 C4 model

1.3.1.9.4 Arc42

- Introduction and goals
- Constraints
- Context & scope
- Solution strategy
- Building block view
- Runtime view
- Deployment view
- Crosscutting concerns
- Architectural decisions
- Quality requirements
- Risks and technical debt
- Glossary

1.3.1.10 Architecturally evident coding style

More info: (Fairbanks, 2010)

1.4 Role of Software architect and stakeholders

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE04_Rol_SoftwareArchitect.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te04_RolArquitectoSoftware.pdf

1.4.1 Role of software architect

1.4.1.1 Expectations of an architect

- ○ Make architectural decisions
- ○ Continually analyse the architecture
- ○ Keep current with existing trends
- ○ Ensure compliance with existing decisions
- ○ Diverse exposure and experience
- ○ Have business domain knowledge
- ○ Possess interpersonal skills
- ○ Understand and navigate politics

1.4.1.2 Working in teams

More info: (Winters et al., 2020)

1.4.1.2.1 Social interactions

- Hiding and The Genius myth
- The Bus factor
- The 3 pillars of social interactions
  - ○ Respect
  - ○ Trust
  - ○ Humility

1.4.1.2.2 Architect personalities

More info: (Richards & Ford, 2020)

- Control freak vs Armchair architect
- 3 legs of software architect
  - ○ Skill
  - ○ Leadership
  - ○ Impact

1.4.1.3 Team topologies

More info: (Matthew Skelton, 2019)

1.4.1.3.1 Conway's law

1.4.1.3.2 Inverse Conway Maneuver

1.4.1.4 Team size

- ■ Process loss

- Pluralistic ignorance
- Diffusion of responsibility

1.4.1.5 Leveraging checklists

More info: (Gawande, 2011)

1.4.2 Stakeholders

1.4.2.1 Identifying stakeholders

1.4.2.2 Stakeholders’ expectations

1.4.2.3 Stakeholder map

1.4.2.4 Business goal statements

1.5 Quality attributes/architecture characteristics

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE05_QualityAttributes.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te05_AtributosCalidad.pdf

1.5.1 Types of requirements

1.5.1.1 Functional requirements

1.5.1.2 Quality attribute requirements

Also known as: Non-functional requirements

1.5.1.3 What is quality?

1.5.1.4 Quality attributes and trade-offs

1.5.2 Specifying quality attributes

1.5.2.1 Quality attribute scenarios

1.5.2.1.1 Components of a quality attribute scenario

Source, stimulus, artifact, response, response measure, environment

1.5.2.1.2 Types of quality attribute scenarios

1.5.2.1.3 Prioritizing quality attribute scenarios

1.5.2.2 Types of quality attributes

1.5.2.2.1 Operational

1.5.2.2.2 Structural

1.5.2.2.3 Cross-cutting

1.5.2.3 Finding quality attributes

1.5.2.3.1 Quality attribute workshops

1.5.2.3.2 Formal checklists: ISO 25010

1.5.2.3.3 Quality attribute tree

1.5.3 Measuring quality attributes

1.5.3.1.1 Operational measures

1.5.3.1.1.1 Absolute numbers vs Statistical models

1.5.3.1.1.2 Structural measures

Example: Cyclomatic complexity

1.5.3.1.1.3 Process measures

- Example: Code coverage

1.5.3.1.1.4 Governing quality attributes

  - Fitness functions

1.6 Achieving software architecture

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE06_AchievingSoftwareArchitecture.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te06_AlcanzandoArquitecturaSoftware.pdf

1.6.1 How much architecture?

More info: (Keeling, 2017)

- Total project = Development time + Architecture and risk reduction time + Rework time (fixing defects, rewrites, mistakes)
- Design sweet spot

1.6.2 Design concepts

More info: (Keeling, 2017)

- Design thinking
- 4 principles
  - Design for humans
    - All design is social in nature
  - Design for change
    - Architecture minimalism
    - Delay decisions until the least responsible moment
  - Design is redesign
    - Exploring patterns and past designs
  - Make ideas tangible
    - Communicate the architecture
- Adopting a design mindset
  - Understand the problem
  - Explore ideas
  - Make it real
  - Evaluate fit
- 4 questions of every design (More info: (Berkun, 2020)
  - What are you trying to improve?
  - Who are you trying to improve it for?
  - How do you ensure you are successful?
  - Who might be hurt by your work, now or in the future?
- MECE (Mutually Exclusive, Collectively Exhaustive) lists
  - More info: (Hewitt, 2018)
  - Wikipedia
- Logic tree, Ishikawa diagram or fishbone diagram, 5 whys
  - Find the root cause

1.6.3 ADD: Attribute driven design

- ADD 3.0
- More info: (Humberto Cervantes, 2016)

1.6.4 Risk based approach

1.6.5 Making decisions

- Record design decisions
  - Architecture decision records
    - Links
      - https://adr.github.io/
      - https://github.com/joelparkerhenderson/architecture_decision_record
  - Architecturally significant decisions
  - Delay decisions to the least responsible moment

1.6.6 Architectural issues

- Risks
  - Risk assessment matrix
  - 2 ways to reduce risks
    - Reduce the probability
    - Reduce the impact
- Unknowns
- Problems
- Technical debt
- Gaps in understanding
- Architectural erosion
- Contextual drift

1.6.7 Architectures evaluation

- Evaluation ATAM: Architecture Trade-off Analysis Method
- Cost Benefit Analysis Method

1.7 Architectural techniques and taxonomies

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE07_ArchitectureTechniques.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te07_TecnicasArquitectura.pdf

1.7.1 Definitions

1.7.1.1 Tactics

- More information: (Bass et al., 2012)

1.7.1.2 Architectural styles

- Are there pure styles?

1.7.1.3 Architectural pattern

1.7.1.4 Pattern vs style

1.7.1.5 Pattern languages and catalogs

1.7.1.6 Reuse vs create

1.7.1.6.1 Reference architectures

1.7.1.6.2 Externally developed components

- Technology stacks
- Products
- Application frameworks
- Platforms
- Libraries

1.8 Construction and maintenance

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE08_Construction.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te08_Construccion.pdf

1.8.1 Software: product vs service

1.8.2 Configuration management

1.8.3 Software construction

1.8.3.1 Overview of methodologies

1.8.3.1.1 Incremental piecemeal

1.8.3.1.2 Waterfall

1.8.3.1.3 V model

1.8.3.1.4 Big design up front

1.8.3.1.5 Iterative models

1.8.3.1.6 Overview of agile methodologies

- Adapt change
- Test-driven development
  - Types of testing:
    - Unit testing, integration, acceptance, performance/capacity, regression testing
    - Manual vs automated
  - Acceptance testing and Behavior driven development
  - FIRST principles
  - Test doubles
  - Environments: development, testing, production, staging
- Pair programming and code reviews
- Simplicity
- Refactoring
- Collective ownership of code
- Continuous integration
  - Best practices for CI
  - Tools
- On-place customer
- Continuous delivery
- Sustainable pace
- Clean code & code conventions

- Soma agile methods: Scrum, kanban, etc.

1.8.4 Configuration management

- Version control
    - Releases and versions
    - Version naming
        - Semantic versioning
    - Publishing releases
- Continuous delivery
    - Continuous deployment
- DevOps

1.8.5 Construction tools

- Construction languages
- Coding aspects
    - Naming conventions
- Testing
- Construction for reuse
- Construction with reuse
    - COTS
    - FOSS
- Tools
    - Text editors, IDEs, GUIs, QA tools
- Quality assurance tools
    - Tests, assertions, code reviews
    - Code analysis
        - Static vs dynamic code analysis
        - Debuggers
        - Profilers
        - Test-coverage tools
        - Program slicing

1.8.6 Control version systems

- Repository, baseline, delta, trunk/master, branch, tag
- Checkout, commit, merge
- Branching styles
- Centralized vs distributed version control systems
- Git
    - Definitions and components

1.8.7 Dependency management

- Dependency graph
- Cumulative component dependency (CCD)
- Cyclic dependencies
- Models for dependency management
- DSM Design-structure matrix

1.8.8 Build management

- Automation vs manual building
- Types of build automation
- Build automation tools
    - Make, ant, sbt, gradle, npm
    - Other tools: pants, bazel, buck

1.9 Modularity

Slides	
English	https://arquisoft.github.io/slides/course2021/EN.ASW.TE09_Modularity.pdf
Spanish	https://arquisoft.github.io/slides/course2021/ES.ASW.Te09_Modularidad.pdf

1.9.1 Big ball of mud

1.9.2 Modularity definitions

Module, interface, body

1.9.3 Modularity recommendations

1.9.3.1 SOLID principles

- Single responsibility principle
- Open/Closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency injection principle

1.9.3.2 High Cohesion

Definition of cohesion

1.9.3.2.1 Cohesion principles

More info: (Martin, 2017)

1.9.3.2.1.1 Reuse/release equivalent principle (REP)

1.9.3.2.1.2 Common reuse principle (CRP)

1.9.3.2.1.3 Common closure principle (CCP)

1.9.3.2.2 Cohesion metric

1.9.3.2.2.1 LCOM (Lack of Common Methods) metric

More info: (Richards & Ford, 2020)

1.9.3.3 Low Coupling

- Coupling principles
  - Acyclic dependencies principle (ADP)
  - Stable dependencies principle (SDP)
  - Stability metrics:
    - fan-in/fan-out, instability

- Stable abstractions principle (SAP)

1.9.3.4 Connascence

1.9.3.5 Robustness principle: Postel's law

1.9.3.6 Demeter's law

1.9.3.7 Fluid interfaces

1.9.3.8 Other modularity recommendations

- Facilitate external configuration
- Default implementation
- GRASP principles
- DRY
- YAGNI, KISS

1.9.4 Module systems

- Java: OSGi, Jigsaw
- .Net assemblies
- NodeJs: CommonJs

1.9.5 Modularity styles

1.9.5.1 Layers

- Layers ≠ modules
- Layers ≠ tiers
- Variants
    - Virtual machines
    - 3-layers, n-layers
- Sink-hole antipattern?

1.9.5.2 Aspect Oriented

- Crosscutting features and concerns
- Aspects and aspect weaving
- Applications
- Variants: Data-Context-Interaction

1.9.5.3 Domain based

- Technical vs domain partitioning
- Data models and domain models

1.9.5.3.1 Domain Driven Design

- Bounded context and ubiquitous language
- Entities

    - Value objects
    - Aggregates
    - Repositories
    - Factories
    - Services

1.9.5.3.2 Hexagonal architecture

- Data model
- Adapters

1.9.5.3.3 Clean architecture

- Entities
- Use cases
- Controllers/gateways/presenters

1.9.5.3.4 Data centered

- Semi-automatic generation

1.9.5.3.5 Naked objects

1.10 Runtime – Basic and Monolith styles

Slides	
English	
Spanish	

1.10.1 Data flow

1.10.1.1 Batch

1.10.1.2 Pipes and filters

- Challenges:
  - Backpressure
    - Resistance or force opposing the desired flow of data through the pipes
    - See: https://www.youtube.com/watch?v=I6eZ4ZyI1Zg

1.10.1.2.1 Pipes and filters with uniform interface

1.10.1.2.2 Variants

1.10.1.2.2.1 Flow based programming

- Flow based programming

1.10.1.2.2.2 Stream programming

- Example:
  - Graph DSL from Akka

1.10.1.3 Job organization

1.10.1.4 Master-slave

1.10.2 Interactive systems

1.10.2.1 MVC

Also known as Model-View-Controller
Variants of MVC:
MVU (Model-view-update) ?

1.10.2.2 PAC

Also known Presentation-Abstraction-Control

1.10.3 Repository

1.10.3.1 Shared data

1.10.3.2 Blackboard

- TupleSpace

1.10.3.3 Rule based

1.10.4 Invocation

1.10.4.1 Call-return

1.10.4.2 Client-server

- Variants
  - Stateless
  - Replicated server
  - With cache

1.10.5 Event driven architecture

1.10.5.1 Publish-subscribe

1.10.5.2 Actor models

1.10.5.3 CQRS: Command query responsibility segregation

1.10.5.4 Event sourcing

- Event store
- Event driver
- Snapshots

1.10.6 Adaptable systems

1.10.6.1 Plugins

1.10.6.2 Microkernel

1.10.6.3 Reflection

1.10.6.4 Interpreters and DSLs

DSLs = Domain specific languages

1.10.6.5 Mobile code

- Code on demand
- Remote evaluation
- Mobile agents

1.10.6.6 Cooperation systems

1.10.7 Distributed systems

1.11 Runtime - Distributed and Big Data systems

1.11.1 Distributed systems

1.11.1.1 Integration styles

1.11.1.1.1 File transfer

1.11.1.1.2 Shared database

1.11.1.1.3 Remote procedure call (RPC)

- 8 fallacies of distributed computing
- gRPC

1.11.1.1.4 Messaging

- Asynchronous communication
- Topologies
  - Hub & spoke
  - Bus
    - Enterprise service bus
    - Message Oriented Middleware (MOM)

1.11.1.2 Broker pattern

1.11.1.3 Peer-to-peer

Blockchain

1.11.1.4 Service Oriented Architectures (SOA)

1.11.1.4.1 Definitions: Endpoint, Contracts, Messages, Policies, Registries

*1.11.1.4.1.1 WS-**

- SOAP, WSDL, UDDI

1.11.1.4.1.2 REST

- Definitions: resources
- RESTful vs RPC hybrid
- REST as a composed style
  - Uniform interface
    - MIME types
    - HATEOAS
  - Service Based architecture

1.11.1.5 Microservices

- Microservices and Conway's law
- Variants:

    - Self-contained systems (SCS)

1.11.1.6 Serverless

- Function as a service
- Backend as a service
- Rich clients
    - Single Page Applications

1.11.2 Scalable and Big data

1.11.2.1 Space architecture

Wikipedia: https://en.wikipedia.org/wiki/Space-based_architecture
TupleSpace

1.11.2.2 MapReduce

1.11.2.3 Lambda architecture

1.11.2.4 Kappa architecture

1.12 Allocation & deployment

Slides	
English	???
Spanish	

1.12.1 Deployment view

More info: (Clements et al., 2010)

1.12.2 Packaging, distribution and deployment

1.12.2.1 Packaging

1.12.2.2 Distribution channels

- Physical distribution, web based, application markets

1.12.2.3 Software computation options

- Data center (On premises)
- Cloud computing
- Edge computing
- Fog computing

1.12.2.4 Execution environments

- Physical hosts
- Virtual machines
- Containers

1.12.2.5 Continuous delivery and continuous deployment

More info: (Jez Humble, 2010)

1.12.2.5.1 Deployment pipeline

1.12.2.5.2 Infrastructure as code

- Best practices: 12 factor

1.12.2.5.3 Testing and continuous delivery

1.12.2.5.3.1 Feature toggles

1.12.2.5.3.2 Canary releases

1.12.2.5.3.3 A/B testing

1.12.3 Software in production

More info: (Nygard, 2018)

1.12.3.1 Quality attributes in production

- Availability, reliability, observability

1.12.3.2 Capacity planning

- Load testing
- Load balancing
    - Redundancy
    - Failover

1.12.3.3 Logging & monitoring

- Health checks

1.12.3.4 Incidents & post-mortem analysis

1.12.3.4.1 Root cause analysis

1.12.3.4.2 Preventive measures

1.12.3.5 Chaos engineering

1.13 Software architecture and enterprise environment

Slides	
English	???
Spanish	

1.13.1 Software architect at enterprises

- Software architect elevator
  - More info: (Hohpe, 2020)
- Role of software architect
  - Architectural drivers
  - Designing software
  - Technical risks
  - Architecture evolution
  - Coding
  - Quality assurance
- Other architects
  - Enterprise architect
  - Solutions architect
  - Business architect
- Enterprise architecture approaches
  - Zachman framework
  - TOGAF
- Software architecture and trends
  - Gartner hype cycles
  - Soft skills

1.13.2 Enterprise software

- Enterprise software taxonomy
  - CRM (Customer relationship management)
  - SCM (Supply chain management)
  - WMS (Warehouse management system)
  - PLM (Product lifecycle management)
  - B2B (Business to business)
  - BPM (Business Process Management)
  - ECM (Enterprise Content Management)
  - ERP (Enterprise Resource Planning)
  - EAI (Enterprise Application Integration

1.13.3 Software product lines

- Product lines
- Automatic system generation

1.13.4 Software and enterprise services

- From product to service

- Service terminology
    - Service level indicators
    - Service level objective
    - Service level agreement
- Service governance
    - Release management and deployment
        - Reliability
        - API management
        - Monitoring
    - Production support
        - Incidence response
        - On-call rotations
    - Cost model
    - Client onboarding
- Evolutionary architectures
    - Evolvability: Incremental, guided change
    - Fitness functions

2 References

Bass, L., Kazman, R. & Clements, P., 2012. *Software Architecture in Practice*. Addison Wesley.
Berkun, S., 2020. *How Design Makes the World*. LIGHTNING SOURCE INC.
Brown, S., 2018. *Software architecture for developers*. Leanpub. Available at: http://leanpub.com/visualising-software-architecture.
Clements, P., Bachmann, F. & Bass, L., 2010. *Documenting Software Architectures: Views and Beyond*. ADDISON WESLEY PUB CO INC.
Fairbanks, G., 2010. *Just Enough Software Architecture: A Risk-Driven Approach*. MARSHALL & BRAINERD.
Gawande, A., 2011. *The Checklist Manifesto*. Profile Books.
Hewitt, E., 2018. *Technology Strategy Patterns*. O'Reilly Media, Inc, USA. Available at: https://www.ebook.de/de/product/33775257/eben_hewitt_technology_strategy_patterns.html.
Hohpe, G., 2020. *The Software Architect Elevator*. O'Reilly Media, Inc, USA.
Humberto Cervantes, R.K., 2016. *Designing Software Architectures: A Practical Approach*. ADDISON WESLEY PUB CO INC.
Jez Humble, D.F., 2010. *Continuous Delivery*. Addison Wesley.
Keeling, M., 2017. *Design It!* O'Reilly UK Ltd.
Martin, R.C., 2017. *Clean Architecture*. Prentice Hall.
Matthew Skelton, M.P., 2019. *Team Topologies*. It Revolution Press.
Nygard, M., 2018. *Release It!* O'Reilly UK Ltd.
Richards, M. & Ford, N., 2020. *Fundamentals of Software Architecture*. O'Reilly UK Ltd.
Winters, T., Manshreck, T. & Wright, H., 2020. *Software Engineering at Google: Lessons Learned from Programming Over Time*. O'Reilly UK Ltd.

3 Calendar 2020/21

- Class 1, 3-Feb, 2h:
  - Presentation
  - Basic definitions
  - Seminar 0 (5-Feb)
- Class 2, 10-Feb,
  - 1h Conference Empathy
  - 1h Documenting Software architecture
- Class 3, 17-Feb, 2h: Quality attributes
  - Seminar 1 (19 Feb)
    - S1. Architecture decision records
    - S2. Agile architecture
  - Week 22-26 Feb – Deliverable lab assignment 1 (documentation)
- Class 4, 24-Feb, 2h: Achieving software arch. Building (part 1)
- Class 5, 3-March, 2h: Building (part 2), Modularity
  - Seminar 2 (5 Mar)
    - S3. Continuous delivery
    - S4. Branching models
- Class 6, 10-Mar, 2h: Runtime
  - Deliverable lab assignment 2 (1st prototype)

- Class 7, 17 March, 2h: Runtime
    - Seminari 3 (19 Mar)
        - S5. Clean architecture
        - S6. Event sourcing
- Class 8, 24 March, 2h: Integration
    - Easter break.
- Class 9, 7-Apr, 2h: Allocation & deployment
    - Seminario 4: 9 April
        - S7. Fallacies of Microservices
        - S8. Circuit breaker
    - Devlierable lab 3 (documentation/prototype)
- Class 10, 14-Apr, 2h: <<Conference Jorge Manrubia >>
- Class 11, 21-Apr, 1h: Enterprise
    - Seminar 5: 23 Aprill
        - S09. Chaos engineering
        - S10. Serverless
    - Deliverable final lab assignment
    - Seminar 6: 7 May
        - Questions about exam

4 Index