

Software architecture

Study guide

Author: Jose Emilio Labra Gayo

Course: 2020-21

Date: 25 May 2021

Web page: <https://arquisoft.github.io/course2021.html>

1.1	Presentation	3
1.2	Software Architecture – definitions.....	4
1.3	Communicating software architecture.....	5
1.4	Role of Software architect and stakeholders	7
1.4.1	Role of software architect.....	7
1.4.2	Stakeholders	8
1.5	Quality attributes/architecture characteristics	9
1.5.1	Types of requirements.....	9
1.5.2	Specifying quality attributes	9
1.5.3	Measuring quality attributes	10
1.6	Achieving software architecture	11
1.6.1	How much architecture?	11
1.6.2	Design concepts.....	11
1.6.3	ADD: Attribute driven design.....	11
1.6.4	Risk based approach.....	12
1.6.5	Making decisions.....	12
1.6.6	Architectural issues	12
1.6.7	Architectures evaluation	12
1.7	Architectural techniques and taxonomies	13
1.7.1	Definitions.....	13
1.8	Construction and maintenance	14
1.8.1	Software: product vs service	14
1.8.2	Configuration management.....	14
1.8.3	Software construction.....	14
1.8.4	Configuration management.....	14
1.8.5	Construction tools.....	15
1.8.6	Control version systems.....	15

1.8.7	Dependency management	15
1.8.8	Build management	15
1.9	Modularity.....	16
1.9.1	Big ball of mud.....	16
1.9.2	Modularity definitions.....	16
1.9.3	Modularity recommendations	16
1.9.4	Module systems	17
1.9.5	Modularity styles.....	17
1.10	Runtime – Basic and Monolith styles	19
1.10.1	Data flow	19
1.10.2	Interactive systems	19
1.10.3	Repository	20
1.10.4	Invocation.....	20
1.10.5	Event driven architecture	20
1.10.6	Adaptable systems.....	20
1.11	Runtime - Distributed and Big Data systems	22
1.11.1	Distributed systems.....	22
1.11.2	Scalable and Big data	24
1.12	Allocation & deployment.....	25
1.12.1	Packaging, distribution and deployment	25
1.12.2	Deployment	25
1.12.3	Deployment pipeline.....	26
1.12.4	Software in production	27
1.13	Software architecture and enterprise environment	30
1.13.1	Software architect at enterprises	30
1.13.2	Enterprise software	30
1.13.3	Software product lines	31
1.13.4	Software and enterprise services.....	31
2	References	32
3	Calendar 2020/21	32
4	Index.....	34

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.

The goal of this document is to contain a guide of the contents given in the course which can facilitate its search. It also 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.

Any errors or suggestions can be done at: <https://github.com/Arquisoft/faq/issues>

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
-  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 \neq modules
- Layers \neq 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 https://arquisoft.github.io/slides/course2021/EN.ASW.TE10_Monolithic.pdf

Spanish https://arquisoft.github.io/slides/course2021/ES.ASW.Te10_Comportamiento_Monolito.pdf

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=l6eZ4Zyl1Zg>

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

1.11 Runtime - Distributed and Big Data systems

Slides

English https://arquisoft.github.io/slides/course2021/ES.ASW.Te11_DistribuidosBigData.pdf

Spanish https://arquisoft.github.io/slides/course2021/EN.ASW.TE11_DistributedBigDataSystems.pdf

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

1.11.1.1.2.1 Variants

1.11.1.1.2.1.1 Data warehousing

1.11.1.1.2.1.2 ETL

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
 - ESB - Enterprise Service Bus
 - Message Oriented Middleware (MOM)

1.11.1.2 Broker pattern

- Elements: Broker, Client, Server, Bridge
- Applications: CORBA, Android

1.11.1.3 Peer-to-peer

- Variants: Super-peers
- Applications:
 -  Blockchain

1.11.1.4 Service Oriented Architectures (SOA)

- Elements:
 - Provider/consumer
 - 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
- REST as a composed style

1.11.1.4.2 Service based architecture

1.11.1.5 Microservices

- Microservices and Conway's law
- Features/advantages
 - Heterogeneity
 - Resilience
 - Scalability
 - Deployability
 - Organizational alignment
 - Inverse Conway Maneuver
 - Decentralized data management
 - Optimizing for replaceability
- Challenges
 - Structural decay
- 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

- Algorithm description: mapper, reducer, merge & sort
- File system: HDFS: data node, name node

1.11.2.3 Lambda architecture

- Elements: Batch layer, speed layer, serving layer

1.11.2.4 Kappa architecture

- Elements: append-only immutable log, service databases

1.12 Allocation & deployment

Slides

English https://arquisoft.github.io/slides/course2021/EN.ASW.TE12_Allocation.pdf

Spanish https://arquisoft.github.io/slides/course2021/ES.ASW.Te12_Disposicion.pdf

1.12.1 Packaging, distribution and deployment

1.12.1.1 Packaging & distribution

- Parts of a software package:
 - o compiled code
 - o configuration files
 - o libraries & dependencies
 - o user manuals and docs
 - o installation scripts
- The problem of shipping software
 - o Dependencies
 - o Developer's environment vs production environment
- Distribution channels
 - o Physical distribution
 - o Web based distribution
 - o Application markets

1.12.2 Deployment

1.12.2.1 Deployment view

1.12.2.1.1 UML deployment diagrams

More info: (Clements et al., 2010)

1.12.2.2 Software computing options

1.12.2.2.1 On premises

1.12.2.2.2 Cloud computing

Software as a Service

Pet vs cattle metaphor

1.12.2.2.3 Edge computing

1.12.2.2.4 Fog computing

1.12.2.3 Execution environments

1.12.2.3.1 Physical hosts

Lots of possibilities

- Commodity computer
- Super computer
- Server-farms

- End-user devices

1.12.2.3.2 System virtual machines

1.12.2.3.3 Containers and docker

Operating system level virtualization

Elements:

- Container descriptions (images)
- Containers
- Platform that runs containers
- Container registries

1.12.2.3.3.1 Docker architecture

- Docker client
- Docker server (DOCKER_HOST)
- Docker registry
- Containers
- Images
 - o Layered architecture
- Docker daemon

1.12.2.3.3.2 Mutable vs immutable infrastructure

1.12.2.3.3.3 Container management

1.12.2.3.3.3.1 Docker-compose

1.12.2.3.3.3.2 Container orchestration

- Kubernetes
- Docker swarm

1.12.3 Deployment pipeline

1.12.3.1 Manual vs Automatic deployment

1.12.3.2 Continuous deployment

More info: (Jez Humble, 2010)

1.12.3.2.1 Patterns

- Patterns
 - o Infrastructure as code
 - o Keep everything in version control
 - o Align development and operations
- Best practices: 12 factor
 - o Codebase
 - o Dependencies
 - o Config
 - o Backing services
 - o Build, release, run
 - o Processes
 - o Port binding

- Concurrency
- Disposability
- Dev/prod parity
- Logs
- Admin processes

1.12.4 Software in production

More info: (Nygard, 2018)

1.12.4.1 Quality attributes in production

- Configurability, observability, availability, stability, reliability

1.12.4.2 Configurability

- Configurable properties
- Config files
- Sensitive information

1.12.4.3 Observability

1.12.4.3.1 Logging

Log locations
Logging levels

1.12.4.3.2 Monitoring

1.12.4.3.3 Profiling

1.12.4.3.4 Data in production

- High availability and data replication
- Database schemas and change requests
- Data purging
- Sensible data in production

1.12.4.3.5 System problems

- Fault
- Error
- Failure
- Law of large systems

1.12.4.4 In-production patterns

1.12.4.4.1 Load balancing

1.12.4.4.2 Timeouts

1.12.4.4.3 Circuit breaker

1.12.4.4.4 Bulkheads

1.12.4.4.5 Steady state

1.12.4.4.6 Fail fast

1.12.4.4.7 Let it crash

1.12.4.4.8 Handshaking

1.12.4.4.9 Create backpressure

1.12.4.4.10Governor

1.12.4.4.11Test harnesses

1.12.4.4.12Chaos engineering

1.12.4.5 In-production antipatterns

1.12.4.5.1 Integration points

1.12.4.5.2 Chain reactions

1.12.4.5.3 Cascading failures

1.12.4.5.4 Users

1.12.4.5.5 Blocked threads

1.12.4.5.6 Self-denial attacks

1.12.4.5.7 Scaling effects

1.12.4.5.8 Unbalanced capacities

1.12.4.5.9 Dogpile

1.12.4.5.10 Force multiplier

1.12.4.5.11 Slow responses

1.12.4.5.12 Unbounded result sets

1.12.4.6 Testing in production

Progressive delivery

1.12.4.6.1 Canary releases

1.12.4.6.2 Feature toggles

1.12.4.6.3 Types of tests

1.12.4.6.3.1 A/B testing

1.12.4.6.3.2 Multi-armed bandits

1.12.4.7 Load and stress testing

- Load testing
- Stress testing

1.12.4.8 Incidents & post-mortem analysis

1.12.4.8.1 Root cause analysis

1.12.4.8.2 Preventive measures

1.13 Software architecture and enterprise environment

Slides

English https://arquisoft.github.io/slides/course2021/EN.ASW.TE13_EnterpriseSoftware.pdf

Spanish https://arquisoft.github.io/slides/course2021/ES.ASW.Te13_Empresarial.pdf

1.13.1 Software architect at enterprises

- Role of software architect
 - Architectural drivers
 - Designing software
 - Technical risks
 - Architecture evolution
 - Coding
 - Quality assurance
- Expectations of software architect
- Understand and navigate positics
 - Software architect elevator
 - More info: (Hohpe, 2020)
- Some types of companies
 - Product-based companies in software
 - Product-based companies in other domains
 - Consulting or service-based companies
 - Startups and entrepreneurs
- Other architects
 - Enterprise architect
 - Solutions architect
 - Business architect
 - Software architect
 - Others: data architect, application architect, technology architect,...
- Enterprise architecture
 - 2 main 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

- Trend toward services
- 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
- Software evolution
 - Lehman's laws on software evolution
 - Continuing change
 - Increasing complexity
 - Other laws
- Software refactoring
- Legacy projects
- Behavioral code analysis
 - System hotspots
- 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 - Monolith
 - **Deliverable lab assignment 2 (1st prototype)**
- Class 7, 17 March, 2h: Runtime – Monolith
 - Seminar 3 (19 Mar)
 - S5. Clean architecture
 - S6. Event sourcing
- Class 8, 24 March, 2h: Distributed and big data
 - Easter break.

-
- Class 9, 7-Apr, 2h: Allocation & deployment
 - Seminar 4: 9 April
 - S7. Fallacies of Microservices
 - S8. Circuit breaker
 - **Deliverable lab 3 (documentation/prototype)**
 - Class 10, 14-Apr, 2h: <<Conference Jorge Manrubia: Fighting the merchants of complexity>>
 - Class 11, 21-Apr, 1h: Enterprise
 - Seminar 5: 23 April
 - S09. Chaos engineering
 - S10. Serverless
 - **Deliverable final lab assignment**
 - Seminar 6: 7 May
 - Questions about exam

4 Index

- .Net assemblies, 24
- 12 factor, 34
- 3 pillars of social interactions, 13
- A/B testing, 36
- Actor models, 27
- Acyclic dependencies principle, 23
- Adaptable systems, 27
- Adapters, 25
- ADL. Véase
 - Architecture description languages
- Aggregates, 25
- Agile software architecture, 10
- Allocation, 32
- API management, 38
- Arc42, 12
- Architect personalities, 13
- architectural decisions, 13
- Architectural decisions, 12
- Architectural erosion, 18
- Architectural pattern, 19
- Architectural styles, 19
- Architecturally evident coding style, 12
- architecture characteristics, 15
- Architecture decision records, 18
- Architecture description languages, 11
- Architecture drivers, 10
- Architecture Trade-off Analysis Method, 18
- Armchair architect, 13
- artifact, 15
- Aspect Oriented, 24
- aspect weaving, 24
- Asynchronous communication, 29
- ATAM. Véase Architecture Trade-off Analysis Method
- Automatic system generation, 38
- Automation, 22
- B2B. Véase Business to business
- Backend as a service, 31
- Batch, 26
- Beauty, 10
- Behavioral code analysis, 38
- Benefits of software architecture, 10
- Big ball of mud, 23
- Big Data systems, 29
- Blackboard, 27
- Bounded context, 24
- BPM. Véase Business Process Management
- Broker, 29
- Brownfield system, 10
- Building block view, 12
- Bus, 29
- Business architect, 37
- Business goal statements, 14
- Business Process Management, 37
- Business to business, 37
- C4 model, 12
- Call-return, 27
- Canary releases, 36
- CCD. Véase
 - Cumulative component dependency
- CCP. Véase
 - Common closure principle
- Challenges of software architecture, 10
- checklists, 14
- Clean architecture, 25
- Client-server, 27
- Client-server with cache, 27
- Code coverage, 16
- Code on demand, 28
- Command query responsibility segregation, 27
- Common closure principle, 23
- Common reuse principle, 23
- CommonJs, 24
- Communicating software architecture, 11
- Concerns, 10
- Configuration management, 21
- Connascence, 24
- Constraints, 10
- Contextual drift, 18
- Continuing change, 38
- continuous delivery, 36
- Continuous integration, 20
- Contracts, 30
- Control freak, 13
- Controllers, 25
- Conway's law, 30
- Conway's law, 13
- Cooperation systems, 28
- Cost Benefit Analysis Method, 18
- Cost model, 38
- COTS, 21
- Coupling, 23
- CQRS. Véase Command query responsibility segregation
- creativity, 10
- CRM. Véase
 - Customer-relationship management
- Crosscutting concerns, 12, 24
- Cross-cutting quality attribute, 16
- CRP. Véase
 - Common reuse principle
- Cumulative component dependency, 21
- Customer relationship management, 37
- Cyclic dependencies, 21
- Cyclomatic complexity, 16

- Data centered, 25
- Data flow, 26
- Data model, 24, 25
- Data warehousing, 29
- Data-Context-Interaction, 24
- De architectura, 10
- Definition of software architecture, 10
- Demeter's law, 24
- Dependency graph, 21
- dependency management, 21
- Deployment, 32
- Deployment view, 12
- design, 10
- Design objectives, 10
- Development view, 11
- DevOps, 21
- Diffusion of responsibility, 14
- Documentation requirements, 11
- Documenting views, 11
- Domain Driven Design, 24
- Domain model, 24
- Domain partitioning, 24
- Domain specific languages, 27
- Drawing tools for diagrams, 11
- DRY, 24
- DSLs. Véase Domain Specific Languages
- [DSM Design-structure matrix](#), 21
- Durability, 10
- EAI. Véase Enterprise Application Integration
- ECM. Véase Enterprise Content Management
- Elegance, 10
- Endpoint, 30
- enterprise, 37
- Enterprise Application Integration, 38
- Enterprise architect, 37
- Enterprise Content Management, 37
- Enterprise Resource Planning, 37
- Enterprise service bus, 29
- Entities, 24, 25
- environment, 15
- ERP. Véase Enterprise Resource Planning
- ESB, 29
- ETL
 - Extract-Transform-Load, 29
- Event driver, 27
- Event sourcing, 27
- Event store, 27
- Event-based, 27
- Evolutionary architectures, 38
- Factories, 25
- fallacies of distributed computing, 29
- fan-in, 23
- fan-out, 23
- Feature toggles, 36
- File transfer, 29
- Firmitas, 10
- Fitness functions, 16, 38
- Fluid interfaces, 24
- FOSS, 21
- Function as a service, 31
- Functional requirements, 10, 15
- Gaps in understanding, 18
- Glossary, 12
- GRASP principles, 24
- Greenfield systems, 10
- gRPC, 29
- HATEOAS, 30
- Hexagonal architecture, 25
- Hiding, 13
- Hub and spoke topology, 29
- Humility, 13
- Increasing complexity, 38
- Infrastructure as code, 33
- instability, 23
- Integration styles, 29
- Interactive systems, 26
- Interpreters, 27
- Inverse Conway Maneuver, 13
- Invocation, 27
- ISO/IEC/IEEE 42010:2011, 10
- Jigsaw, 24
- Kappa architecture, 31
- KISS, 24
- Kruchten 4+1, 11
- Lack of Common Methods metric, 23
- Lambda architecture, 31
- Laws of software architecture, 10
- Layers, 24
- LCOM. Véase Lack of Common Methods
- Legacy projects, 38
- Lehman's laws on software evolution, 38
- Load testing, 36
- Logical view, 11
- manual building, 22
- MapReduce, 31
- Master-slave, 26
- Measuring quality attributes, 16
- Message Oriented Middleware, 29
- Messaging, 29
- Messaging topologies, 29
- Microkernel, 27
- Microservices, 30
- MIME types, 30
- Mobile agents, 28
- Mobile code, 28
- Modeling tools, 11
- Model-View-Controller, 26
- MOM. Véase Message Oriented Middleware

- Monitoring, 38
- Multi-armed bandits, 36
- MVC. *Véase* Model View Controller
- Naked objects, 25
- Non-functional requirements, 15
- On-place customer, 20
- Operational measures, 16
- Operational quality attribute, 16
- OSGi, 24
- PAC. *Véase* Presentation-Abstraction-Control
- Pattern catalogs, 19
- Pattern languages, 19
- Physical view, 11
- Pipes and filters, 26
- Pipes and filters with uniform interface, 26
- PLM. *Véase* Product lifecycle management
- Plugins, 27
- Pluralistic ignorance, 14
- Policies, 30
- Postel's law, 24
- post-mortem analysis, 36
- Presentation-Abstraction-Control, 26
- Preventive measures, 36
- Problems, 18
- Process loss, 13
- Process view, 11
- Product lifecycle management, 37
- Product lines. *Véase* Software product lines
- Progressive delivery, 36
- Publish-subscribe, 27
- quality, 15
- Quality attribute scenarios, 15
- Quality attribute tree, 16
- Quality attribute workshops, 16
- Quality attributes, 10, 15
- Quality requirements, 12
- Registries, 30
- Release management, 38
- Reliability, 38
- Remote evaluation, 28
- Remote procedure call, 29
- REP. *Véase* • Reuse/release equivalent principle
- Replicated server, 27
- Repositories, 25
- Repository, 27
- Respect, 13
- response, 15
- response measure, 15
- REST, 30
- RESTful, 30
- Reuse/release equivalent principle, 23
- Reverse engineering, 11
- Rich clients, 31
- Risks, 12, 18
- Robustness principle, 24
- Rol of Software architect, 13
- Root cause analysis, 36
- RPC. *Véase* Remote procedure call
- RPC hybrid, 30
- Rule based, 27
- Runtime view, 12
- Scenarios view, 11
- SCM. *Véase* Supply chain management
- Self-contained systems, 30
- Serverless, 31
- Service governance, 38
- Service level agreement, 38
- Service level indicators, 38
- Service level objective, 38
- Service Oriented Architectures, 30
- Services, 25
- Shared data, 27
- Shared database, 29
- Single Page Applications, 31
- Sink-hole antipattern, 24
- Sketches, 11
- Snapshots, 27
- SOA. *Véase* Service Oriented Architectures
- SOAP, 30
- Software architecture, 10
- Software architecture documentation, 11
- Software architecture templates, 11
- Software evolution, 38
- Software product lines, 38
- Software refactoring, 38
- Solution strategy, 12
- Solutions architect, 37
- Space architecture, 31
- Stable abstractions principle, 24
- Stable dependencies principle, 23
- Stakeholder map, 14
- stakeholders, 13
- Stakeholders, 14
- Stateless client-server, 27
- Statistical models, 16
- stimulus, 15
- Stress testing, 36
- Structural quality attribute, 16
- Supply chain management, 37
- System hotspots, 38
- Tactics, 19
- Team size, 13
- Team topologies, 13
- technical debt, 12
- Technical debt, 18
- Technical partitioning, 24
- Text-based diagramming tools, 11
- The Bus factor, 13

The Genius myth, 13	Venustas, 10
TOGAF, 37	viewpoints, 11
Traditional buildings architecture, 10	Views, 11
Trust, 13	Views and beyond, 12
Ubiquitous language, 24	Virtual machines, 24
UDDI, 30	Vitruvius, 10
Unknowns, 18	Warehouse management system, 37
Use cases, 25	WMS, 37
Utilitas, 10	WS-*, 30
Utility, 10	WSDL, 30
V model, 20	YAGNI, 24
Value objects, 25	Zachman framework, 37