University of Oviedo

Universidad de Oviedo

School of Computer Science

Achieving
software architecture

SOFTWARE
ARCHITECTURE

School of Computer Science

Course 2022/23

Jose E. Labra Gayo

Achieving software architecture

Methodologies

ADD

Risk driven architecture

Making decisions

Architectural issues

Risks, unknowns, problems, gaps, drift
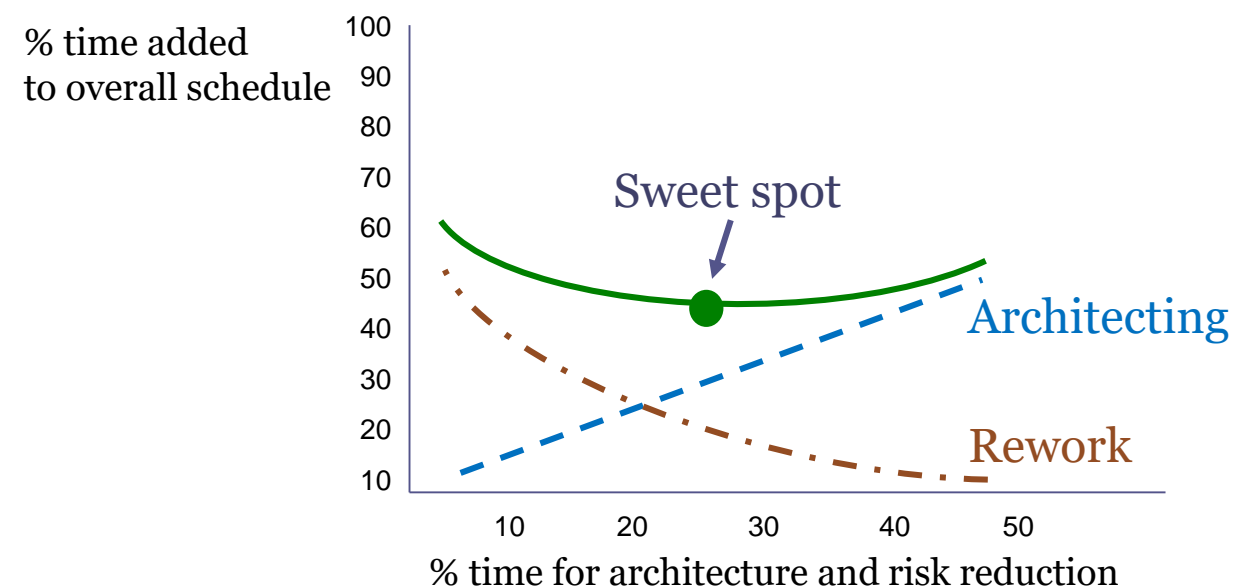
Architecture evaluation

University of Oviedo

How much architecture?

Sweet spot between too much architecture and too much rework

$$Total\ project\ time = \begin{cases} Development\ time\ + \\ Architecture\ time\ + \\ Rework\ time \end{cases}$$

% time added
to overall schedule

School of Computer Science

Attribute driven design

ADD: Attribute-driven design

Defines a software architecture based on QAs

Recursive decomposition process

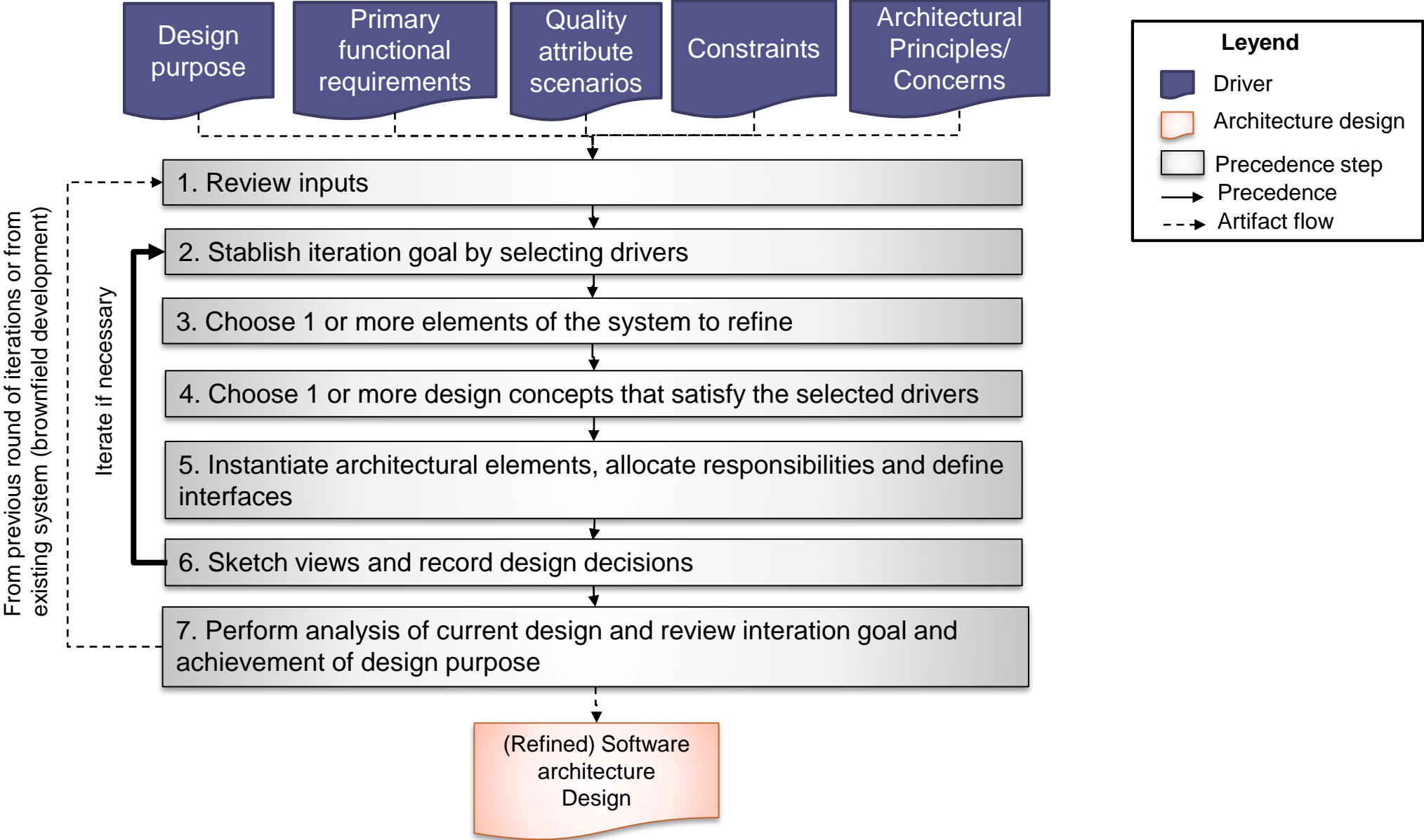At each stage tactics and patterns are chosen to satisfy a set of QA scenarios

Input

- QA requirements
- Constraints
- Architectural significant functional requirements

Output

- First levels of module decomposition
- Various views of the system as appropriate
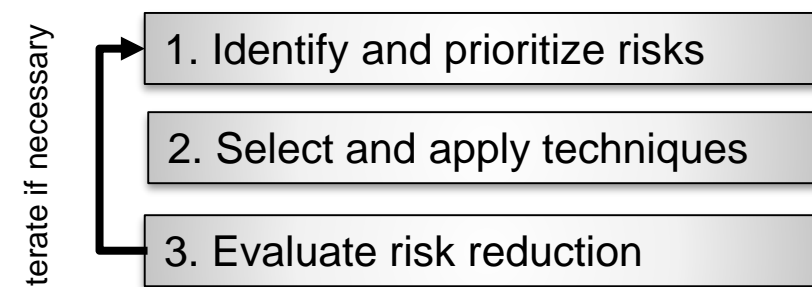- Set of elements with assigned functionalities and the interactions among the elements

University of Oviedo

School of Computer Science

ADD 3.0

Design purpose

Primary functional requirements

Quality attribute scenarios

Constraints

Architectural Principles/ Concerns

Leyend

Driver

Architecture design

Precedence step

→ Precedence

--→ Artifact flow

1. Review inputs

2. Stablish iteration goal by selecting drivers

3. Choose 1 or more elements of the system to refine

4. Choose 1 or more design concepts that satisfy the selected drivers

5. Instantiate architectural elements, allocate responsibilities and define interfaces

6. Sketch views and record design decisions

7. Perform analysis of current design and review interation goal and achievement of design purpose

From previous round of iterations or from existing system (brownfield development)

Iterate if necessary

(Refined) Software architecture Design

Risk based approach

Goal = Just enough software architecture

Avoid Big design up front

Integrate software architecture and agile methods

Iterate if necessary

| 1. Identify and prioritize risks |
| 2. Select and apply techniques |
| 3. Evaluate risk reduction |

More info: https://www.methodsandtools.com/archive/agilesoftwarearchitecture.php

Architectural decisions

Significant decisions from the point of view of software architecture

Usually, decisions that affect

    Quality attributes

    Structure

    Dependencies

    Interfaces

    Construction techniques

Architecture decisions anti-patterns

Analysis paralysis

Decisions not taken

Fear of taking decisions

Advise: Evaluate decisions with team (and expect changes)

Wait until last responsible moment (but no longer)

Trapped in time (groundhog day)

People don't know why a decision has taken

It keeps getting discussed over and over again

Advise: Always add proper justification

Email-based decisions

People lose, forget or don't even know a decision

Advise: Architecture Decision Records

Record design decisions

Every design decision is *good enough* but seldom optimal

It is necessary to record justification and risks affected

Things to record:

What evidence was provided to justify the decision?

Who did that?

What are the trade-offs?

What are the main assumptions?

Driver	Design decisions and location	Rationale and assumptions
QA-1	Introduce concurrency (tactic) in the `TimeServerConnector` and `FaultDetectionService`	Concurrency should be introduced to be able to receive and process several events simultaneously
QA-2	Use of a messaging pattern through the introduction of a message queue in the communications layer	Although the use of a message queue may seem to go against the performance imposed by the scenario, it hill be helpful to support QA-3
. . .	. . .	. . .

Architectural decision records

Templates: https://adr.github.io/

Basic structure:

Title

Short descriptive title

Status

Proposed, accepted, superseded

Context

What is forcing to make the decision

Include alternatives

Decision

Decision and corresponding justification

Consequences

Expected impact of the decision

Records should be immutable: If a change appears, create a new record that supersedes a previous one

For drafts, it may be useful to use RFCs (Request for comments)

Architectural issues

Architectural issues

Risks

Unknowns

Problems

Technical debt

Gaps in understanding

Erosion

Contextual drift

Risks

Risk = something bad that might happen but hasn't happened yet

Risks should be identified and recorded

Risks can appear as part of QA scenarios

Risks can be mitigated or accepted

If possible, identify mitigation tasks

Risk = perceived probability of failure $\times$ perceived impact

Risk assessment table

Assess risks in two dimensions:

Impact/severity of risk

Probability of risk occurring

Risk matrix 3x3:

Simplify values as: low (1), medium (2), high (3)

	Probability		
Impact	Low (1)	Medium (2)	High (3)
Low (1)	1	2	3
Medium (2)	2	4	6
High (3)	3	6	9

3x3 Risk matrix

Risk Criteria \ Area	Customer registration	Order Fulfillment
Scalability	2	1
Availability	3	2
Performance	4	3
Security	6	1
Data integrity	9	1

Example of risk assessment table

Risk storming: recommended exercise to collaboratively evaluate risks

Unknowns

Sometimes we don't have enough information to know if an architecture satisfies the requirements

Under-specified requirements

Implicit assumptions

Changing requirements

...

Architecture evaluations can help turn unknown unknowns into known unknowns

Problems

Problems are bad things that have already passed

They arise when one makes design decisions that just doesn't work out the desired way

They can also arise because the context changed

A decision that was a good idea but no longer makes sense

Problems can be fixed or accepted

Problems that are not fixed can lead to technical debt

Technical debt

Debt accrued when knowingly or unknowingly wrong or non-optimal design decisions are taken

If one pays the instalments the debt is repaid and doesn't create further problems

Otherwise, a penalty in the form of interest is applicable

If one is not able to pay the bill for a long time the total debt is so large that one must declare bankruptcy

In software terms, it would mean the product is abandoned

Several types:

Code debt: Bad or inconsistent coding style

Design debt: Design smells

Test debt: Lack of tests, inadequate test coverage,...

Documentation debt:

Outdated documentation,

No documentation for important concerns,

...

Gaps in understanding

They arise when what stakeholders think about an architecture doesn't match the design

In rapidly evolving architectures gaps can arise quickly and without warning

Gaps can be addressed though education

Presenting the architecture

Asking questions to stakeholders

Architectural erosion (drift)

Gap between designed and as-built architecture

The implemented system almost never turns out the way the architect imagined it

Without vigilance, the architecture drifts from the planned design a little bit every day until the implemented system bears little resemblance to the plan

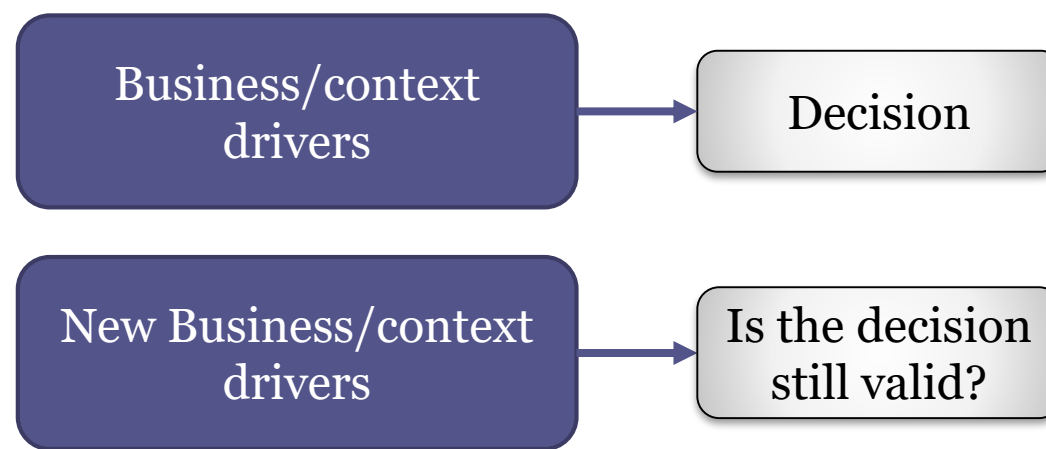Architecturally evident code can mitigate drift

Contextual drift

It happens any time business or context drivers change after a
design decision has been taken

Necessary to continually revisit requirements

Evolutionary architecture

| Business/context drivers | → | Decision |

| New Business/context drivers | → | Is the decision still valid? |

Architectures evaluation

Architecture evaluation

ATAM (Architecture Trade-off Analysis Method)

Architecture evaluation method

Simplified version of ATAM:

- Present business drivers

- Present architecture

- Identify architecture approaches

- Generate quality attribute utility tree

- Analyse architectural approaches

- Present results

Cost Benefit Analysis Method (CBAM)

1. Choose scenarios and architectural strategies
2. Assess quality attribute benefits
3. Quantify the benefits of each strategy
4. Quantify the costs and schedule implications
5. Calculate the desirability of each option

$$VFC \ (Value \ For \ Cost) = \frac{Benefit}{Cost}$$

6. Make architectural design decisions

More info: https://slideplayer.com/slide/12550708/
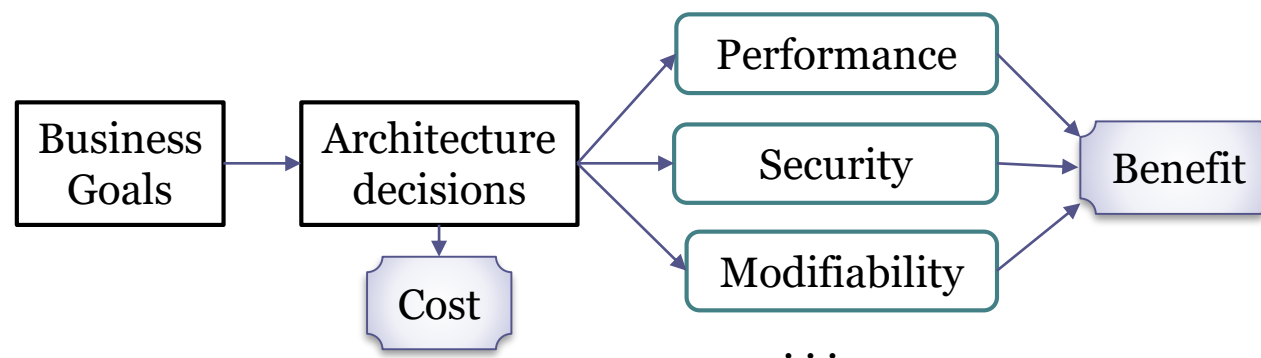
The end

Other slides

4 principles of design thinking

Design for humans

All design is human in nature

Design for change

Delay some decisions until least responsible moment

All design is redesign

Explore patterns, styles and past designs

Make the architecture tangible

Communicate the architecture