

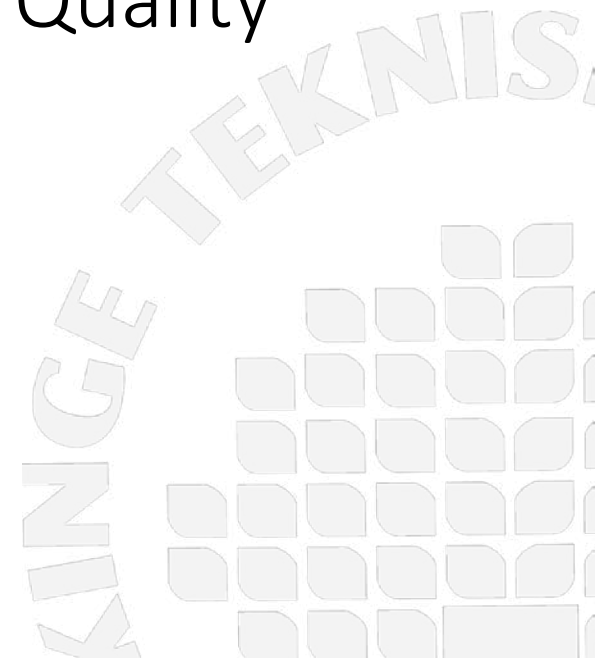


PA1422: Software Architecture and Quality

Software Architecture Design

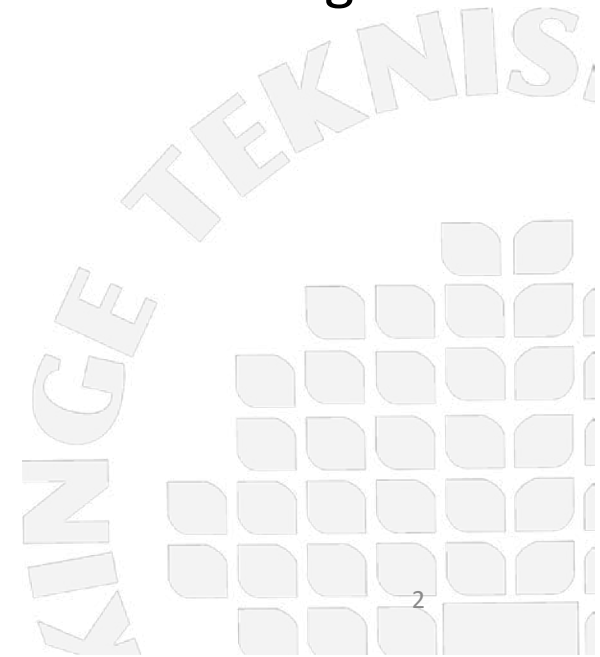
Javier Gonzalez Huerta

Thursday, 17th November, 2016



Objectives

- Understand the difficulties of architectural design
- Discuss structured approaches to architectural design
- Discuss the importance of competing qualities in architectural design



What is the purpose of Architectural Design?

- **Fulfill the needs**

- Functionality
- Qualities

- **Create *added value***

- What is this added value?
- Usually can be mapped either into functionality, into qualities or into money

The Good and Bad News

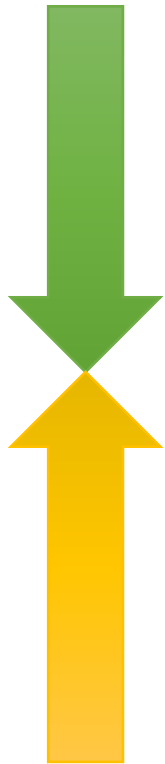
- **Bad news:**

- There is no automatic /algorithmic way for doing architectural design!
- So as architects we have to design the architectures without assistance

- **Good news**

- There is no automatic /algorithmic way for doing architectural design!
- If there is no way of automating the architectural design, there is a future career for humans as software architects

Top-Down vs Bottom-Up




- **Top Down strategy:**

- Start by designing the highest abstractions and structures first
- Decide the styles and patterns to be applied and refine the views
- Continue with the non-architectural design
- When all this is done, start the implementation

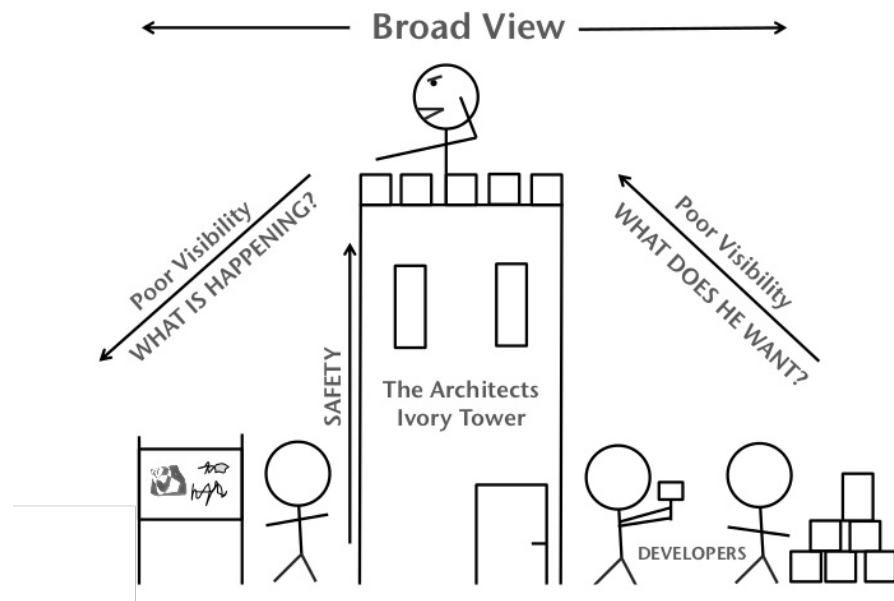
- **Bottom up strategy:**

- Start coding
- Find the abstraction and the structures in the code

Top-Down Strategies: Drawbacks

- 
- A large green arrow pointing downwards, indicating a flow or progression from the title to the list of drawbacks.
- We cannot start developing / coding until we have the architecture
 - Time pressure
 - Ivory-tower architects, those without connection with the real implementation problems and the code

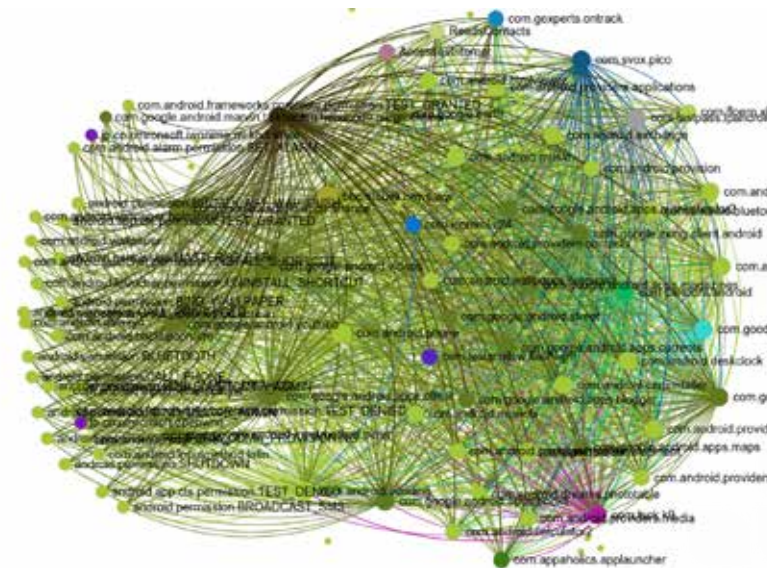
TOWER SYNDROME



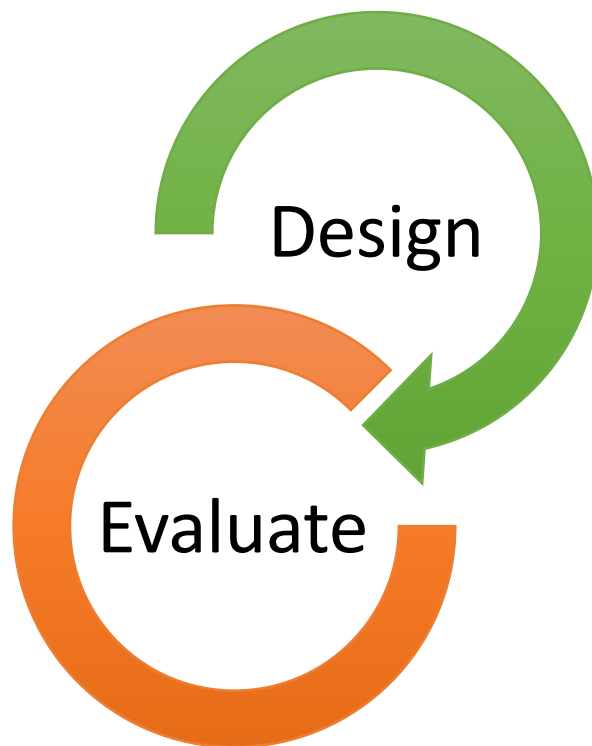
Andrew Rendell: The role of an architect in an agile

Bottom-up Strategies: Drawbacks

- Architecture not well structured and organized
- Implementation usually will not meet the quality requirements



Iterative Architectural Design



- Design and Evaluate
- Repeat and Modify based on what you found in the evaluation

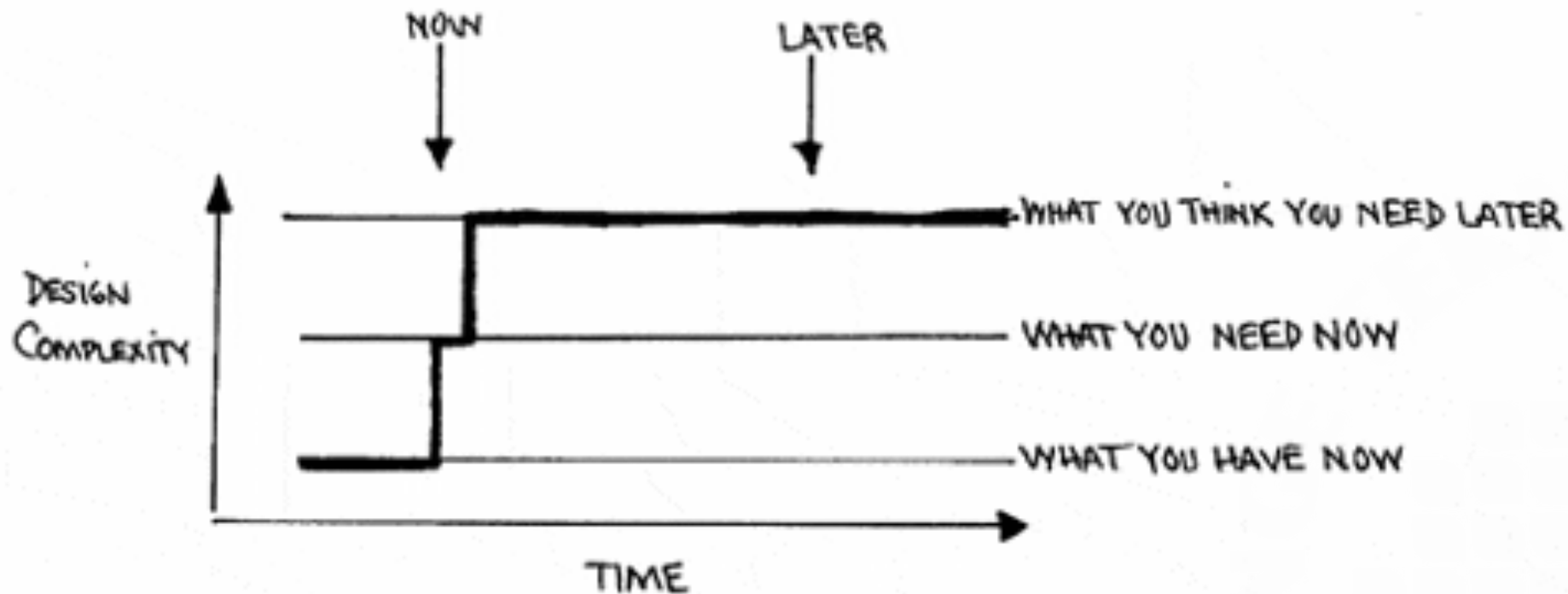
Incremental Architectural Design



- Start with a use case and design the simplest architecture
- Add a one more use case and extend the architecture to support the added use cases
- Until you have supported all the relevant use cases

Traditional Approaches: Architecting for Tomorrow

Developers are really good at trying to anticipate future problems and design for the future

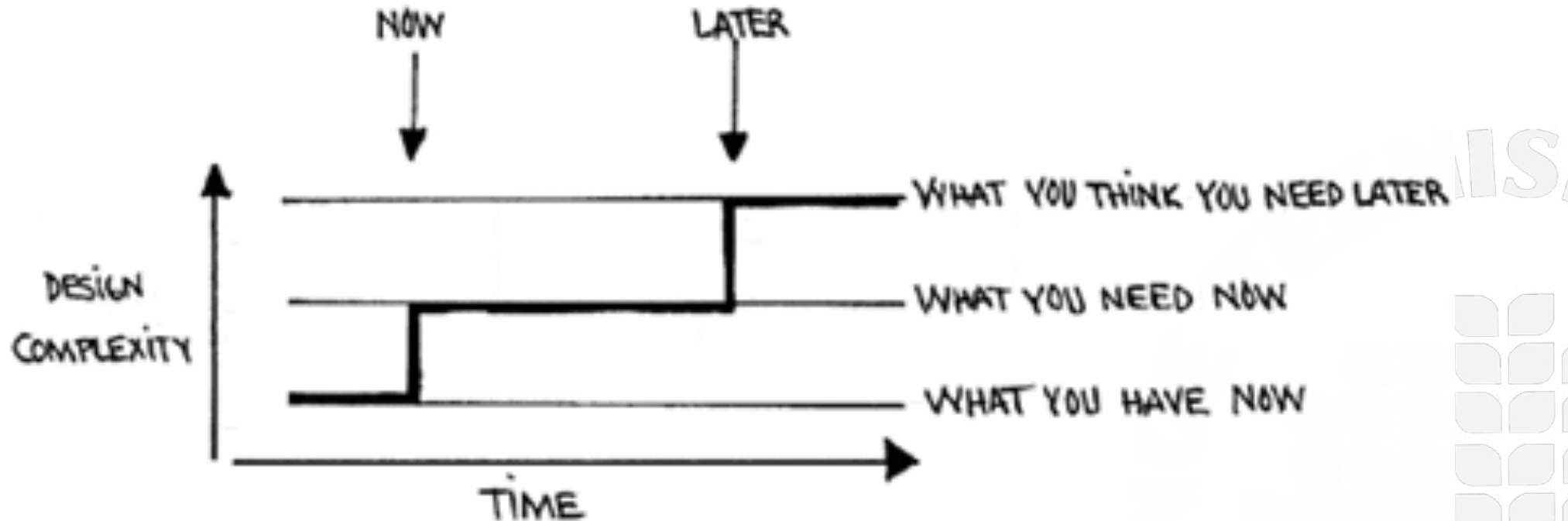


Beck, Kent, and Cynthia Andres. 2005. Extreme Programming Explained: Embrace Change (The XP Series). Addison-Wesley Professional.

Traditional Approaches: Architecting for Tomorrow

- The problem of this approach is the uncertainty
- What happens if tomorrow never comes?
 - May be the changes we thought ahead are removed from the product by the customer
 - May be we need to change something else that makes the new architecture not feasible
 - May be we learn how to do things better between *now* and *later*

Agile Architecting and Design



Beck, Kent, and Cynthia Andres. 2005. *Extreme Programming Explained: Embrace Change (The XP Series)*. Addison-Wesley Professional.

Evolutionary Architecting

- All approaches are evolutionary
 - Long architecting cycles: Architectural releases
 - Short architecting cycles: small design decisions
- Evolutionary Architecting Benefits
 - Build on the experience of previous iterations
 - Build new experience
- Needs
 - Refactor to improve the qualities

Refactoring

- **Refactoring:** Controlled technique to change the structure of an existing code, improving certain qualities
 - The essence is apply **small behavior-preserving changes** and transformations, each of which, if isolated is **“too small to be worth doing”**

Refactoring

- **Refactoring (noun):** a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- **Refactoring (verb):** to restructure software by applying a series of refactorings without changing its observable behavior.

M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

Agile Refactoring

- The main principle is to implement the given feature or change request
- Apply the refactorings needed to make the implementation of the feature or change request easier (anywhere on the code)
- When testing the new development the refactorings are tested also "for free"

Refactoring vs (Re)Architecting

- Sometimes these small changes made during the development are not enough or can start hindering the actual architecture
- We can have also changes on the context of the project (at functionality or quality level)
- In these cases big changes to the structure are needed
 - Means that we are not doing refactoring, but re-architecting
 - Costly (Gain vs Pain)

Software Architecture Design

An Structured Approach to Find Architectural Drivers: Global Analysis

An Structured Approach to Find Architectural Drivers: Global Analysis

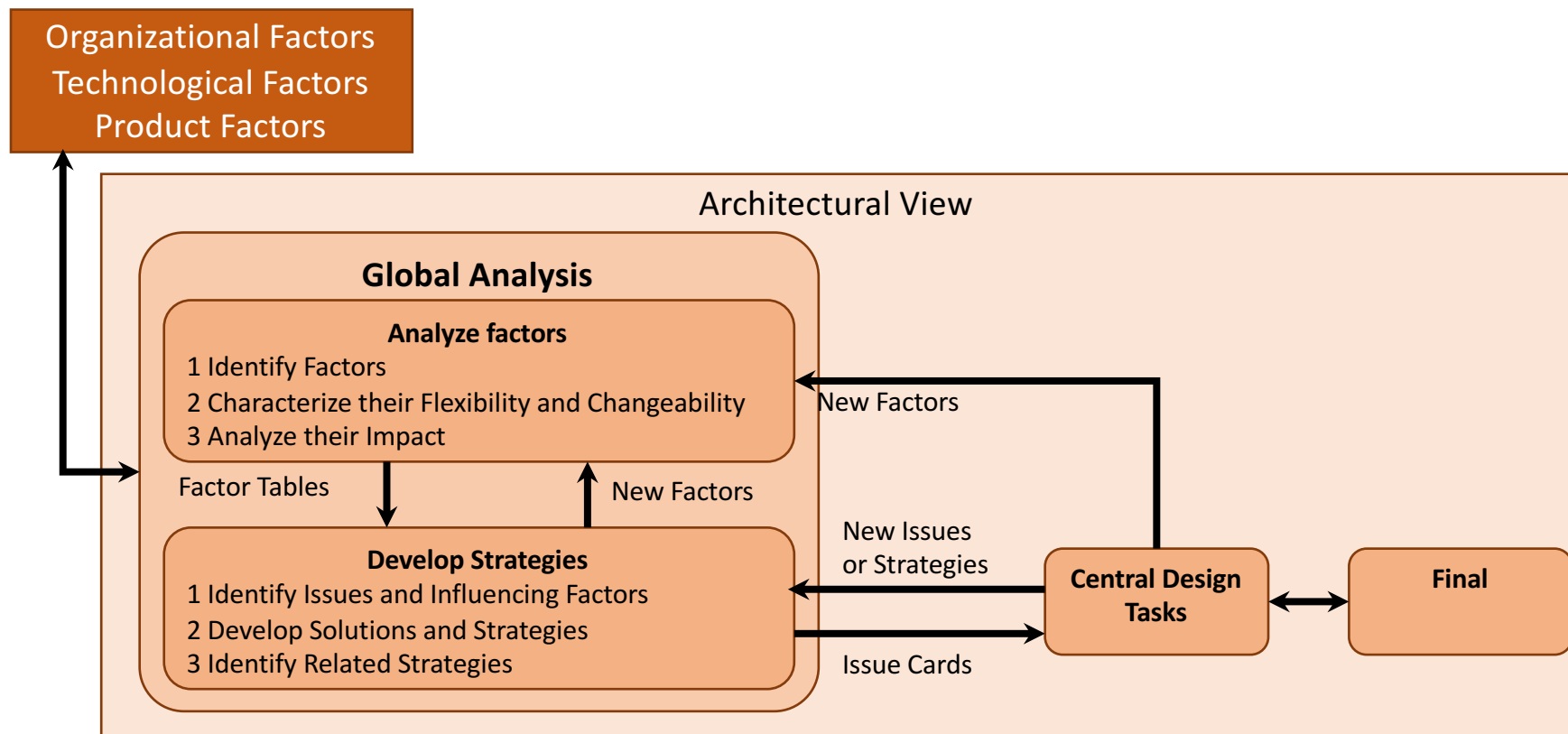
- For each view:

Global Analysis

Central Design Tasks

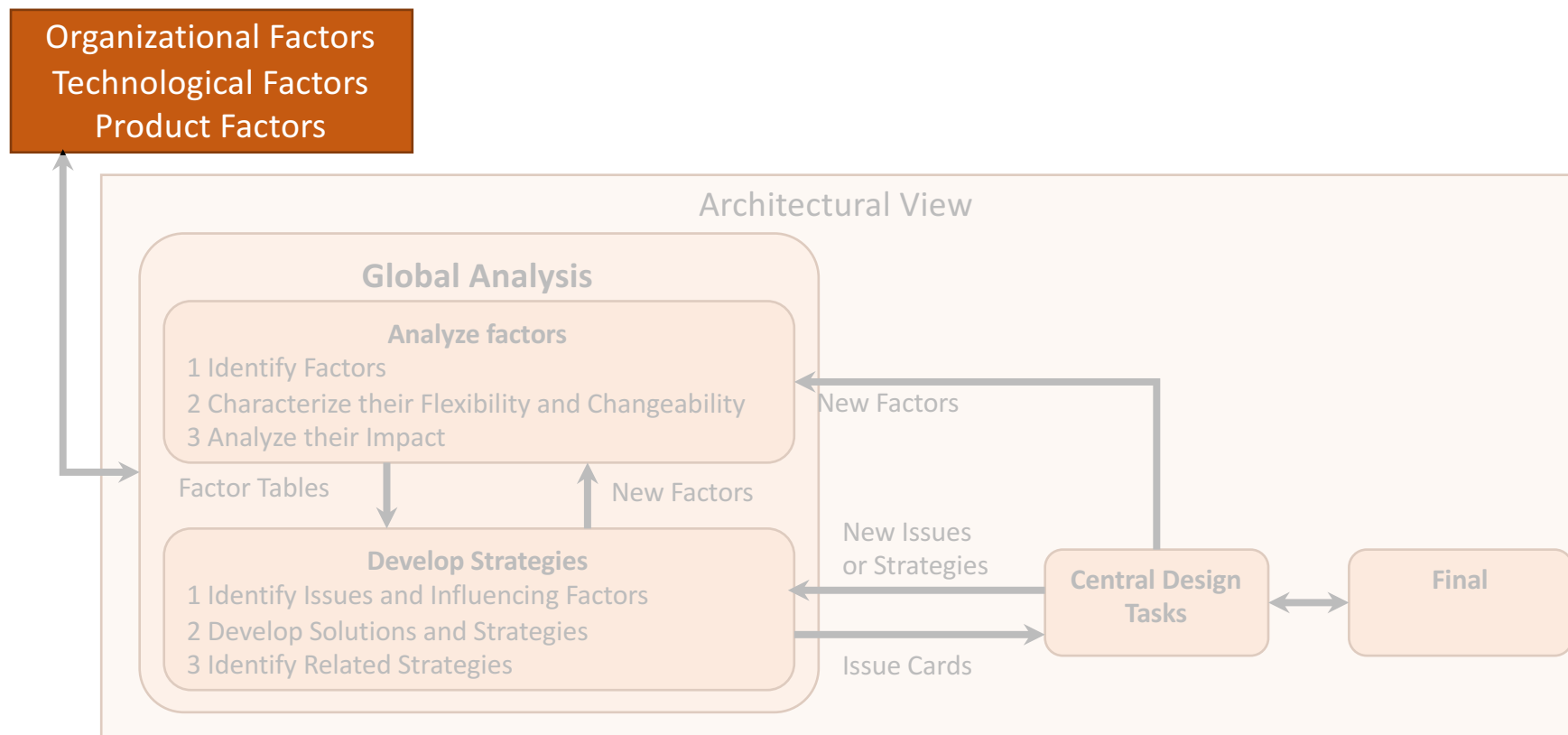
Final Design Tasks

An Structured Approach to Find Architectural Drivers: Global Analysis



C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Addison-Wesley Professional, 2000.

An Structured Approach to Find Architectural Drivers: Global Analysis



Identify and Analyze Factors

- Organizational Factors
 - Management: Build vs Buy,...
 - Staff skills, Interests, Strengths...
 - Process and Development Environment: Development Platform, Processes, Tools
 - Development Schedule: Time to market, delivery of features
 - ...
- Technological Factors
- Product Factors

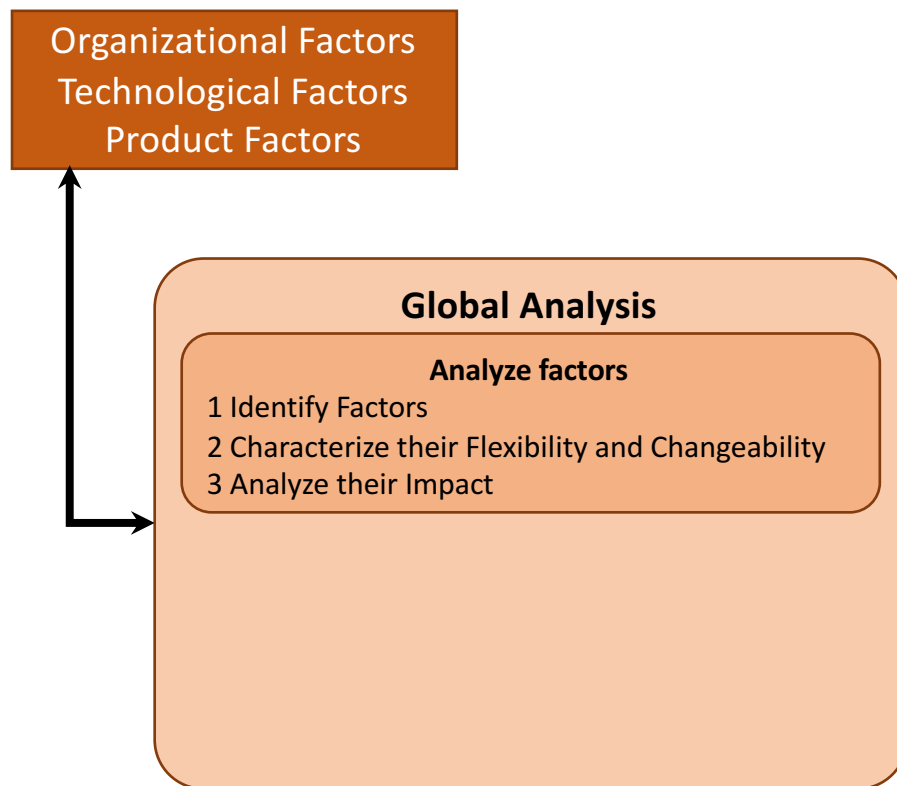
Identify and Analyze Factors

- Organizational Factors
- Technological Factors
 - General-Purpose Hardware: Processors, Network, Memory, Disk -
 - Domain-Specific Hardware: Sensor hardware, actuator hardware
 - Software Technology: Operating System, User Interface, Software Components, Implementation Language, Design Patterns, Frameworks
 - Architecture Technology: Architecture styles, patterns, frameworks, domain specific architectures, architecture description languages, product-line-technologies
 - Standards
- Product Factors

Identify and Analyze Factors

- Organizational Factors
- Technological Factors
- Product Factors
 - Functional Features
 - User Interface
 - Performance, Dependability (Availability, Reliability, Safety)
 - Failure detection, reporting, recovery: Error classification, Error logging, diagnostics, recovery
 - Service: Service features, software installation and upgrade, maintenance of hardware, software testing, software maintenance
 - Product Cost: hardware budget, software licencing budget

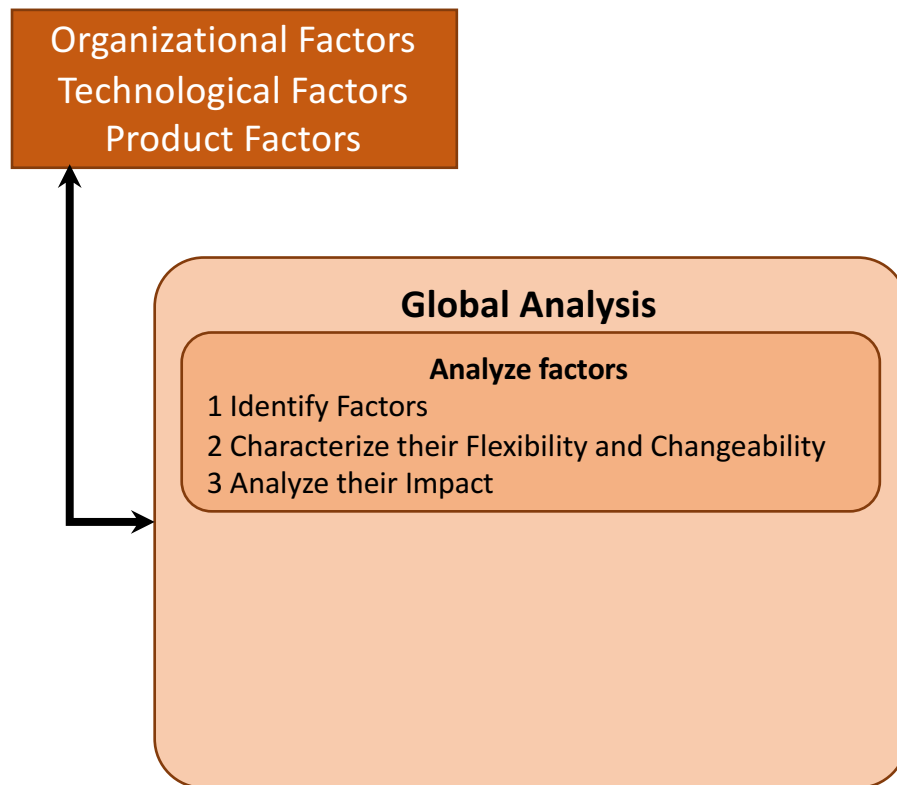
Analyze factors



Step 1. Identify Factors: Does the factor have any global influence (does it need to be dealt by or with the architecture)?

- Can the factor's influence be localized to one component in the design?
- During which stages of development is the factor important?
- Does the factor require any new expertise or skill?

Analyze factors

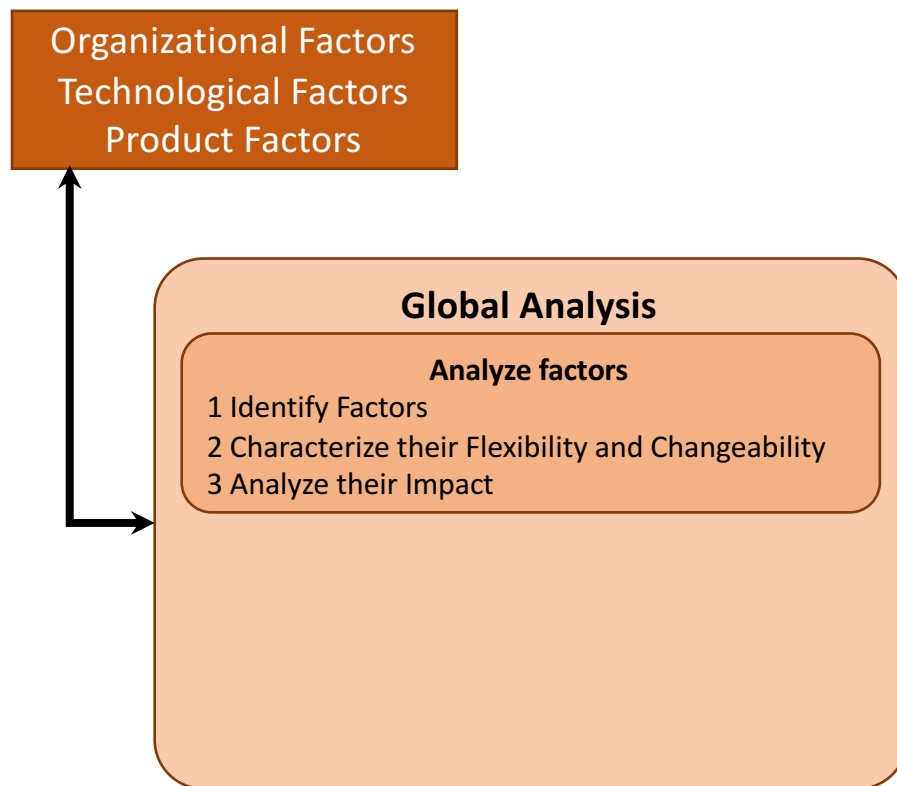


Step 2. Characterize their Flexibility and Changeability

What can be negotiated? (Flexibility) / What may change? (Changeability)

- **F** Is it possible to influence or change the factor to make your task of architecture development easier?
- **F** In what way can you influence it?
- **F** To what extent can you influence it?
- **C** In what way could the factor change?
- **C** How likely will it change during or after development?
- **C** How often will it change?
- **C** Will the factor be affected by changes in other factors?

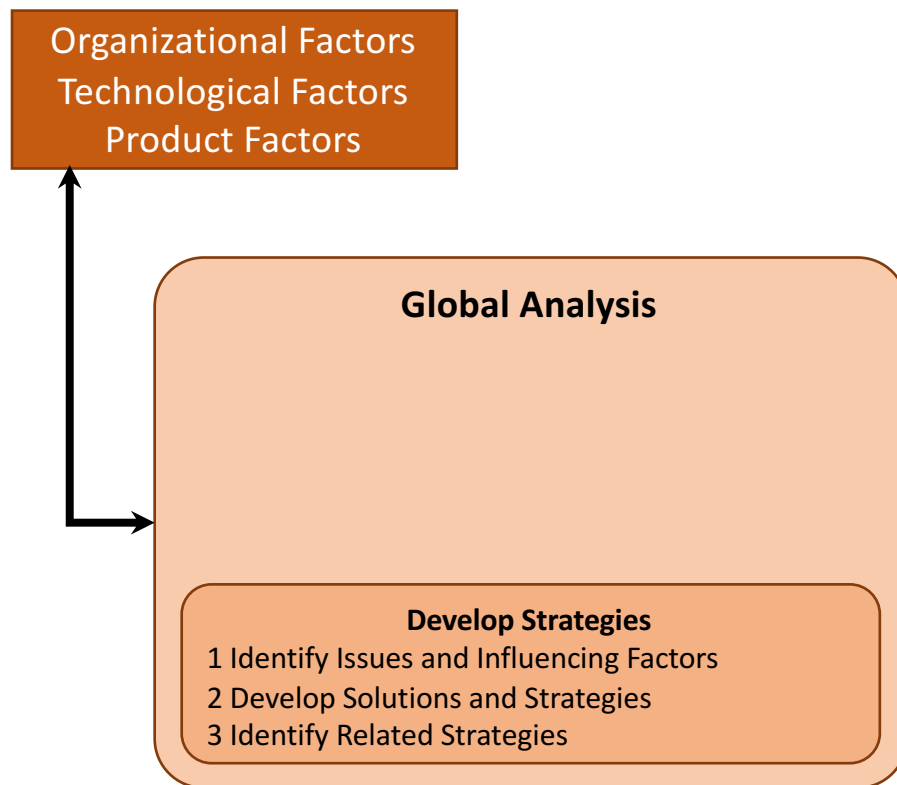
Analyze factors



Step 3. Analyze their Impact: If the factor was to change, what would be affected and how?

- Other factors
- Components
- Modes of operation of the system
- Other design decisions

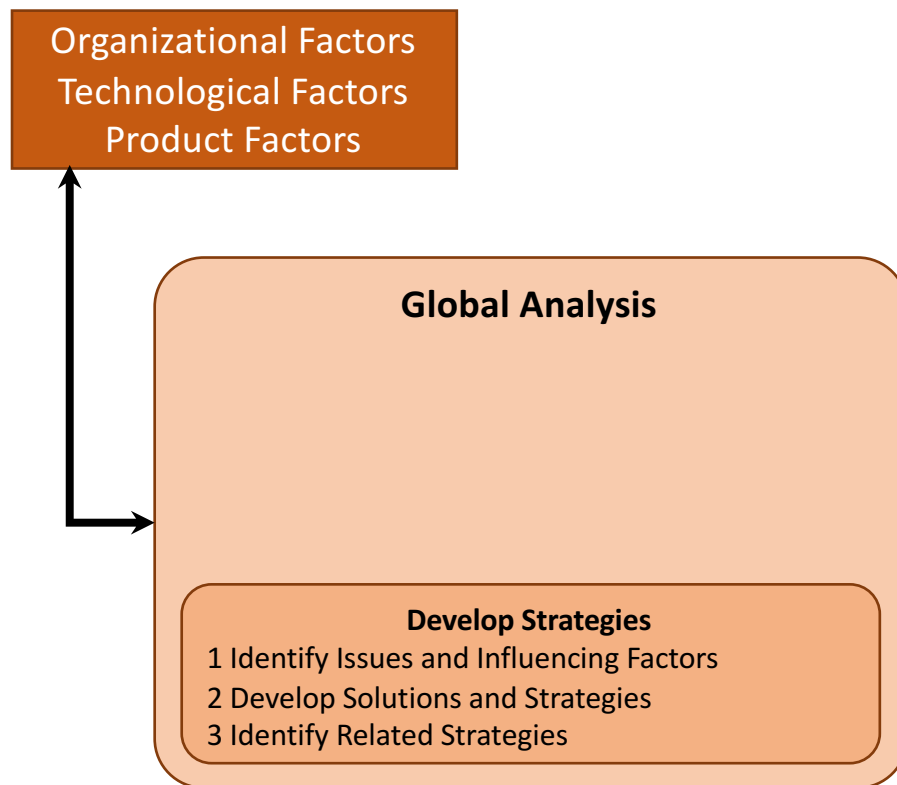
Develop Strategies



- **Step 1: Identify Issues and Influencing Factors**

- An Issue is something we need to address in the architecture, often stemming from several factors and their changeability
 - Might arise from constraints imposed by factors (Example: buy vs build)
 - May result from the need to reduce the impact of changeability of factors. (Example: Design the architecture to reduce the cost of porting to new Operating systems)
 - May result because of difficulties in satisfying product factors (Example: high resolution and high frame rate for a game / game engine)
 - An issue may arise from the need to have a common solution to global requirements (Example: high availability and low latency time for an online game)

Develop Strategies



• Step 2: Develop Solutions and Strategies

- How you plan to solve the issue. Each strategy should be consistent with
 - Influencing factors
 - Changeability
 - Interactions with factors
- Address the following goals
 - Reduce the factor's influence
 - Reduce the impact of the factor changeability on the design and on other factors
 - Reduced or localize required areas of expertise or skills
 - Reduce overall time, cost, and effort

Example: GalaxyUML

Design the architecture of a multi-platform clone of StarUML, also for modeling with ADLs such SysML and AADL and that allows multi-touch interfaces, as well as cooperative design in distributed teams

- Main Features
 - Multiple SOs (Windows, Linux, MacOS, iOS, Android)
 - Multi-touch & surface technology
 - Sketch recognition (hand-drawing recognition and translation to modeling primitives)
 - Cloud Persistency
- Potential factors that make it complicated
 - Multi-cloud
 - Concurrency problems
 - Imperfect inputs in Sketch recognition

GalaxyModeling: Multi-Touch, Multi-User Modeling Tool

Product Factor	Change Flexibility	Impact
P1 Functional Features		
Several Modeling Languages supported (UML / SYSML / AADL) and their correspondent types of diagrams	C New Modeling Languages / Diagrams can come later	This affects to how we detect model entities, how we store the models, how we render and if there are relationships between elements in different diagrams
Several output image formats are supported	C New image formats can come later	This affects the compression components used and the storage components
P2 User Interface		
Multi-Touch User Interface	C The gesture requirements are stable	This affects the UI component, the rendering component and the layout component
Multiple-users editing the same model interactively in different screens	F Limit concurrency on the same entities, but this is optional	This might affect concurrency mechanisms

GalaxyModeling: Multi-Touch, Multi-User Modeling Tool

Product Factor	Change Flexibility	Impact
T1 Software Technology		
Persistence Cloud Servers	C Change / New cloud providers	We will have to isolate the cloud provider details
Multiplatform	F Initially Surface, Android, IOS	This affects the model entities detector, how we store the models, how we render and if there are relationships between elements in different diagrams

Develop Strategies

Develop Strategies

- 1 Identify Issues and Influencing Factors
- 2 Develop Solutions and Strategies
- 3 Identify Related Strategies

STEP 1. Identify Issues: An Issue is something we need to address in the architecture, often stemming from several factors and their changeability

- Might arise from constraints imposed by factors (Example: buy vs build)
- May result from the need to reduce the impact of changeability of factors. (Example: Design the architecture to reduce the cost of porting to new Operating systems)
- May result because of difficulties in satisfying product factors (Example: high resolution and high frame rate for a game / game engine)
- An issue may arise from the need to have a common solution to global requirements (Example: high availability and low latency time for an online game)

Develop Strategies

Develop Strategies

- 1 Identify Issues and Influencing Factors
- 2 Develop Solutions and Strategies
- 3 Identify Related Strategies

STEP 2 Develop Solutions and Strategies

How you plan to solve the issue. Each strategy should be consistent with

- Influencing factors
- Changeability
- Interactions with factors

Address the following goals

- Reduce the factor's influence
- Reduce the impact of the factor changeability on the design and on other factors
- Reduced or localize required areas of expertise or skills
- Reduce overall time, cost, and effort

Develop Strategies (Examples)

- *Multiple Modeling Languages, Diagrams and Output Formats*
 - Strategy: Separation of recognition, rendering and persistence
- *Multiple users interacting at the same time*
 - Strategy: block entities to avoid multiple users interacting with the same entity (e.g., Mutex)
- *Cloud Persistency*
 - Strategy: GUI communicating only actions, broker pattern to hide cloud details

Defining the Conceptual View

- Which Issues are useful to create this view?
- Which Factors?

Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Defining the Conceptual View

- Which Issues are useful to create this view?
 - *Multiple Modeling Languages, Diagrams and Output Formats*
 - Strategy: Separation of recognition, rendering and persistence
 - *Multiple users interacting at the same time*
 - Strategy: block entities to avoid multiple users interacting with the same entity (e.g., Mutex)
 - *Cloud Persistency*
 - Strategy: GUI communicating only actions, broker pattern to hide cloud details
- Which Factors?

Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Defining the Conceptual View

- Which Issues are useful to create this view?
 - *Multiple Modeling Languages, Diagrams and Output Formats*
 - Strategy: Separation of recognition, rendering and persistence
 - *Multiple users interacting at the same time*
 - *Cloud Persistency*
- Which Factors?

Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Defining the Conceptual View

- Which Issues are useful to create this view?
 - *Multiple Modeling Languages, Diagrams and Output Formats*
 - *Multiple users interacting at the same time*
 - Strategy: block entities to avoid multiple users interacting with the same entity (e.g., Mutex)
 - *Cloud Persistency*
- Which Factors?

This will not add new conceptual elements, this will be something to be considered in the execution view

Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Defining the Conceptual View

- Which Issues are useful to create this view?
 - *Multiple Modeling Languages, Diagrams and Output Formats*
 - *Multiple users interacting at the same time*
 - *Cloud Persistency*
 - **Strategy: GUI communicating only actions, broker pattern to hide cloud details**
- Which Factors?

This will not add new conceptual elements, this will be something to be considered in the execution and deployment views

Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Defining the Conceptual View

- Which Issues are useful to create this view?
 - *Multiple Modeling Languages, Diagrams and Output Formats*
 - *Cloud Persistency*
 - *Multi-platform*
 - Isolate GUI and the functional features, and use cross-platform APIs
- Which Factors?

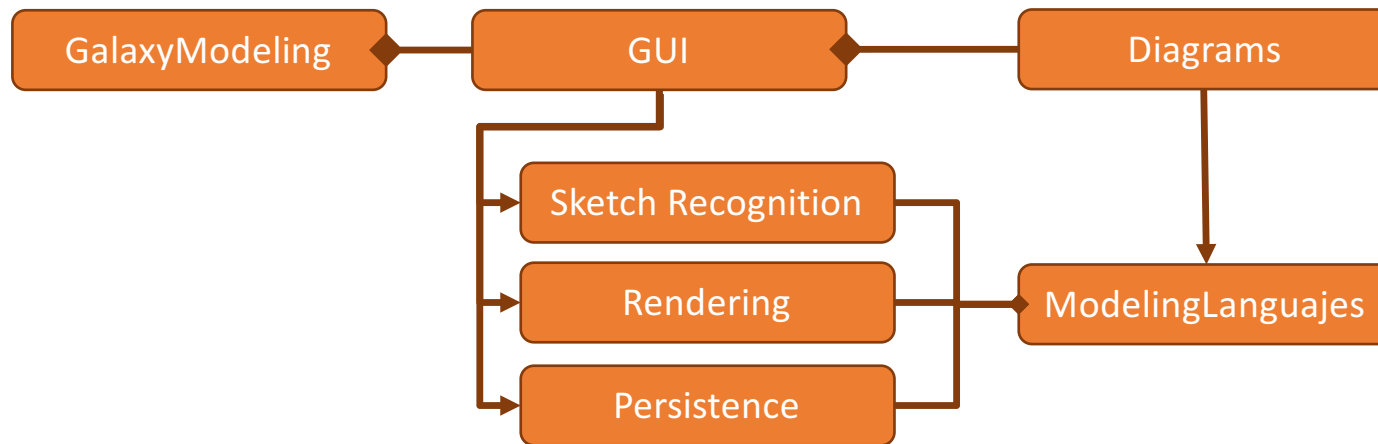
Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Defining the Conceptual View

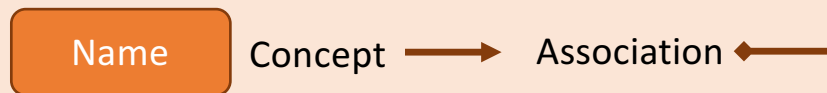
- Which Issues are useful to create this view?
- Which Factors?
 - Several modeling languages supported
 - Several output formats
 - Multi-Touch GUI

Note: Creating viewpoints is not the goal of this lecture; this is included as a reference to understand how to go from issues and strategies to a design in a traceable way.

Conceptual View (Super Rough Simplification)



Legend:



What's Next?

- The identification of issues and factors only gives a rough traceability to the elements on the views
- We will have to work with a big box and start adding elements to your views to cope with the issues and factors (some of the elements where already identified)
- Address factors and issues with:
 - Styles and patterns (we already saw broker style, the Mutex, but layering can be also a natural style)
 - Domain-specific reference architectures

Summary

- Global Analysis is a *tool* to help you identify architecture relevant concerns
- It provides a *structured approach*
- It provides basic *traceability* from your concerns to your actual architecture
- It does not help you to actually create a workable architecture; it just makes sure you do not forget things

Software Architecture Design

Quality Requirements Trade-offs

System's Driving Requirements

- Some quality requirements are more important / critical than others
 - Therefore these requirement will have more impact on the architectural decision
-
- Driving requirements of a nuclear plant control systems are safety or fault tolerance
 - Driving requirements of a game might be high performance

System's Driving Requirements

- Driving requirements allow us to take smart trade-off decisions
- For example
 - If fault tolerance is critical
 - We can prioritize my decisions to make my system highly redundant
 - What is the price I'll pay in terms of complexity and resource consumption?

Conflicting Requirements

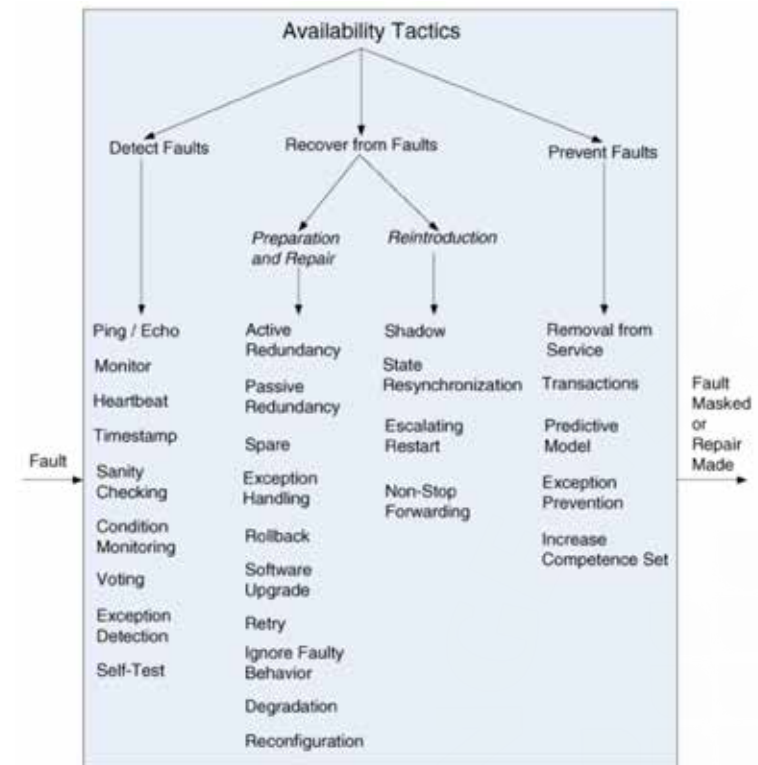
- Some of the qualities might conflict each others and also can be competing for the same resources in the system or during the development
- For example,
 - Processing capacity versus development speed
 - Memory is needed both for storing data AND to store data structures that make development simple
 - CPU is needed to do business logic AND to navigate complex data structures

Quality Trade-offs

- Since not all qualities are not equally important for each stakeholder we will have to perform trade-offs
- Some qualities can be trade-off against some other directly
 - Fault Tolerance (which introduced complexity) vs Maintainability (which suffers from complexity)
- Doing trade-offs between conflicting qualities can give us a competitive advantage

Quality Design Tactics

- Each general quality has associated a well know set of tactics to help achieve or improve it
- Often these come from a field knowledge of its own
- We don't need to reinvent the wheel every time!
- Also we can consider tactics when doing our global analysis



Software Architecture Design

Design Constraints and Implementation Rules

Once we have the views

- Once we have the views, we have to define and document the constraints and rules that, being not visible, help to develop the system while following the architecture

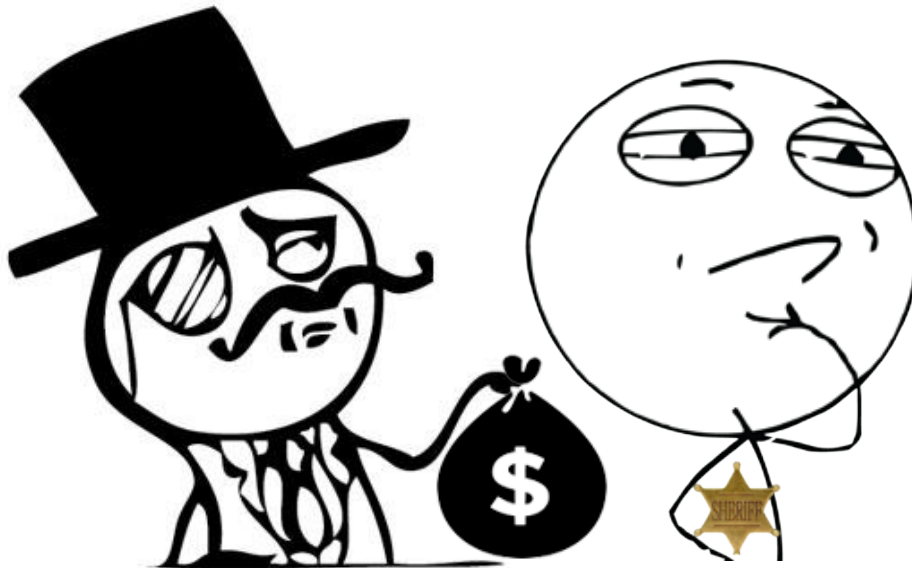
Design Constraints and Implementation Rules

- **Design constraints** - Fix boundaries of the design
- **Implementation rules** - Allow the architect and developers meeting:
 - The design constraints
 - The NFRs
 - Implement the use cases

Design Constraints

"Investors"

Architect



- Imposed by the business or financial environment
- law/business/agreements
- **Examples**
 - Let's use Swift
 - We are going to use non-commercial, open source libraries
 - Everything should be in the cloud
 - We'll only use Microsoft Azure Servers
 - All changes on the REST API should be approved by the Change Control Board

Implementation Rules

- Rules established by the architect (and the team):
- Examples:
 - Components on each layer can only communicate / invoke functionality from the layer below
 - We will expose the API through REST
- Are these rules that a programmer must follow to ensure that the code is actually implementing the architecture.
- Breaking a rule also means the implementation breaks the architecture.

Architect



Developers





PA1422: Software Architecture and Quality

Software Architecture Design

Javier Gonzalez Huerta

Monday, 17th November, 2016

