PV260 SOFTWARE QUALITY

SOFTWARE ARCHITECTURE AND ITS ROLE IN SOFTWARE QUALITY

ONDREJ KRAJICEK @ Y SOFT

The goal of this talk is to provide good enough answers to two rather fundamental questions: “What is Software Architecture” and “How is it related to quality of software?”

Software architecture as a discipline.

Can there be no architecture in a software system?

Principles of good architecture according to Gilb, Martin and others.

Screaming architecture.

Cost of change and expensive questions. Architecture as an artifact and as a process. We all know refactoring, is there "Rearchitecting"?

Anatomy of development team and the role of software architect.

QUESTIONS FOR TODAY?

…AND HOPEFULLY SOME ANSWERS.

- What is Software Architecture?
- What is the role of Software Architecture in Software Quality Assurance?
- Any other business?

A (software) product is a software system which delivers value to deliberately chosen stakeholders.

Software Architecture provides answers to the most difficult technical questions about your product.

Ralph Johnson famously defined software architecture as “the important stuff (whatever that is).”

Software Architecture is a discipline of Software Engineering determining the core principles behind your product.

Principles apply to wide area of contexts. They are (almost) universal.

Patterns are heavily context dependent.

CAN THERE BE NO ARCHITECTURE?

A MILLION DOLLAR QUESTION

COMPETITIVE ENGINEERING

TOM GILB

SOFTWARE ARCHITECTURE IS THE SERVANT OF HIGH-PRIORITY STAKEHOLDER VALUES.

IS AS SIMPLE AS POSSIBLE, BUT NOT SIMPLER AND IS DESIGNED TO BE REPLACEABLE.

(Tom Gilb)

HIGH PRIORITY STAKEHOLDER VALUES

- Stakeholders: vested interest / impact on the success of your product.
  - Stakeholders are Internal vs. External
  - Stakeholder prioritization along Power and Interest axes
- Who determines stakeholders?
- Who determines stakeholder values?

STAKEHOLDER VALUES / PRODUCT QUALITIES

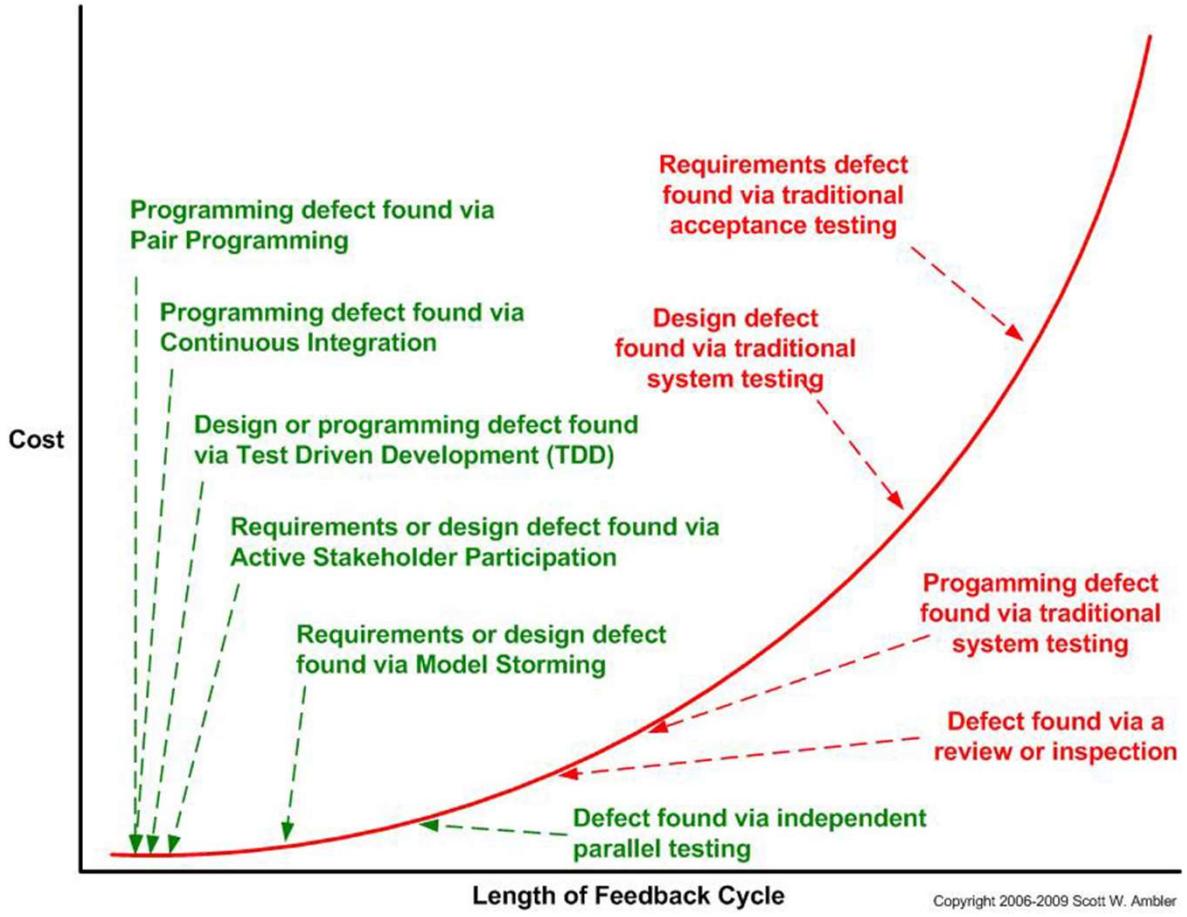
- Stakeholder values translate (trace) into Product Qualities
- Quality is / has…
  - Measurable (in hard numbers)
  - Scale of Measure (lightyears)
  - Tolerable Minimum
  - Goal (rather Objective)
  - Stretch Goal (this is when the goodness of your architecture shows up)
- Surprisingly, almost everything can be quantified and measured.
  - Just do not try to measure people. Measure results or (if need be) processes.

SAMPLE QUALITY

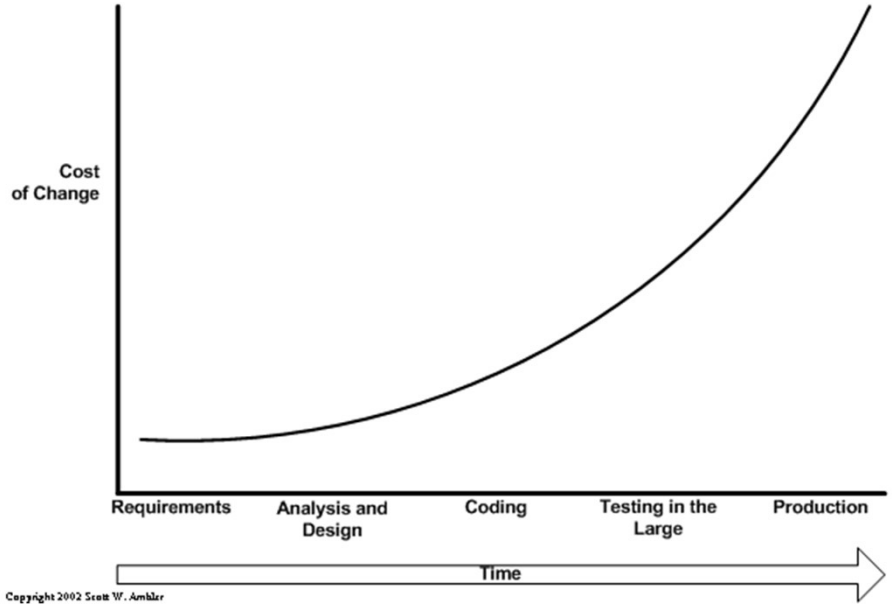
Tag: Ease of Access.
Version: 11-Aug-2003.
Owner: Rating Model Project (Bill).
Scale: Speed for a defined [Employee Type] with defined [Experience] to get a defined [Client Type] operating successfully from the moment of a decision to use the application.
Alternative Scales: None known yet.
Qualifier Definitions:
** Employee Type: {Credit Analyst, Investment Banker, …}.*
** Experience: {Never, Occasional, Frequent, Recent}.*
** Client Type: {Major, Frequent, Minor, Infrequent}.*
Meter Options:
** Test all frequent combinations of qualifiers at least twice. Measure speed for the combinations.*
Known Usage: Project Capital Investment Proposals [2001, London].
Known Problems: None recorded yet.
Limitations: None recorded yet.

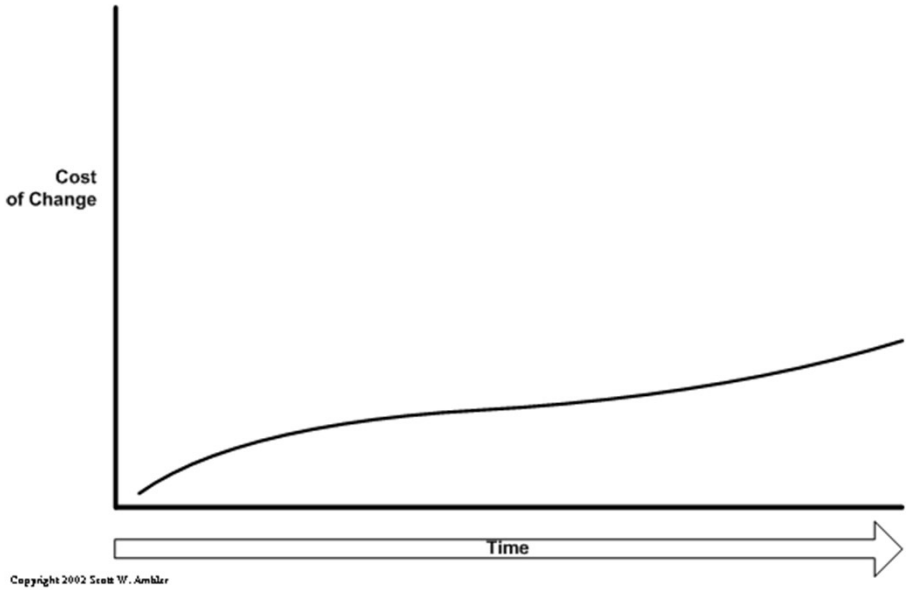
DESIGNED TO BE REPLACEABLE

- Is “Replaceability” a quality?
- How can we measure it?
- Cost of Change
  - Really expensive changes!



WATERFALL VS. “AGILE” COST OF CHANGE





MANAGING COST OF CHANGE

- Do not try to predict everything, nobody is that smart!
- Iterative process with small increments and continuous validation provides insight.
- Close the feedback loop.
- Make it short.

EVOLUTIONARY ARCHITECTURE

NEAL FORD ET AL

EVOLUTIONARY ARCHITECTURE

- An initial part of an architect's job is to understand the business or domain requirements for a proposed solution.
  - Neal Ford et al: Building Evolutionary Architectures [O'Reilly, 2018]
- Architecture is designed around *architectural concerns* (-ilities).
  - Securability, Testability, Trustability, Usability, Scalability, Availability…

The moral: introducing changes to a highly dynamic (eco) system can yield unpredictable results.

— Neal Ford, Building Evolutionary Architectures

HOW CAN WE INTRODUCE CHANGES…

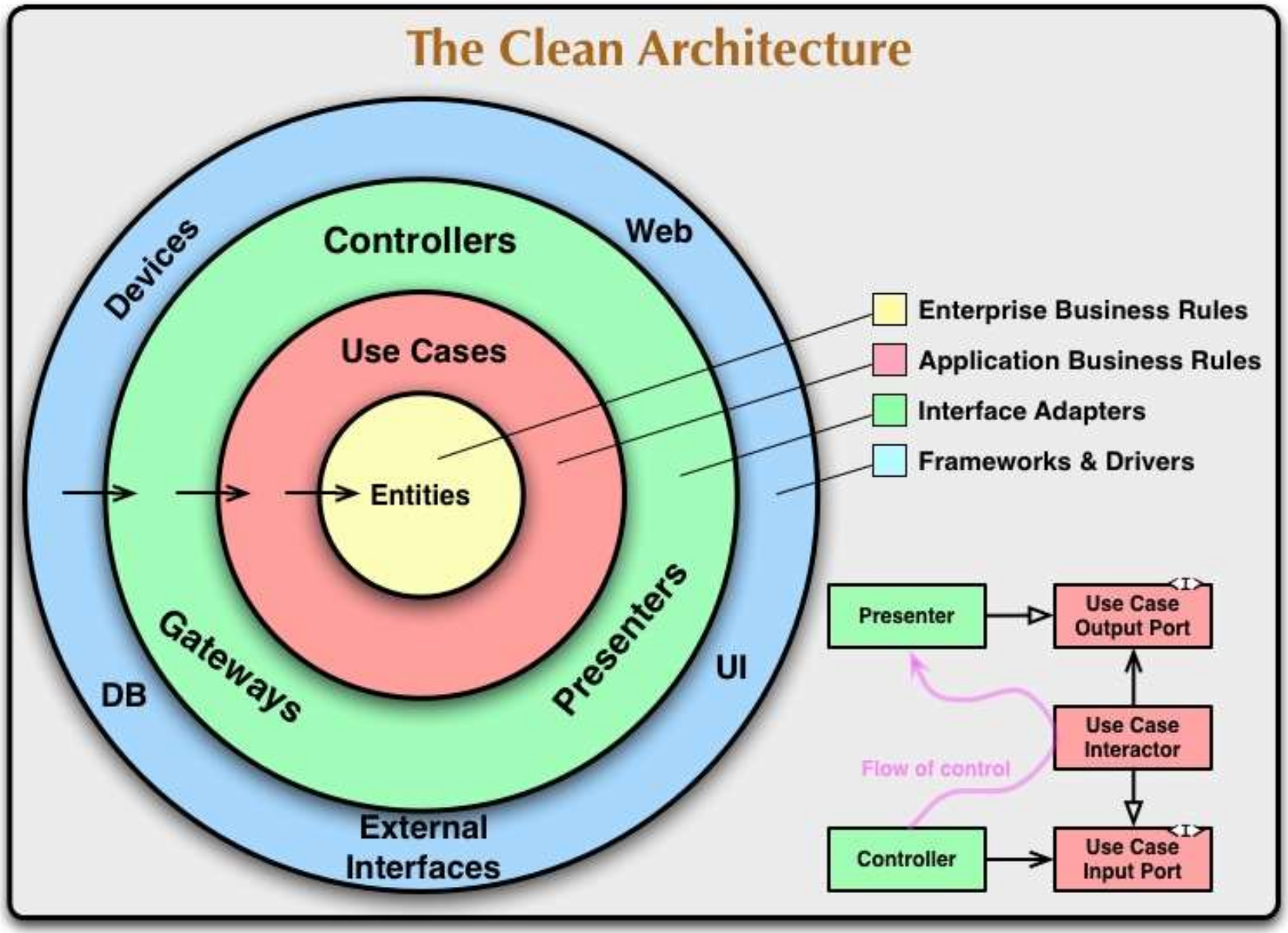
- By protecting the system with **Fitness Functions**.
- Fitness Function: An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

FITNESS FUNCTIONS

- Atomic vs. Holistic
- Static vs. Dynamic
- **Automated vs. Manual**
- Triggered vs. Continual
- Temporal
- **Intentional vs. Emergent**
- **Domain Specific**

SCREAMING / CLEAN ARCHITECTURE

UNCLE BOB MARTIN


The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
<I>
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port
<I>

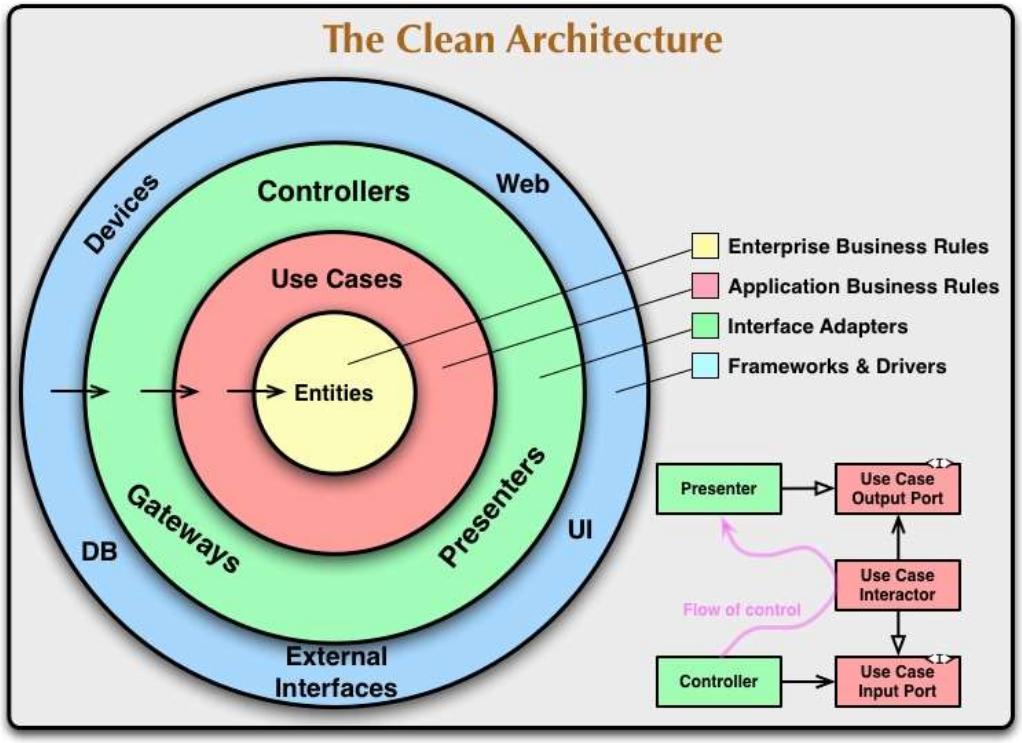
PROPERTIES OF GOOD ARCHITECTURE

1. Independent of Frameworks.
2. Testable: all parts and as a whole.
3. Independent of UI.
4. Independent of the data store / database / object persistence.
5. Independent of any external impact.

PRINCIPLES OF GOOD ARCHITECTURE

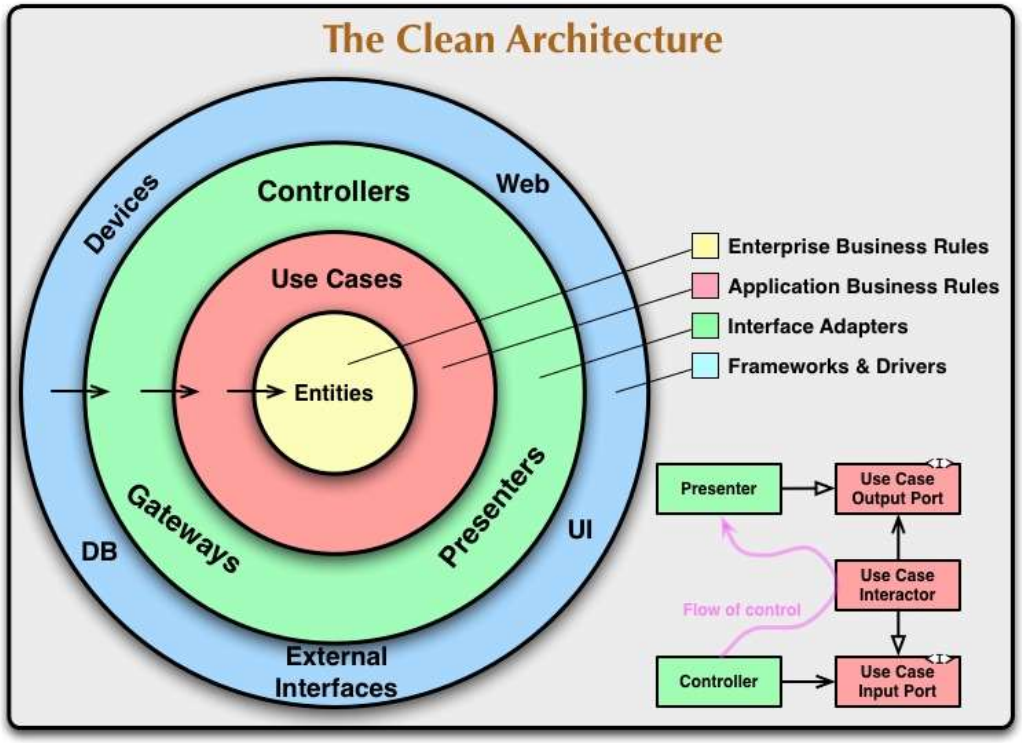
- **Dependencies** (source code dependencies) can **only point inwards**.
  - Assuming a layered / circular model, the outer layers / rings depend on the inner layer rings, not vice versa.
  - This applies not only to behavior (classes, functions, services, etc.), but also to the data / conceptual model, entities, protocols, data formats, etc.
- Used hand in hand with the **Dependency Inversion** principle.
- **Separation of Concepts** to isolate and encapsulate knowledge and responsibility.

ENTITIES



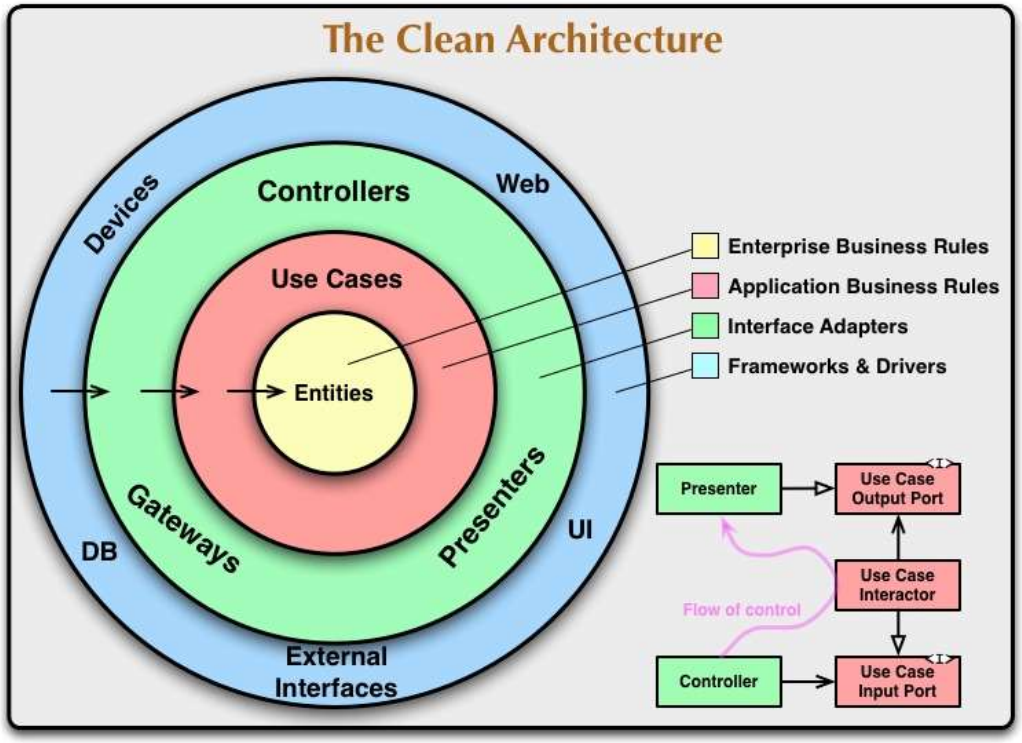
- Based on Concepts and Context of your system
  - Some modeling approaches incorporate Conceptual and Context modeling
  - These are the core elements of your problem domain

USE CASES



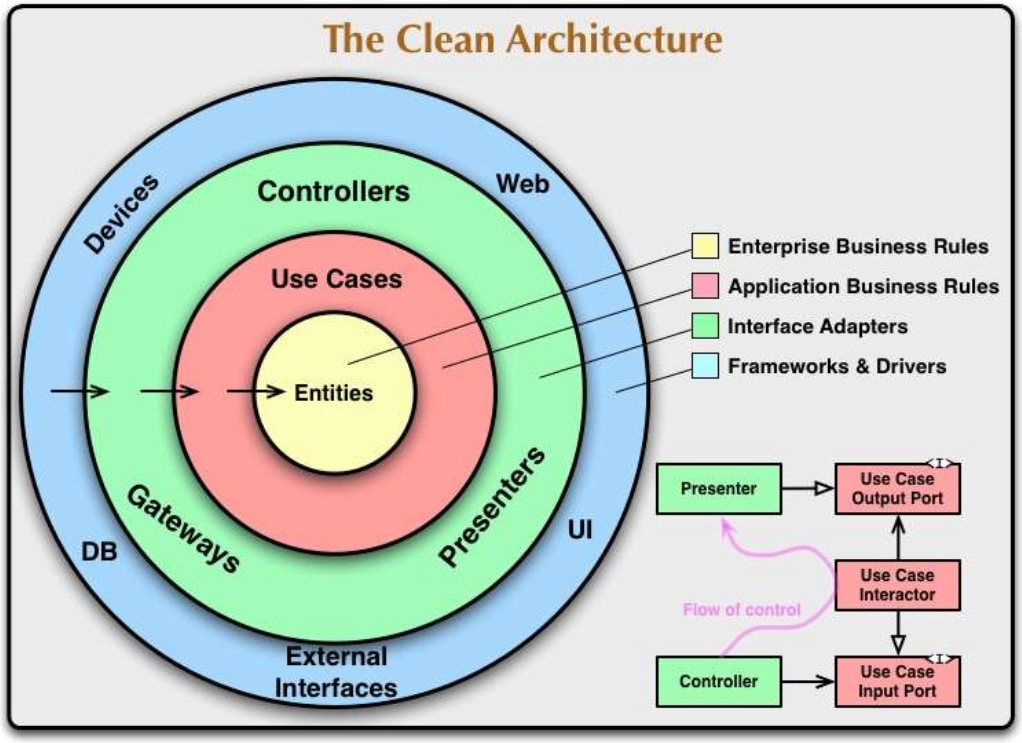
- Use cases provide behavior specific to your application within the context of the entities.
- Originally referred to as “Application logic”.

CONTROLLERS



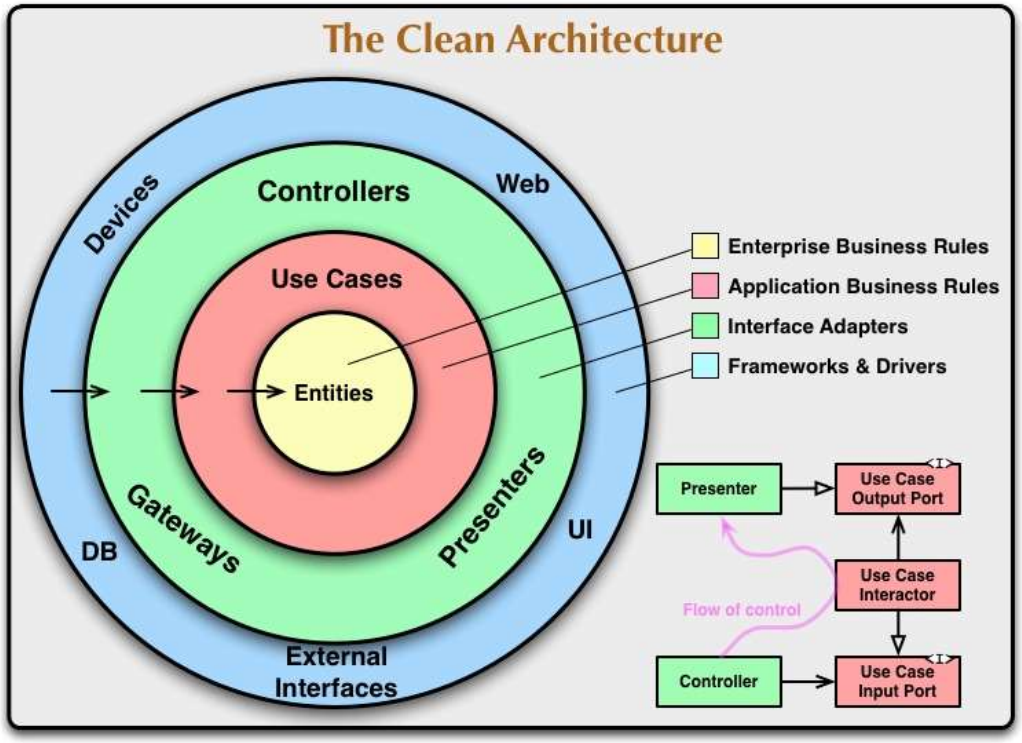
- Translates Entity / Use Case data into representation suitable for persistence, integration or presentation.
- Driven by specific requirements of individual interfaces.
  - Different data and behavior for persistence than for presentation.

FRAMEWORKS AND DRIVERS



- Specific frameworks and technologies for persistence, presentation and integration.

SCREAMING ARCHITECTURE



- Architecture of your product *screams* intent.
- In other words: structure, naming and dependencies are modeled along your problem domain.

So what does the architecture of your application scream? When you look at the top level directory structure, and the source files in the highest level package; do they scream: **Health Care System**, or **Accounting System**, or **Inventory Management System**? Or do they scream: **Rails**, or **Spring/Hibernate**, or **ASP**?

...AND THE QUALITY?

- Because software architecture is driven by stakeholder values / architectural concerns / quality requirements (names are not that important),
- the choices you make as a software architect largely **determine the resulting quality of your software**,
- in the form of limitations or constraints which may be difficult to eliminate later.
- Having proper architecture according to stakeholder values provides good foundation, nothing more, nothing less. No magic beans.

FOUR RULES OF SIMPLE DESIGN

KENT BECK

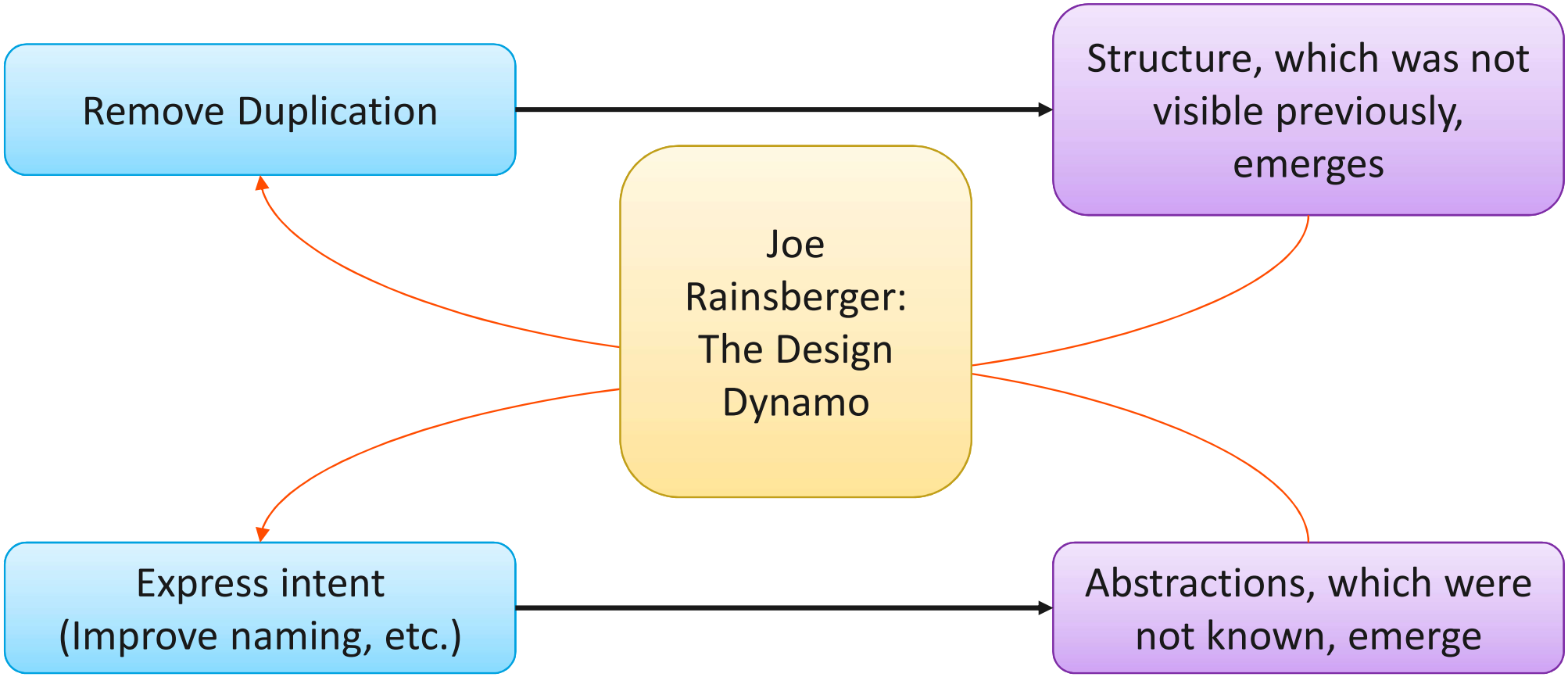
4 RULES OF SIMPLE DESIGN

- **Tests Pass**
  - Tests: Repeatable, Traceable and Explicit
  - Tests Pass not “Unit Tests Pass”
  - Your test suite can always be bigger (coverage).
  - Your test suite can always run faster (frequency).
- **Expresses Intent**
  - Good naming and decomposition.
  - Measurements are indicative and good for pinpointing the biggest culprits (interface cohesion, dependencies, abstraction leaks).
  - SOLID helps this.
  - Similar to Screaming Architecture

4 RULES OF SIMPLE DESIGN (2)

- **Do Not Repeat Yourself** (Lazy is Good ;-)
  - In the meaning of knowledge duplication.
  - Every piece of knowledge should have only one representation.
    - Look back at the Clean Architecture and its separation of concepts.
- **Small**
  - Code / module / service which is no longer used.
  - Are there duplicate abstractions?
  - Abstractions are tricky as they are usually not exact duplicates.
  - Over-extraction.

DESIGN IS (ALSO) A PROCESS



WHAT ELSE?

- SOLID Principles
- **S**ingle Responsibility Principle
- **O**pen (for extension) / Closed (for modification)
- **L**iskov Substitution Principle
- **I**nterface Segregation
- **D**ependency Inversion

DESIGN VS. ARCHITECTURE

1. All architecture is design, but not all design is architecture.
    2. What is architecture for one is design for another.

WHAT ARE THE DIFFICULT QUESTIONS?

- Database?
- Which Messaging Bus?
- Shall we use Relational Database or an Object Database?
- What frameworks are we going to use?
- Which programming languages are we going to use?
- Which ORM are we going to use?
- Are we going to use Azure or Amazon cloud?
- How are we going to deploy clustering in Hadoop for high availability?

WWW.KAHOOT.IT

PIN: **7803157**

ANATOMY OF THE TEAM...

...AND THE ROLE OF THE SOFTWARE ARCHITECT

Organisations which design systems are constrained to produce designs which are copies of the communication structures of these organisations.

—M. CONWAY

(Conway's Rule)

- Conway's Rule basically means that our system is only as good as is the communication in your organization.
- One of the major driving ideas behind Microservices and **in my humble opinion** for all the wrong reasons.
- Different Architectural Flavors:
  - 2-, 3-, N-Tier
  - Distributed / Remote Objects (CORBA, DCOM)
  - Service Oriented architecture
  - Microservices
  - Serverless Architecture (Amazon Lambda, Azure Functions, etc.)

THE POSITION OF SOFTWARE ARCHITECT

- Software Architect is most importantly an **Internal Stakeholder** to the development / scrum / * team.
- Software Architect needs to be respected by Product Owner / Product Manager as such.
- The primary responsibility of the Software Architect is to protect key system qualities determined by (high priority) stakeholder values.
- And yes, sometimes you need to go against the team to achieve it. But preferably, you should not have to.

SUMMARY

OH GOD, AT LAST

- Software Architecture provides answers to difficult questions about a software system.
- Software Architecture is...
  - property,
  - artifact(s),
  - process (starts at the beginning, never stops)
- Software Architecture is driven by stakeholder values.
- Do no resist change. Change is fact of life.
  - You can say, that something is “difficult” or “expensive”. But never “impossible”. There are many people who do not know it and will happily do it, if you don’t.

WHAT ELSE?

- We are still scratching the surface.
- We did not touch...
  - Monitoring and Simulation (“Monitoring Driven Development”)
  - Domain Driven Design
  - How to properly automate validation tests?
  - ...and much more.