

#### Software Architecture and Techniques

#### Evolution Of Software Architecture Over The Last Decades





# Truths (1/2)

All architecture is design but not all design is architecture.

Architecture represents the significant design decisions that shape a system, where **significant** is measured by **cost of change** 

Grady Booch, 2006

# Truths (2/2)

Software development does not have economies of scale.

Development has **diseconomies of scale**.

– Allan Kelly

## **Architecture Definitions**

The fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.

#### — ISO/IEC/IEEE 42010:2022

The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.

#### — The TOGAF Standard, Version 10

## **Old School Architect**

- Separate position with highest status
- Decide how the architecture will be
  - Architects are smart
  - Developers are dump
- Ivory tower syndrome
- Powerpoint architect syndrome
- Think about Conway Law Hierarchy vs Meritocracy

# Architecture Kinds (1/2)

- Design  $\rightarrow$  developer
- Application Architecture  $\rightarrow$  within team
- Solution Architecture  $\rightarrow$  within product
- Enterprise Architecture → whole company every traditional architect wants to be an enterprise architect!

# Architecture Kinds (2/2)



#### History 1960 - 2000

- Structured Programming goto are evil -
- Structured Design Yourdon, DeMarco -
- Structured Analysis & Design SASD, SADT -
- Object-Oriented Approach Booch, Rumbaugh, Jacobson -
- Enterprise Architecture Zachmann -

#### The Enterprise Ontology "



© 1987-2011 John A. Zachman, all rights reserved. Zachman® and Zachman International® are registered trademarks of John A. Zachman To request Permission Use of Copyright, please contact: Zachman.com

### Standards (around 2000)

- TOGAF
- Arc42 and ISAQB
- RUP Inception, Elaboration, Construction, Transition
- Hermes
- IEEE



### UML - 4 + 1 View

Evaluate the views in the context of a modern development project and environment



## **First Findings**

- Architecture shall fulfill customer needs
  - Functional requirements
  - Non-functional requirements
- Dichotomy Analysis and Design
  - Analysis is requirement engineering understand the problem
  - Design is architecture identify a solution
  - Modern approaches killed up-front requirements documents and analysis

## Understand the Problem

- Understand the domain
- Functional requirements
- Non-functional requirements
- User interface
- Process improvements



#### Requirements - **SMART**

- **S** Specific
- M Measurable
- **A** Assignable (who will do it?)
- **R** Realistic
- **T** Time-related (when should it be done?)

Look at SMART requirements in agile world

## Stories - INVEST

- I Independent
- N Negotiable
- V Valuable
- E Estimate-able
- **S** Small
- T Testable

# Backlog - DEEP

- **D** Detailed Appropriately
- E Estimated
- E Emergent
- P Prioritized



## **Backlog Item**

- What is a product backlog item *PBI*?
- Is a product backlog item a story?
- Why do you estimate a PBI?
- How do you know when a PBI is completed?

### Create an Architecture

- Define an architecture
- Validate it
- Verify it
- Document it
- Evolve it





### **DDD** and Event Storming

- Customer Language
- Domain Knowledge
- Workshop and Discussion

- UX Workshop
- Design Thinking

Agile Approach (1/2) Vision – Why? Roadmap – What do we get the next 9 -18 months? Release Planning – Story Map Sprint Backlog – What do we do the next 1-2 weeks? MVP Minimum Viable Product **MMP** Minimum Marketable Product

# Agile Approach (2/2)

- Agile Manifesto Principle 6: The most efficient and effective method of conveying information to and within a development team is **face to face** conversation
- Agile Manifesto Principle 11: The best architectures, requirements, and designs emerge from self-organizing teams

## Agile Impact to Architecture



Value Delivery
Risk of failure (missed needs)

#### Agile Impact On Success PROJECT SUCCESS RATES AGILE VS WATERFAI



Standish Group, Chaos Report, 2018

### Architect and Developers

- Team Work
- Craftsmanship
- Team Dynamics



#### Craftsmanship Approach

- Architect is a domain expert
- Architect is a software craftsmanship
- Architect is a lean leader teacher, coach, mentor
- Architect discuss with stakeholders and C-level representatives

#### **QUALITY CODING**

## **Professional Technology Decisions**

- You are an engineer
- You understand the technology
- You understand your customer needs

Total								
	Energy			Time			Mb	
(c) C	1.00		(c) C	1.00		(c) Pascal	1.00	
(c) Rust	1.03		(c) Rust	1.04		(c) Go	1.05	
(c) C++	1.34		(c) C++	1.56		(c) C	1.17	
(c) Ada	1.70		(c) Ada	1.85		(c) Fortran	1.24	
(v) Java	1.98		(v) Java	1.89		(c) C++	1.34	
(c) Pascal	2.14		(c) Chapel	2.14		(c) Ada	1.47	
(c) Chapel	2.18		(c) Go	2.83		(c) Rust	1.54	
(v) Lisp	2.27		(c) Pascal	3.02		(v) Lisp	1.92	
(c) Ocaml	2.40		(c) Ocaml	3.09		(c) Haskell	2.45	
(c) Fortran	2.52		(v) C#	3.14		(i) PHP	2.57	
(c) Swift	2.79		(v) Lisp	3.40		(c) Swift	2.71	
(c) Haskell	3.10		(c) Haskell	3.55		(i) Python	2.80	
(v) C#	3.14		(c) Swift	4.20		(c) Ocaml	2.82	
(c) Go	3.23		(c) Fortran	4.20		(v) C#	2.85	
(i) Dart	3.83		(v) F#	6.30		(i) Hack	3.34	
(v) F#	4.13		(i) JavaScript	6.52		(v) Racket	3.52	
(i) JavaScript	4.45		(i) Dart	6.67		(i) Ruby	3.97	
(v) Racket	7.91		(v) Racket	11.27		(c) Chapel	4.00	
(i) TypeScript	21.50		(i) Hack	26.99		(v) F#	4.25	
(i) Hack	24.02		(i) PHP	27.64		(i) JavaScript	4.59	
(i) PHP	29.30		(v) Erlang	36.71		(i) TypeScript	4.69	
(v) Erlang	42.23		(i) Jruby	43.44		(v) Java	6.01	
(i) Lua	45.98		(i) TypeScript	46.20		(i) Perl	6.62	
(i) Jruby	46.54		(i) Ruby	59.34		(i) Lua	6.72	
(i) Ruby	69.91		(i) Perl	65.79		(v) Erlang	7.20	
(i) Python	75.88		(i) Python	71.90		(i) Dart	8.64	
(i) Perl	79.58		(i) Lua	82.91		(i) Jruby	19.84	

**m** .

## Self-Organizing Teams

Setting Overall Direction	Manag	gement		
Designing the Team and Its Context	Respon	sibilities		
Monitoring and Managing Work Processes		R	Team esponsibilitie	es
Executing the Task				
	Manager-Led Team	Self-Managing (Self-Organizing) Team	Self-Designing Team	Self-Governing Team



## Exercises (1/3)

- Create an UML diagram of your application
  - Why, when and how do you do it,
- Write a functional requirement SMART -
  - How do you insure your requirement is testable?
  - Look at SPIDR for stories (Mike Cohn Video)
- Write a non-functional requirement
  - How do you insure your requirement is testable?
- Reflect changes introduced with agile and lean
  - Quality, speed, costs, success

## Exercises (2/3)

- Read article "Agile Architecture in the Digital Age"
  - Read the ideas, you do not need to memorize the concepts
- Code examples of students
- Write unit tests, execute them in IDE, improve code coverage Java: JUnit 5, Mockito, AssertJ
- Execute SonarLint on the fly on your source code

## Exercises (3/3)

- Explore the refactoring features of the IDE
  - IDEA configure code style, copyright, etc.
  - IDEA Analyze Menu
    - Inspect Code, Clean Code
    - Find Usage, Find Declaration
  - Refactor Menu
    - Refactor (more than 10 operations)
  - Run IDEA "Analyze/Inspect Code...", SpotBugs, SonarLint
  - Use Git integrated client commit, amend, push,
  - Use local history feature