Software Architecture and Techniques

Validate Quality Attributes

of

Software Architecture

Marcel Baumann, tangly llc

Truths (1/3)

- Architecture is requirements + -ility attributes fulfillment
- Do user and customer define requirements?
- Do stakeholders define -ility attributes (*see ISO/IEC 25010*)?
- Remember validation and verification
  - Build the right product
  - Build the product right

Truths (2/3)

- All solutions should be source code based and under version control – *git* -

- All solutions should be integrated in the CI/CD/CD pipeline

- Avoid metric driven development – *this is an anti-pattern*

Truths (3/3)

- ATAM is dead – *too slow, too expensive* – architecture driven design is also dead

- Manual reviews are obsolete
  - Pull request reviews are slow, expensive and therefore should be replaced by better approaches
  - Agile teams shall be co-located and do pair work

    or virtually integrated using high-quality communication tools and no time difference

Non-Functional Requirements

- Context

- Why

- Mandatory Value (Minimum) – *Below minimal value means you have failed*

- Optimal Value (Optimum)

- Maximal Value (Outstanding) – *Above maximal value means you have wasted resources*

ISO/IEC 25010

Systems and software Quality Requirements and Evaluation (SQuaRE)

SOFTWARE PRODUCT QUALITY

Functional Suitability	Performance Efficiency	Compatibility	Usability	Reliability	Security	Maintainability	Portability
• Functional Completeness • Functional Correctness • Functional Appropriateness	• Time Behaviour • Resource Utilization • Capacity	• Co-existence • Interoperability	• Appropriateness Recognizability • Learnability • Operability • User Error Protection • User Interface Aesthetics • Accessibility	• Maturity • Availability • Fault Tolerance • Recoverability	• Confidentiality • Integrity • Non-repudiation • Authenticity • Accountability	• Modularity • Reusability • Analysability • Modifiability • Testability	• Adaptability • Installability • Replaceability

iso25000.com

Some -ility Attributes

Accessibility, accountability, accuracy, adaptability, administrability, affordability, agility, **auditability**, autonomy, availability, compatibility, composability, configurability, **correctness**, credibility, customizability, **debugability**, degradability, determinability, demonstrability, dependability, **deployability**, discoverability, distributability, durability, effectiveness, efficiency, **usability**, extensibility, failure transparency, fault tolerance, fidelity, flexibility, inspectability, installability, integrity, interoperability, learnability, **maintainability**, manageability, mobility, modifiability, modularity, operability, orthogonality, **portability**, precision, predictability, process capabilities, producibility, provability, recoverability, relevance, **reliability**, repeatability, reproducibility, resilience, responsiveness, reusability, robustness, **safety**, scalability, seamlessness, self-sustainability, serviceability, sustainability, tailorability, **testability**, timeliness, **traceability**

Auditing Criteria

| Aspect	Sub-Aspect	Property	Description
Product Quality	Maintainability	Maintainability Metrics	Quality metrics on the code level
	Readability	Coding style	Consistency of coding accross the whole code base
		Code Organization	How easy code components and types of code can be identified.
		Naming	Wether Naming is consistent and helps in understanding
		Comments	Whether comments are helpful, uptodate and of professional style
		Dead code	Whether unused code makes understanding unnecessary difficult
	Platform Choice	Programming Language and Platform	Whether language/platform choice are common or exotic
		Libraries and Frameworks	Whether important libraries/frameworks are common or exotic
Documentation	Architecture	Quality	The structure and style of the document and whether it is uptodate
		Completeness	Whether the architecture document contains the relevant information
	Functionality	Quality	The structure and style of the document and whether it is uptodate
		Completeness	Whether the document contains the relevant information
	Operation	Quality	The structure and style of the document and whether it is uptodate
		Completeness	Whether the document contains the relevant information
	Test Scripts	Quality	Quality and Readability of testcode
		Coverage	Whether all parts of the system have test (manual and/or automated)
Publishability	Sensitive Information	PII	Personal Identifiable information
		Proprietary details	Sensitive information that is not personal
	License Compliancy	Offending Licenses	Licenses that prohibit publication
		Closed source dependencies	Code that is not available
	Completeness	Build Completeness	Whether all artifacts are published that are needed to build a functional system.
	Publication Format	Natual language	Whether the Natural language choice for code identifiers, comments and documentation limits the pool of reviewers. Especially also wether the codebase uses multiple different languages.
		Common Format	Whether the code review should not require tools that are expensive or otherwise difficult to obtain and operate.

Fitness Functions (1/2)

- Fitness functions are the unit tests for non-functional requirements

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

We can also think about the systemwide fitness function as a collection of fitness functions with each function corresponding to one or more dimensions of the architecture.
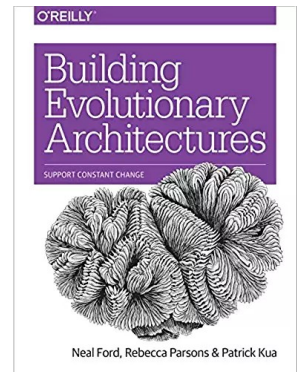
Fitness Functions (2/2)

Double loop architecture is a process that you can use to ensure that your architecture continues to satisfy the business needs of your product.
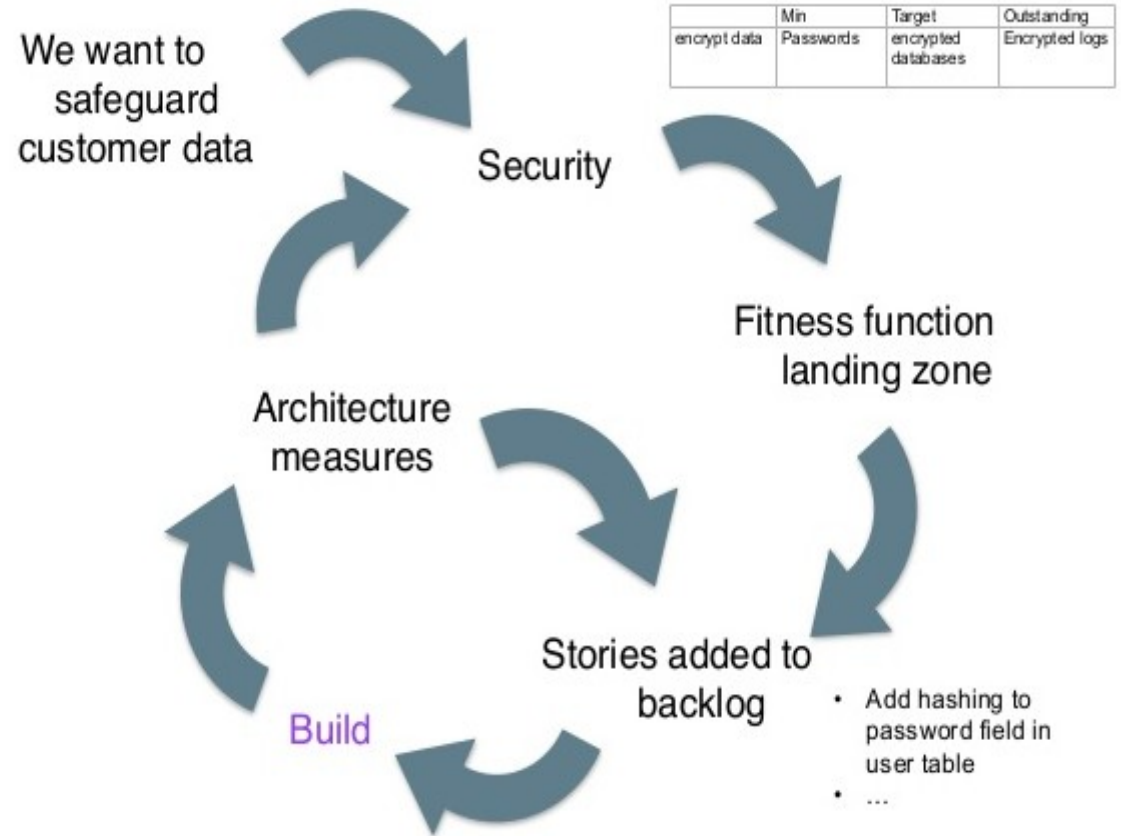
Example Fitness Functions

- Static code analysis

- Unit test frameworks

- Penetration testing tools

- Load testing tools

- Monitoring tools

- Logging tools

```
describe "Resiliency" do
      describe "New Deployment" do
            it "has less than 1% error rate for new deployment" do
                  expect(new_deployment.get_error_rate()).to < .01
            end
      end
      describe "Network Latency" do
            it "has less than 5% error rate even if there is network latency" do
                  expect(network_tests.get_error_rate()).to < .05
            end
            it "completes a transaction under 10 seconds even if
              there is network latency" do
                  expect(network_tests.get_transaction_time()).to < 10
            end
      end
end
```

Assumptions

Architecture, like business capability and infrastructure, can be **expressed in code** through the use of appropriate fitness functions.

Fitness functions are **code** and can be executed as part of CI/CD pipeline or part of the monitoring infrastructure.

Fitness Functions

- Fitness functions are part of the continuous integration CI pipeline

- Often Realtime

- Quality Gate Function
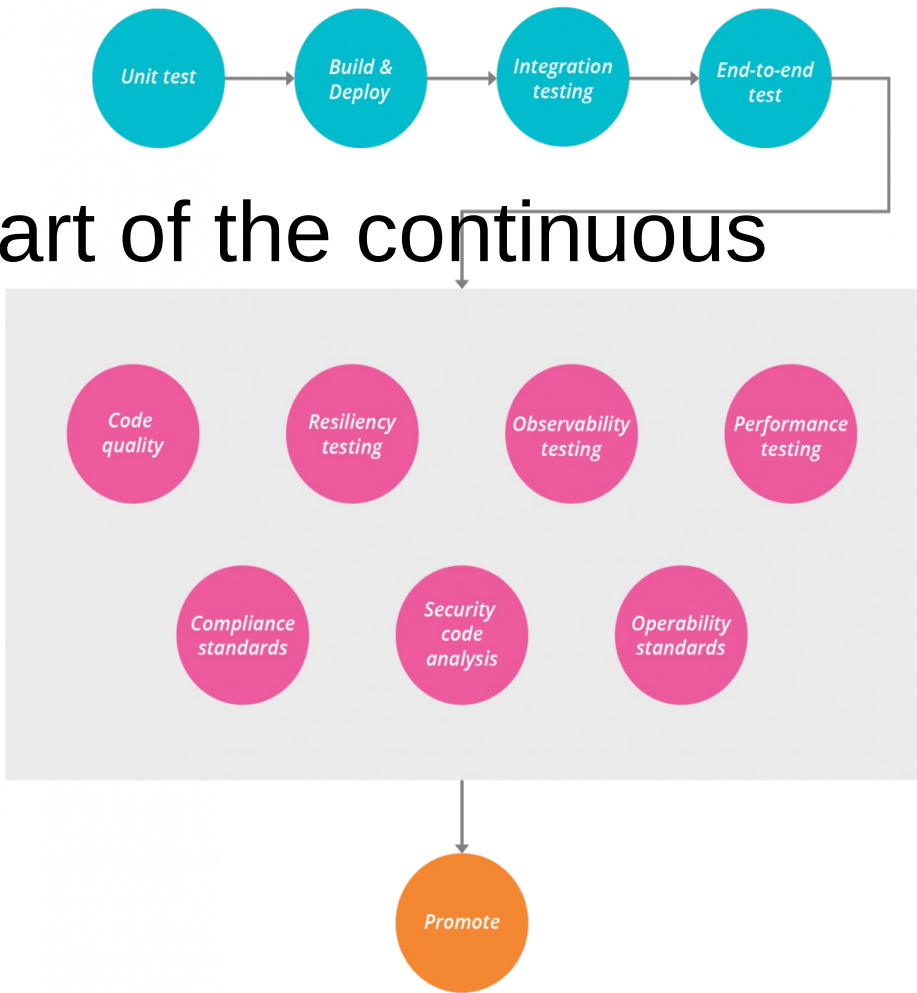
Combining Fitness Functions

- Atomic + triggered
  - ArchUnit rules

- Holistic + triggered
  - Combined Security and Scalability Functions

- Atomic + continual
  - Test REST endpoints verbs and error messages

- Holistic + continual
  - Test resilience when cloud latency changes through infiltration (Netflix)

Functions Examples

- Your code quality must be above 90% to be promoted to the next stage – *Quality Gate in SonarQube*

- UAT versioning must not deviate more than two versions from production

- No secrets may be committed in plain text - *OSWAP*

- You must always have a security testing stage

- You must never deploy with another application's service account

- You must always have two approvers before production

Fitness Functions

- Fitness functions are also part of the production environment

    - Mean Time between Failure

    - Maximum Time to Recover

    - Response Time

    - Latency in your network

    - Resource usage

Code Quality

- Modifiability

- Manageability

- Adaptability

- Legibility

DevOps DORA Metrics
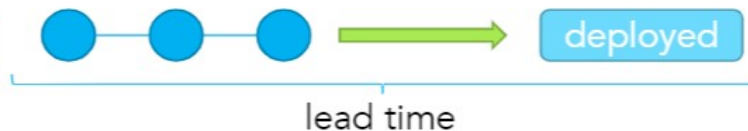

Deployment Frequency
deployments per day
>1/day — Throughput

Lead time for change
hours
<1h — Velocity
lead time
deployed

Time to Restore
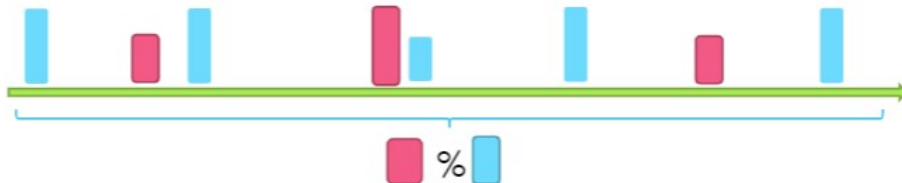*hours*
<1h — Velocity
time to restore | time to restore

Change Failure Rate
Percentage of failed deployments
0-15% — Risk
Elite
%

Code Quality

- Modifiability

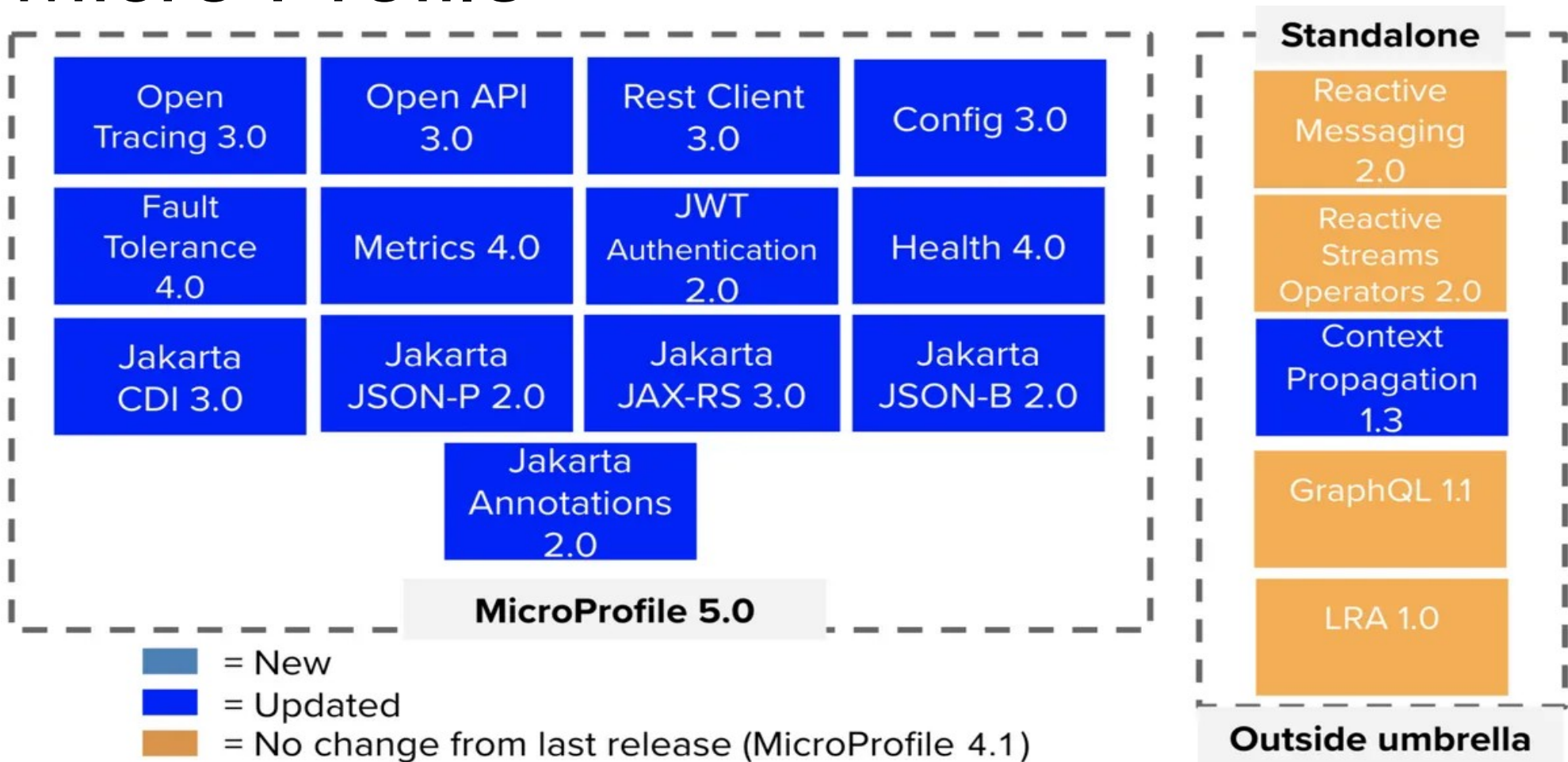- Manageability

- Adaptability

- Legibility

Resilience and Operability

- Stability
- Resiliency
- Availability
- Recoverability

Performance and Security

- Scalability

- Stability

- Response time

- Security

Micro Profile

Open Tracing 3.0	Open API 3.0	Rest Client 3.0	Config 3.0
Fault Tolerance 4.0	Metrics 4.0	JWT Authentication 2.0	Health 4.0
Jakarta CDI 3.0	Jakarta JSON-P 2.0	Jakarta JAX-RS 3.0	Jakarta JSON-B 2.0
	Jakarta Annotations 2.0		

MicroProfile 5.0

Standalone

Reactive Messaging 2.0

Reactive Streams Operators 2.0

Context Propagation 1.3

GraphQL 1.1

LRA 1.0

Outside umbrella

= New
= Updated
= No change from last release (MicroProfile 4.1)

Start Simply with Monitoring

- Use a logging framework e.g. log4j2

- Use a simple monitoring tool e.g. VisualVM

- Learn and improve

  - e.g. application performance management APM with Scouter or Apache SkyWalking.

Exercises (1/2)

- Discuss your architecture quality
  - Which criteria to measure it? *Prove it with facts!*
    - *e.g. use VisualVM as a simple tool to measure Java applications*
  - Should you improve it?
  - How can you improve it? What should change in your team?
- Select architecture questions and discuss how you solved them in your application
  - e.g. logging, creation of objects, persistence, error handling

Exercises (1/2)

- Read article "Modern Java EE Design Patterns"

- Select -ility criteria, define associated fitness functions and show how to implement them

- Reflect how ArchUnit can implement a subset of fitness functions

- Workshop preparation

- Coding Dojos

Exercises (2/2)

- Check your project – *as described during first week of lecture –*
  - Refactoring project and presentation (history in git)
  - Architecture portfolio and participation in exercise coaching – *e.g. pattern example or a solution to an architecture dimension such as logging in your project -*
  - Test automation (TDD, ATDD, CI/CD) concepts and examples