IN4315: Software Architecture

Architecting for Quality

3

Prof. Diomidis Spinellis

http://www.spinellis.gr/
D.Spinellis@tudelft.nl

Based on material by Prof. Cesare Pautasso
http://www.pautasso.info/
cesare.pautasso@usi.ch

Contents

Quality

Types of Requirements

Functional	Shall do	It works!
Non-Functional	Shall be	It works/evolves well

Functional

- Correctness
- Completeness
- Compliance (e.g., Ethical Implications)

Non-Functional ...

- Internal vs. External
- Static vs. Dynamic

Internal vs. External

External qualities concern the fitness for purpose of the software product, whether it satisfies stakeholder concerns. They are affected by the deployment environment.

Internal qualities describe the developer's perception of the state of the software project and change during the design and development process.

Static vs. Dynamic

Static qualities concern structural properties of the system that can be assessed before it is deployed in production

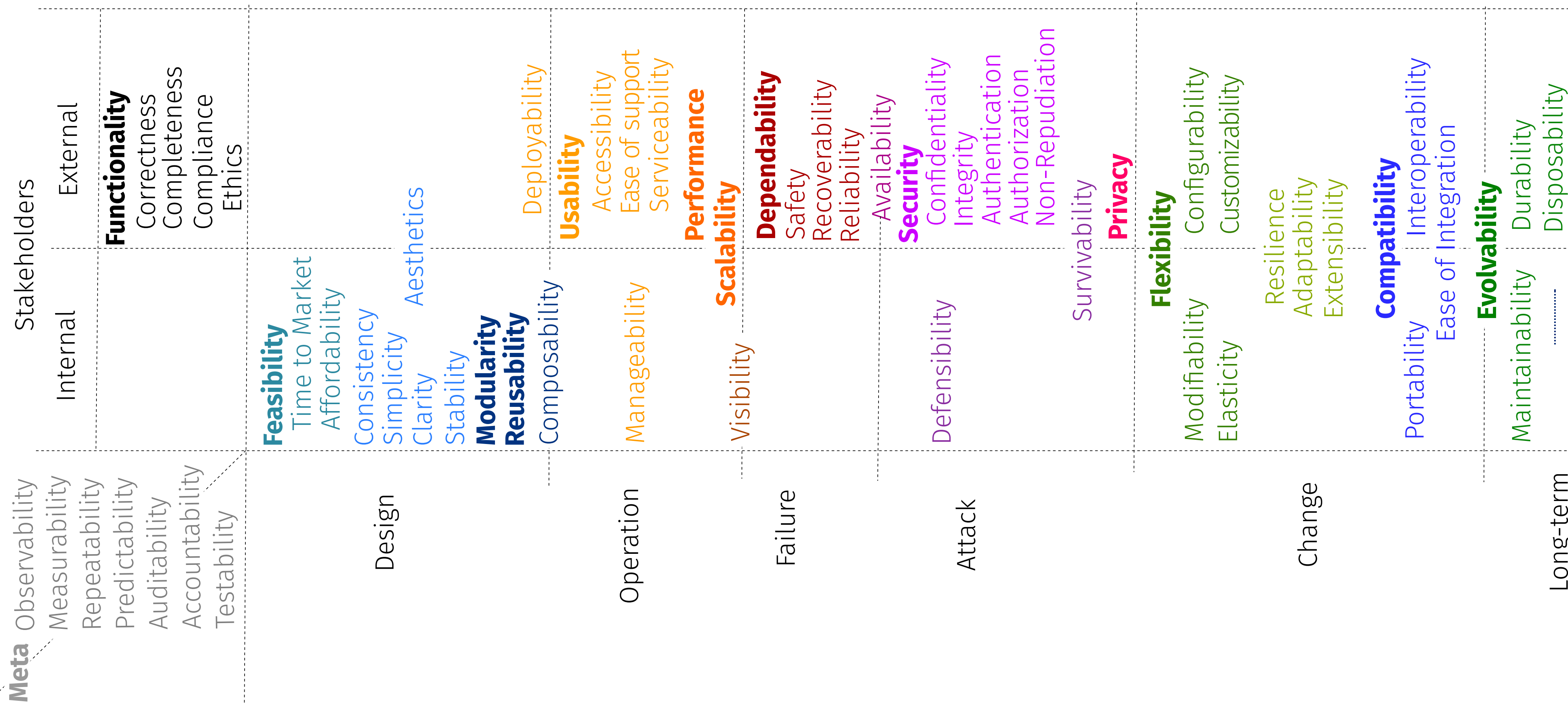
Dynamic qualities describe the system's behavior:

- during normal operation
- in the presence of failures
- under attack
- responding to change
- in the long term

Meta-Qualities

- Observability
- Measurability
- Repeatability (Jitter)
- Predictability
- Auditability
- Accountability
- Testability

Quality Attributes



Quality Attributes

Meta
Observability
Measurability
Repeatability
Predictability
Auditability
Accountability
Testability

Stakeholders
Internal
External

Functionality
Correctness
Completeness
Compliance
Ethics

Feasibility
Time to Market
Affordability

Consistency

Design

Quality Attributes

	Internal	External
Measurability Repeatability Predictability Auditability Accountability Testability		**Functionality** Correctness Completeness Compliance Ethics
Design	**Feasibility** Time to Market Affordability Consistency Simplicity Clarity Stability **Modularity** **Reusability** Composability	Aesthetics Deployability

Quality Attributes

Design

Consistency
Simplicity
Clarity
Aesthetics
Stability
Modularity
Reusability
Composability

Deployability

Operation

Manageability

Usability
Accessibility
Ease of suppo
Serviceability
Performance
Scalability

Visibility

Failure

Dependability
Safety
Recoverability
Reliability

Quality Attributes

14:03

Operation

Manageability

Accessibility
Ease of suppo
Serviceabilit

Performance

Scalability

Visibility

Failure

Dependability
Safety
Recoverability
Reliability
Availability

Attack

Defensibility

Security
Confidentialit
Integrity
Authenticatio
Authorization
Non-Repudiat

powered by ASQ

Quality Attribut

Non-Repudiat

Survivability

Privacy

Change

Flexibility

Modifiability

Configurability

Elasticity

Customizability

Resilience

Adaptability

Extensibility

Compatibility

Portability

Interoperability

Ease of Integration

Evolvability

14:03

Design

powered by ASQ

Design Qualities

- Feasibility
- Consistency
- Simplicity
- Clarity
- Aesthetics
- Stability
- Modularity
- Reusability
- Composability
- eployability

Feasibility

What's the likelihood of success for your new project?

- Affordability
- Time to Market

Affordability

Are there enough resources to complete the project?

- Money
  - Hardware
  - People (Competent, Motivated)
- Time
- Slack

Slack

Are there enough free resources (just in case)?

- Deal with unexpected events
- Breathing space to recharge
- Planning, backlog grooming
- Keep track of the big picture
- Reflect and refactor
- Pay back technical debt
- Learn and experiment

Time to Market

How soon can we start learning from our users?

Slow	Fast
Build from scratch	Reuse and assemble
Perfect product	Minimum viable product (MVP)
Design by committee	Dedicated designer

Modularity

Is there a structural decomposition of the architecture?

Prerequisite for: Code Reuse, Separate Compilation, Incremental Build, Distributed Deployment, Separation of Concerns, Dependency Management, Parallel Development in Larger Teams

Programming Languages **with** modules:

Ada, Algol, COBOL, Dart, Erlang, Fortran, Go, Haskell, Java, Modula, Oberon, Objective-C, Perl, Python, Ruby

Programming languages **without** modules:

C, C++, JavaScript

Modularisation is a design issue, not a language issue (David Parnas)

Reusability

Can we use this software many times for different purposes?

- Reuse Mechanism: Fork (duplication) vs. Reference (dependencies)
- Origin: Internal vs. External (Not Invented Here Syndrome)
- Scope: General-purpose vs. Domain-specific
- Pre-requisites for reuse: trusted "quality" components, standardized and documented interfaces, marketplaces

It is often easier to write an incorrect program than to understand how to reuse a correct one (Will Tracz, 1987)

Design Consistency

What's the design's conceptual integrity and coherence?

Understanding a part helps to understand the whole

Avoid unexpected surprises (POLA):

- Pick a naming convention
- Follow the architectural style constraints
- Document architectural decisions

Know the rules (and when to break them)

It is better to have a system reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas (Fred Brooks, 1995)

Simplicity

What's the complexity of the design?

- A simple solution for a complex problem
- One general solution vs. many specific solutions:
  - Lack of duplication (DRY)
  - Minimal variability
- Conciseness
- Resist changes that compromise simplicity
- Refactor to simplify

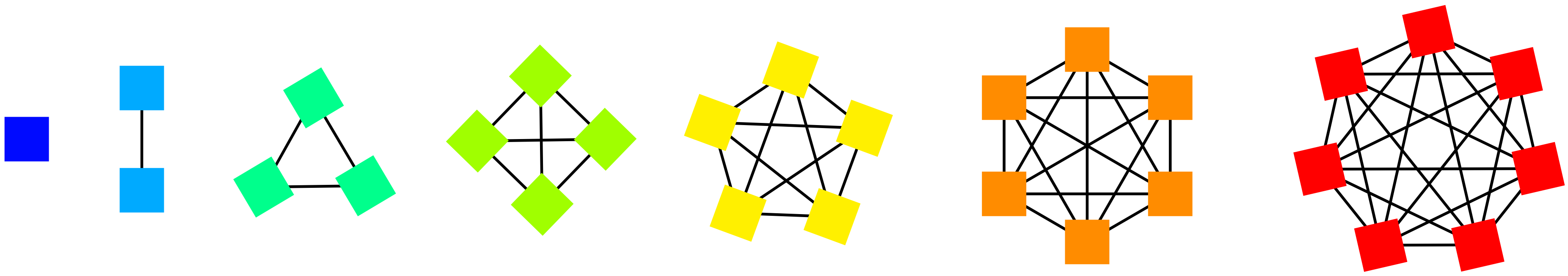
As simple as possible, but not simpler (Albert Einstein)

Done

Complexity

What is the primary source of complexity?

- The number of components of the architecture
- The amount of connections between the components



Clarity

Is the design easy to understand?

A clear architecture distills the most essential aspects into simple primitive elements that can be combined to solve the important problems of the system

Freedom from ambiguity and irrelevant details

Definitive, precise, explicit and undisputed decisions

Opposite: Clutter, Confusion, Obscurity

Stability

How likely to change is your design?

Unstable	Stable
Prototype	Product
Implementation	Interface
Depends on many components	Many components depend on it
Likely to break clients	Platform to build upon
Experimental spike, throw-away code	Worthy of further investment: building, testing, documenting

Composability

How easy is it to assemble the architecture from its constituent parts?

- Assuming all components are ready, putting them together is fast, cheap and easy
- Cost(Composition) < Cost(Components)
- Components can be easily recomposed in different ways

Deployability

How difficult is it to deploy the system in production?

Hard	Easy
Manual Release	Automated Release
Scheduled Updates	Continuous Updates
Unplanned Downtime	Planned or No Downtime
Wait for Dependencies	No synchronization
Changes cannot be undone	Rollback Possible

Normal Operation

Normal Operation

- Performance
- Scalability
- Capacity
- Usability
- Ease of Support
- Serviceability
- Visibility

Performance

How timely are the external interactions of the system?

- Latency
  - Communication/Computation Delay
  - User-Perceived: First Response vs. Completion Time
- Throughput
  - Computation: Number of Requests/Time
  - Communication: Limited by Bandwidth (Data/Time)

Scalability

Is the performance guaranteed with an increasing workload?

- Architecture designed for growth:
  - client requests (throughput)
  - number of users (concurrency)
  - amount of data (input/output)
  - number of nodes (network size)
  - number of software components (system size)

  by taking advantage of additional resources
- Scalability is limited by the maximum **capacity** of the system
- Software systems are expected to handle workload variations of 3-10 orders of magnitude over short/large periods

Capacity

How much work can the system perform?

- **Capacity:** Maximum achievable throughput without violating latency requirements
- **Utilization**: Percentage of time a system is busy
- **Saturation**: Full utilization, no spare capacity
- **Overload**: Beyond saturation, performance degradation, instability
- Ensure that there is always some spare capacity

Done

Measuring Normal Operation Qualities

Results are displayed [ ] after users submit their input

The system can process messages sent in [ ]

After [ ] of initial training, users are already productive

Last Friday the workload reached [ ]

1 second

1M concurrent clients

1000 requests/second

1 hour

Usability

Is the user interface intuitive and convenient to use?

- Learnability (first time users)
- Memorability (returning users)
- Efficiency (expert users)
- Satisfaction (all users)
- Accessibility
- Internationalization

Ease of Support

Can users be effectively helped in case of problems?

Hard	Easy
Cryptic Error Messages	Self-Correcting Errors
Heisen-bugs	Reproducible Bugs
Unknown Configuration	Remotely Visible Configuration
Configuration	No configuration
No Error Logs	Stack Traces in Debug Logs
User in the Loop	Remote Screen; telemetry

Serviceability

How convenient is the ordinary maintenance of the system?

Hard	Easy
Complete Operational Stop	Service Running System
Reboot to upgrade	Transparent upgrade
Install Wizard	Unattended Installation Script
Restart to apply configuration change	Hot/live configuration
Manual Bug Reports	Automatic Crash Report

Visibility

Is it possible to monitor runtime events and interactions?

To which extent the system behavior and internal state can be observed during operation?

Are there logs to debug, detect errors or audit the system in production?

Is the system self-aware?

Process Visibility: can the progress of the project be measured and tracked?

We see in order to move; we move in order to see. (William Gibson)

Failure Mode

Dependability Qualities

- Availability
- Reliability
- Recoverability
- Safety
- Security

Reliability

How long can the system keep running?

- MTBF - Mean Time Between Failures
- MTTF - Mean Time To Failure

Recoverability

How long does it take to repair the system?

- MTTR - Mean Time to Recovery
- MTTR - Mean Time to Repair
- MTTR - Mean Time to Respond

Availability

How likely is it that the system is functioning correctly?

Availability and Reliability

- Availability = MTTF / (MTTF + MTTR)

Availability and Downtime

- Availability = $(T_{total} - T_{down}) / T_{total}$

Availability	Downtime (1 Year)
99%	3.65 days
99.9%	8.76 hours
99.99%	53 minutes
99.999%	5.26 minutes
99.9999%	31.5 seconds

Done

...asuring Availability and Reliability

The service level agreement states up to ______ downtime per ______, an availability of ______

After we call support, they need to be there within ______

Rebooting the server takes ______

The uptime of our oldest server has reached ______

1 month | 30 minutes | 5 seconds | 1 hour | 4 years | 99.861%

Robust

Is damage prevented during **erroneous use** outside the operating range?

Safe

Is damage prevented during use within the operating range?

Secure

Is damage prevented during **intentional/hostile use** outside the operating range?


Under Attack
powered by
ASQ

Security

- Authentication

  How to confirm the user's identity?

- Authorization

  How to selectively restrict access to the system?

- Confidentiality

  How to avoid unauthorized information disclosure?

- Integrity

  How to protect data from tampering?

- Availability

  How to withstand denial of service attacks?

Defensibility

Is the system protected from attacks?

Survivability

Does the system survive the mission?

Privacy

How to keep personal information secret?

Privacy	Good	Poor
Default	Opt-in	Opt-out
Purpose	Specific, explicit	Generic, unknown
Tracking	None	Third-party Fingerprinting
Personal identification	Data anonymization	Data re-identification
Retention	Delete after use	Forever
Breach	Prompt Notification	Silent

Change

Change Qualities

What changes are expected in the future?

No Change: put it in hardware

Software is expected to change

Versioning

Flexibility

- Configurability
- Customizability
- Modifiability
- Extensibility
- Resilience
- Adaptability
- Elasticity

Configurability

Can architectural decisions be delayed until after deployment?

- Component Activation, Instantiation, Placement, Binding
- Resource Allocation
- Feature Toggle

Poor	Good	Better
Undocumented configuration options	Documented configuration options	Sensible defaults provided
Hard-coded parameters (rebuild to change)	Startup parameters (restart to change)	Live parameters (instant change)

Customizability

Can the architecture be specialized to address the needs of individual customers?

- One size Fits All
- Product Line
- White Labeling
- UI Theming, Skin
- Configurability, Composability

Change Duration

- Temporary: **Resilience**

Can the architecture return to the original design after the change is reverted?

- Permanent: **Adaptability**

Can the architecture evolve to adapt to the changed requirements?

Adapt to Changing Requirements

- New Feature: **Extensibility**

  Can functionality be added to the system?

- Existing Feature: **Modifiability**

  Can already implemented functionality be changed?

  Can functionality be removed from the system?

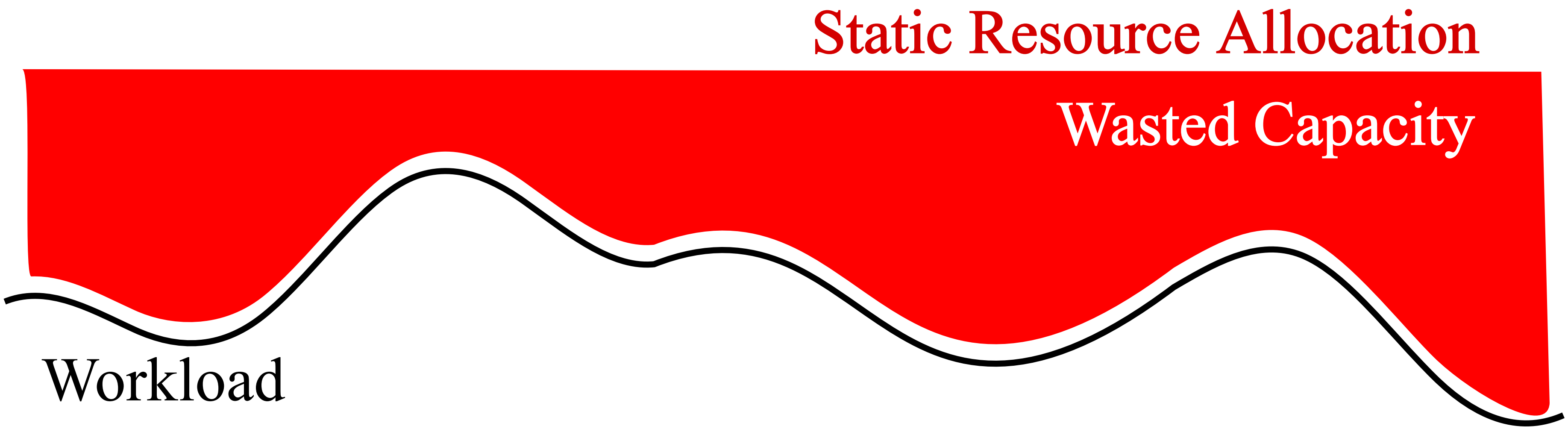
Elasticity

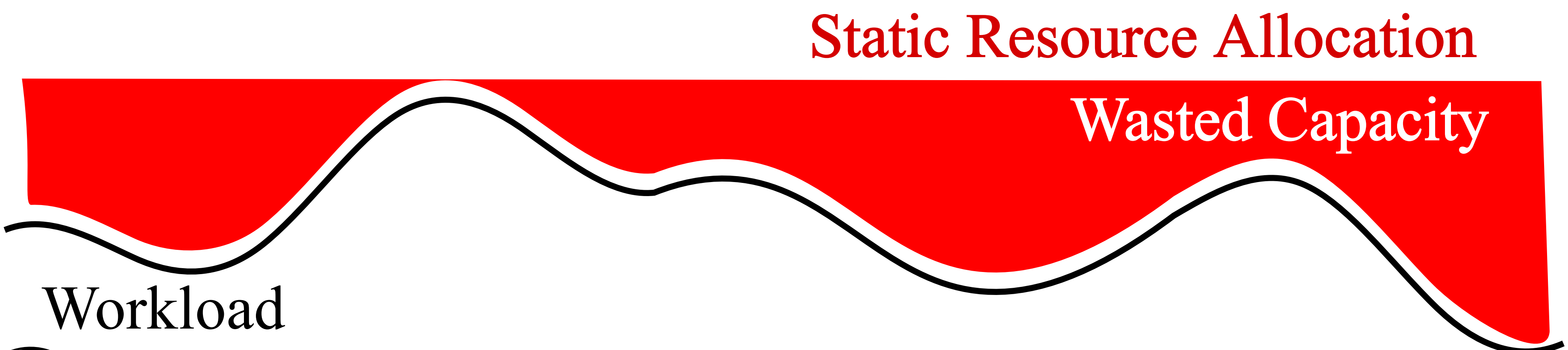
Can workload changes be absorbed by dynamically re-allocating resources?

- Assumption: Scalability + Pay as you go
- Cost(SLA Violation) >> Cost(Extra Resource)
- Example: Cloud Computing

Elasticity

Can workload changes be absorbed by dynamically re-allocating resources?

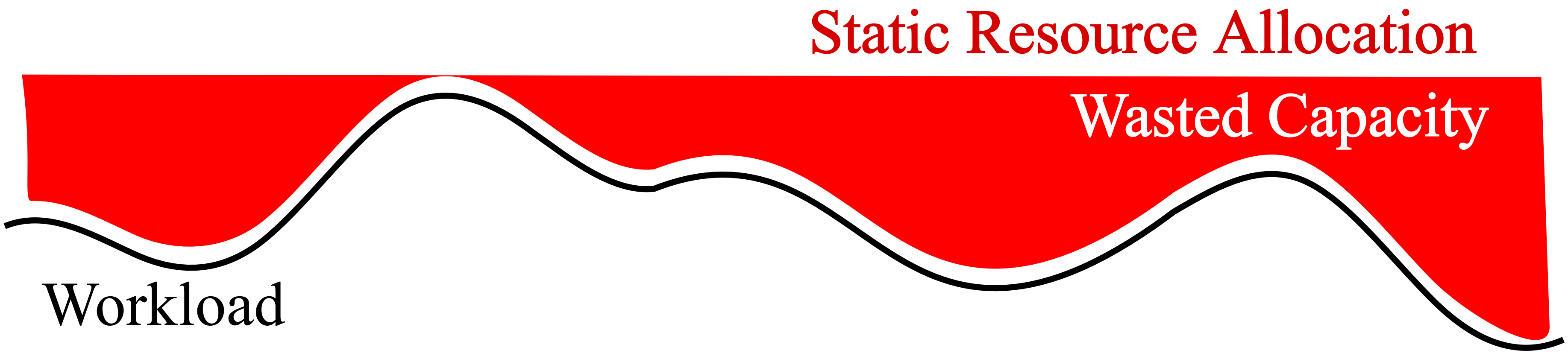


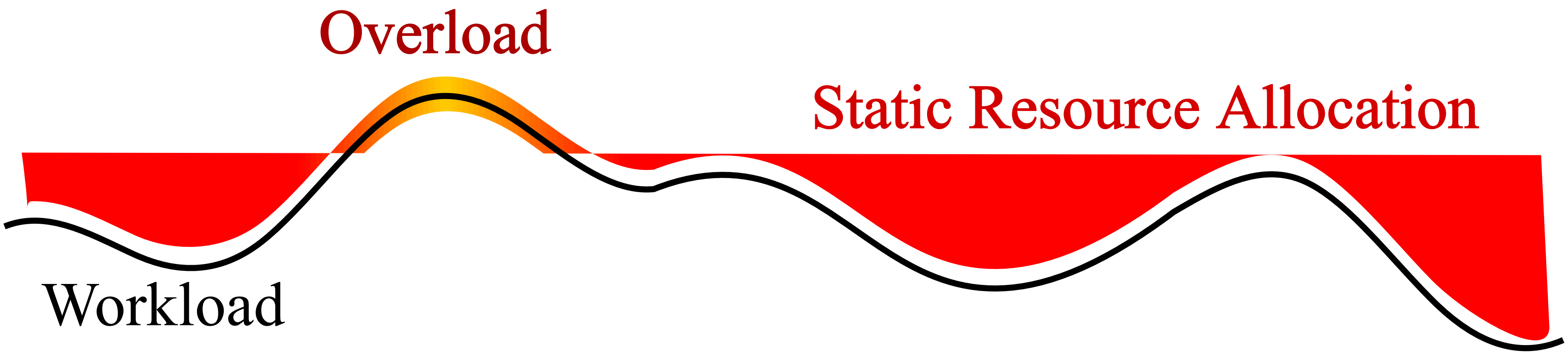


14:04

Elasticity

Can workload changes be absorbed by dynamically re-allocating resources?

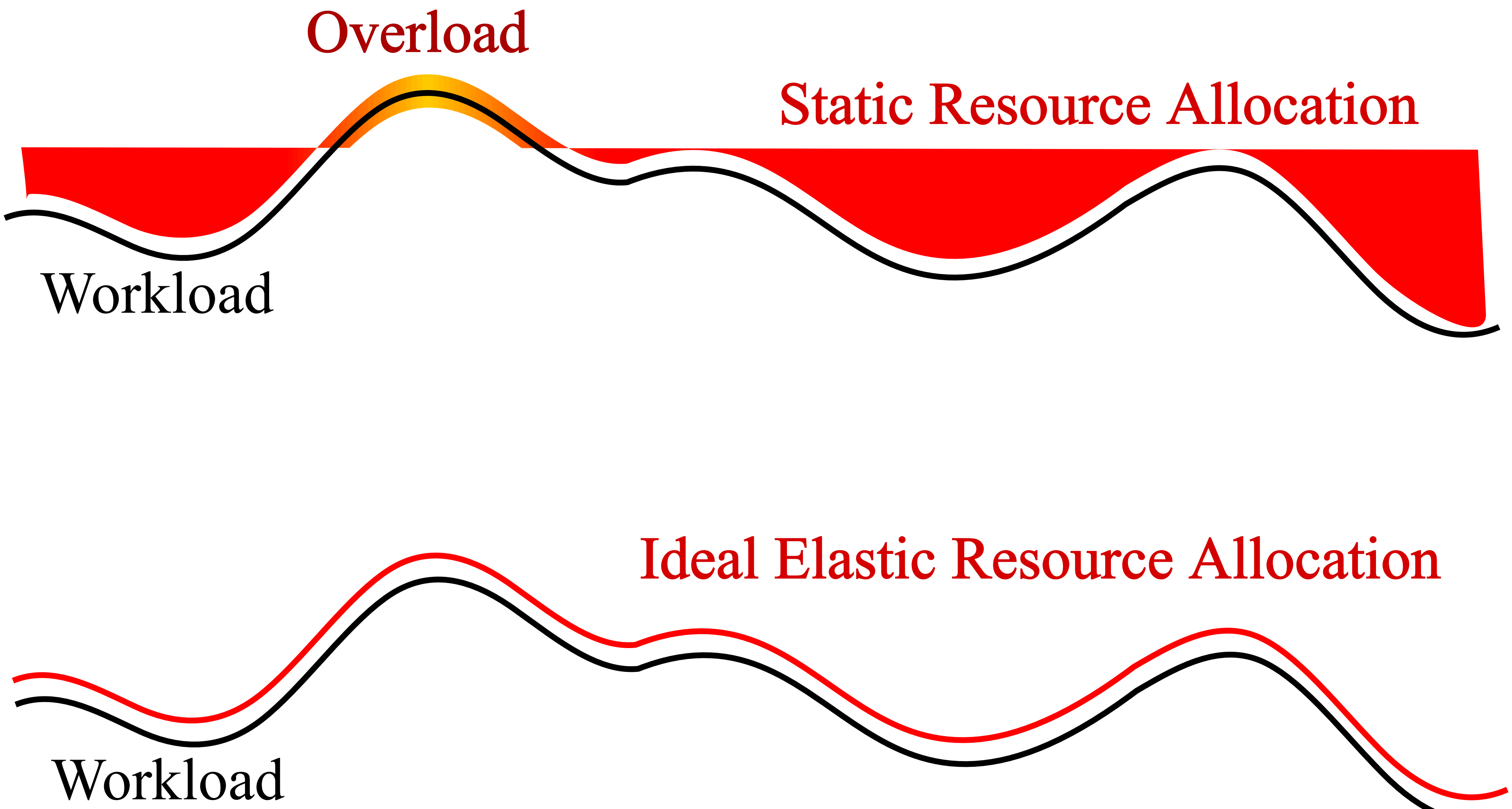




powered by ASQ

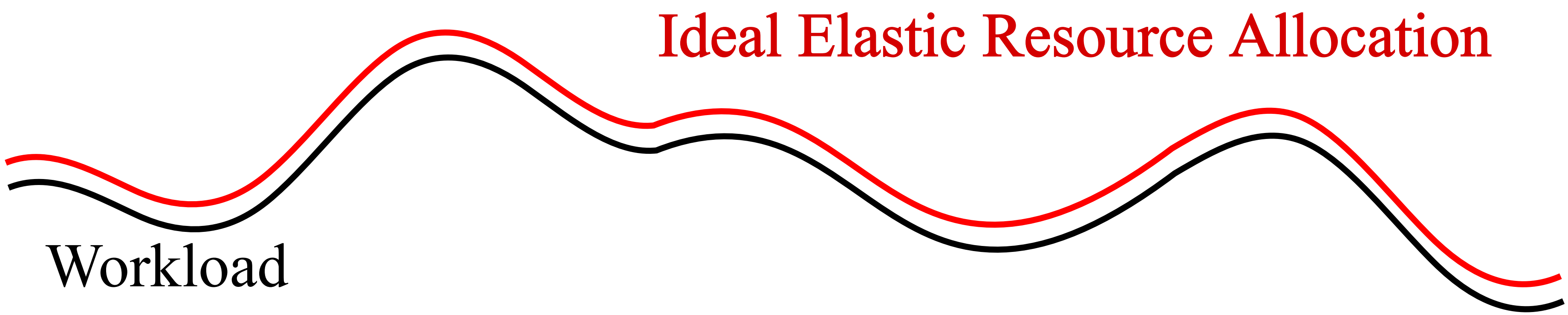
Elasticity

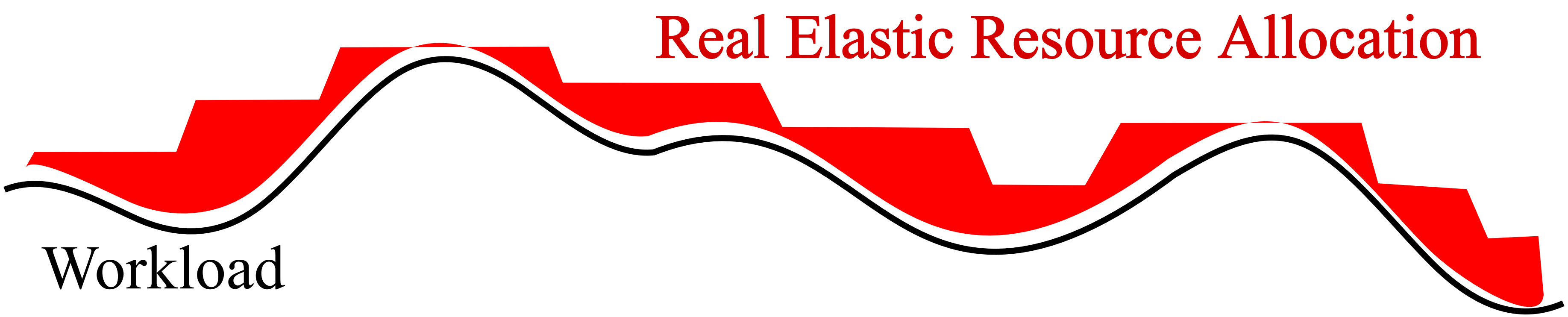
Can workload changes be absorbed by dynamically re-allocating resources?



Elasticity

Can workload changes be absorbed by dynamically re-allocating resources?





Compatibility

Does X work together with Y?

- Interfaces
- Protocols and Data Formats (Interoperability)
- Platforms (Portability)
- Source vs. Binary
- Semantic Versioning (Backwards and Forwards Compatibility)

Portability

Can the software run on multiple execution platforms without modification?

- Write Once, Compile/Run/Test Anywhere
- Cost(porting) << Cost(rewriting)
- Platform-Independent vs. Native Code
- Deployment: Universal Binaries
- Runtime: OS Layer, Virtual Machine Layer, Hardware Abstraction Layer

Interoperability

Can two systems exchange information to successfully interact?

- Abstraction Levels:
  - Payload Syntax
  - Message Semantics
  - Protocols/Conversations
- Content Type Negotiation
- Standardization
- Mediation

Ease of Integration

How expensive is it to integrate our system with others?

Expensive	Easy
Hub and Spoke (2 systems)	Point to Point (2 systems)
Point to Point (N systems)	Hub and Spoke (N systems)
No API	Standard Interface
Custom Binary Data	Standard Text, XML, JSON Data
Air gap	No Firewall
Batched, periodic	Continuous, real-time

Long Term

Long Term Qualities

- Durability
- Maintainability
- Sustainability

Durability

How permanent is the data?

- Persistence Layer (DB, Container, OS)
- Checkpoint and Restore
- Backup and Disaster Recovery
- Long-term Digital Preservation

Maintainability

How to deal with software entropy?

- Change is inevitable
- Keep the quality level over time
- Adaptive, perfective, corrective, preventive maintenance
- Re-engineer, reverse engineer or retire legacy systems

If you never kill anything, you will live among zombies (Gregor Hohpe, 2015)

Done

Maintainability

Adaptive	Perfective	Corrective	Preventive

- Fail Over to Backup Data Center
- Bug Fix
- Year 2038 Time Overflow
- New Feature
- Comply with New Law
- Refactor
- Write Documentation
- Upgrade Dependencies
- Optimize Performance

Types of Maintenance

Adaptive	Deal with external "evolutionary" pressure (avoid quality gets worse over time)
Perfective	Improve external qualities
Corrective	Remove defects (ensure acceptable, good enough quality)
Preventive	Improve internal qualities

Sustainability

- Technical

  How to avoid your software becomes obsolete in the long term?

- Economic

  How to ensure your software development organization does not go bankrupt in the long term?

- Growth

  How to bootstrap the growth of your startup?

References

- George Fairbanks, Just Enough Software Architecture: A Risk Driven Approach, M&B 2010
- Douglas McIlroy, Mass produced software components, NATO Software Engineering Conference, Garmisch, Germany, October 1968
- Will Tracz, Confessions of a Used Program Salesman, Addison-Wesley, 1995
- Frederick Brooks, The mythical man-month: essays on software engineering, Addison-Wesley, 1975
- Tom De Marco, Slack, Getting Past Burnout, Busywork, and the Myth of Total Efficiency, Broadway Books, 2002
- Diomidis Spinellis, Georgios Gousios, Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design, O'Reilly, 2009
- Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl E. Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Sec. Comput. 1(1): 11-33 (2004)
- Ken Thompson, **Reflections on trusting trust**, Comm. ACM, 27(8): 761-763, August 1984