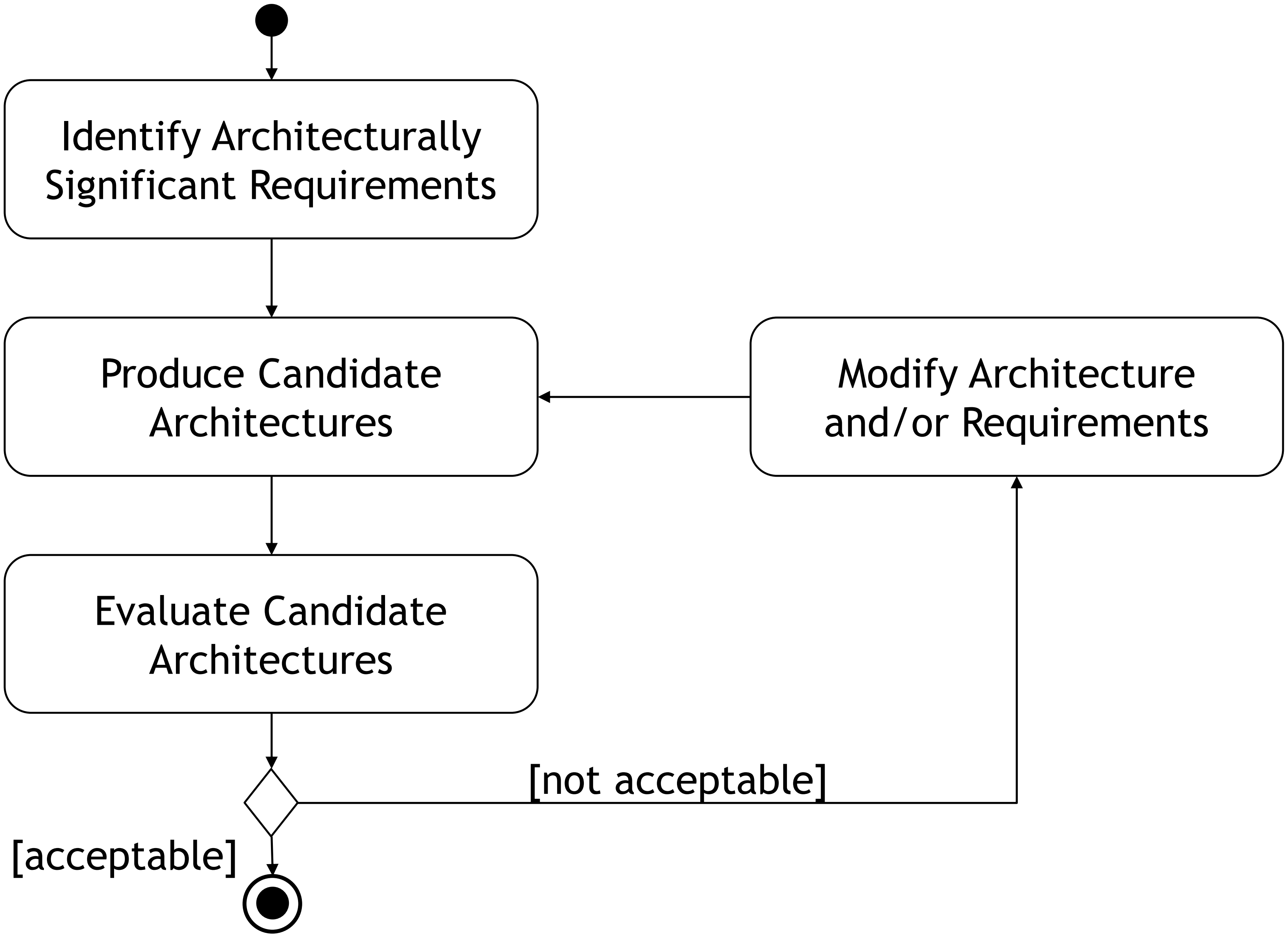
Requirements Engineering and Software Architecture

Architecture and Quality Requirements

Emmanuel Letier

http://letier.cs.ucl.ac.uk/

Architecture Definition Process

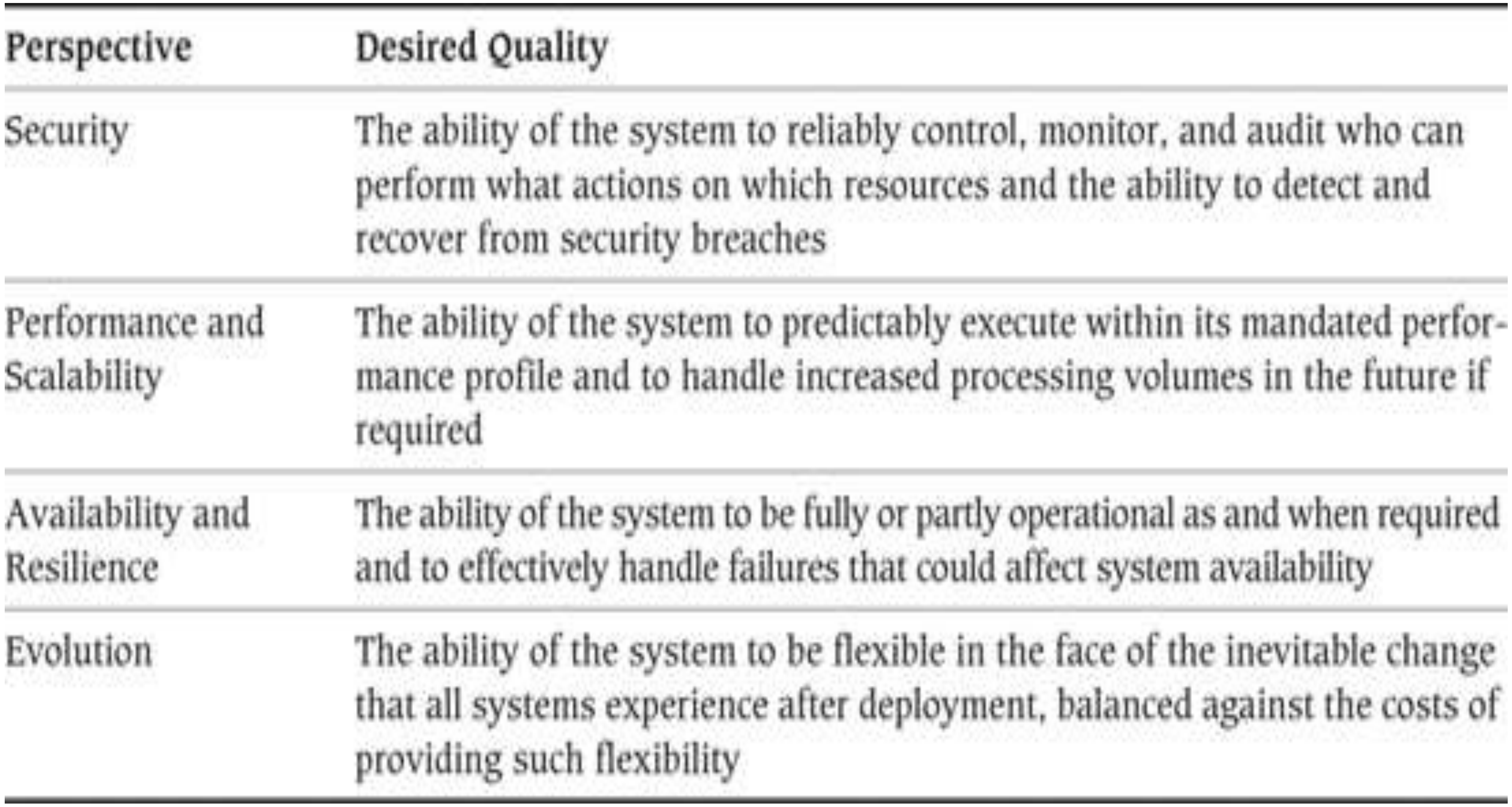

Identify Architecturally Significant Requirements
Produce Candidate Architectures
Modify Architecture and/or Requirements
Evaluate Candidate Architectures
[not acceptable]
[acceptable]

3-4. Evaluate and Rework Architecture

Architectural perspectives =

- guidelines for defining perspective-specific requirements (e.g. performance, availability, security, evolution, etc.)
- techniques for evaluating architecture against these perspective-specific requirements
- architecture tactics for modifying an architecture to satisfy the perspective-specific requirements

Quality Requirements



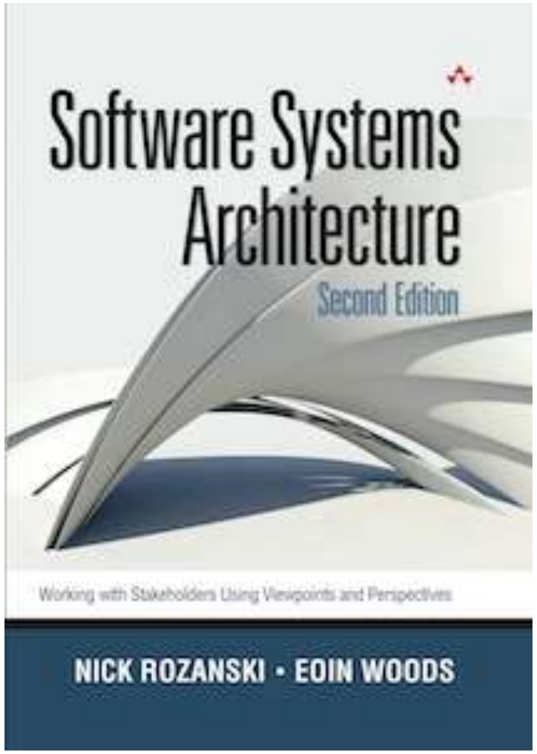
Perspective	Desired Quality
Security	The ability of the system to reliably control, monitor, and audit who can perform what actions on which resources and the ability to detect and recover from security breaches
Performance and Scalability	The ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes in the future if required
Availability and Resilience	The ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability
Evolution	The ability of the system to be flexible in the face of the inevitable change that all systems experience after deployment, balanced against the costs of providing such flexibility

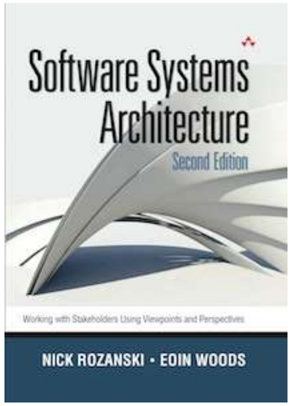
Key orthogonal concern: Cost = development & deployment cost + operational cost + maintenance & evolution cost

Reference

N. Rozanski and E. Woods, Software Systems Architecture : Working With Stakeholders Using Viewpoints and Perspectives, 2nd Ed., Addison-Wesley, 2011

Chapters 25 to 28


Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS


Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS

Chapter 25

Security

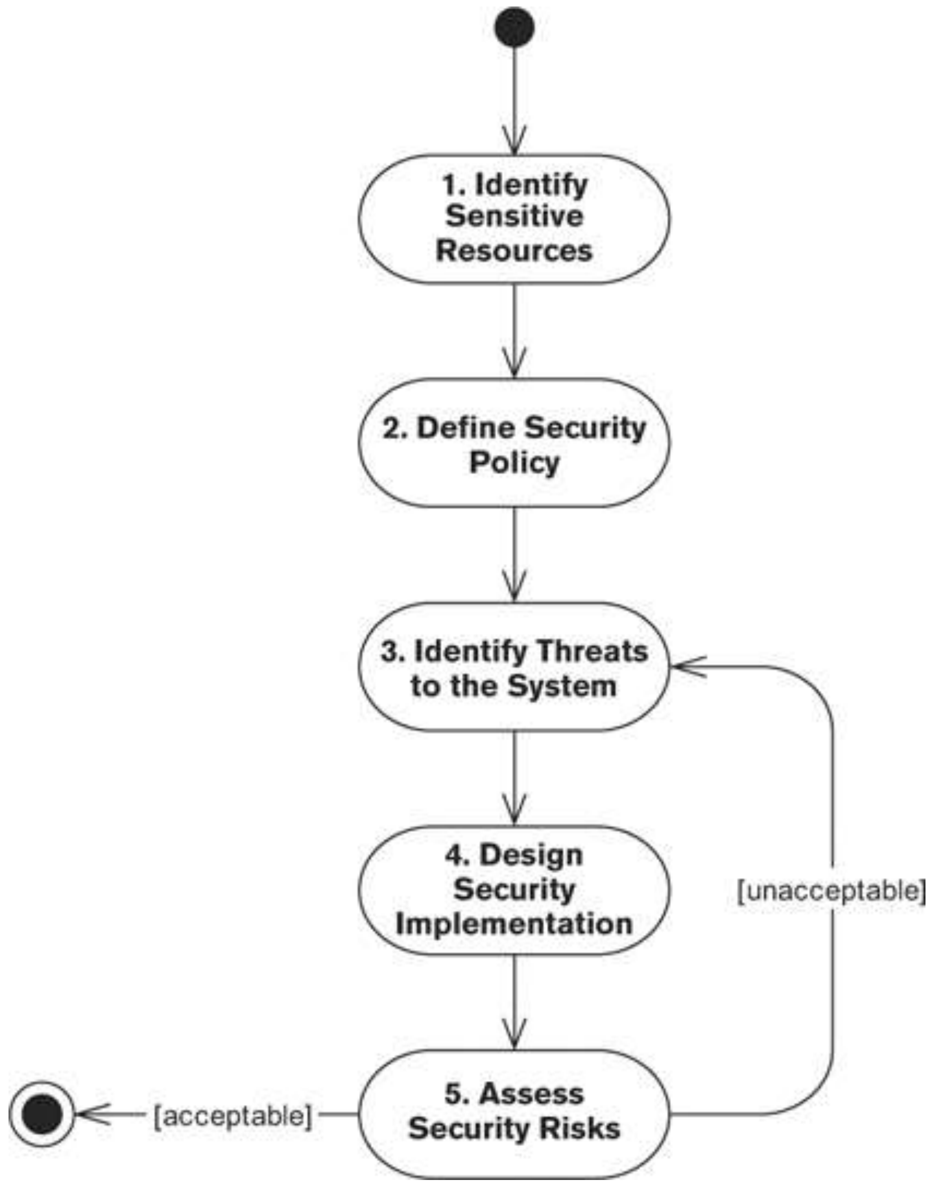
The ability of the system to reliably control, monitor, and audit who can perform what actions on which resources and the ability to detect and recover from security breaches

Security goals

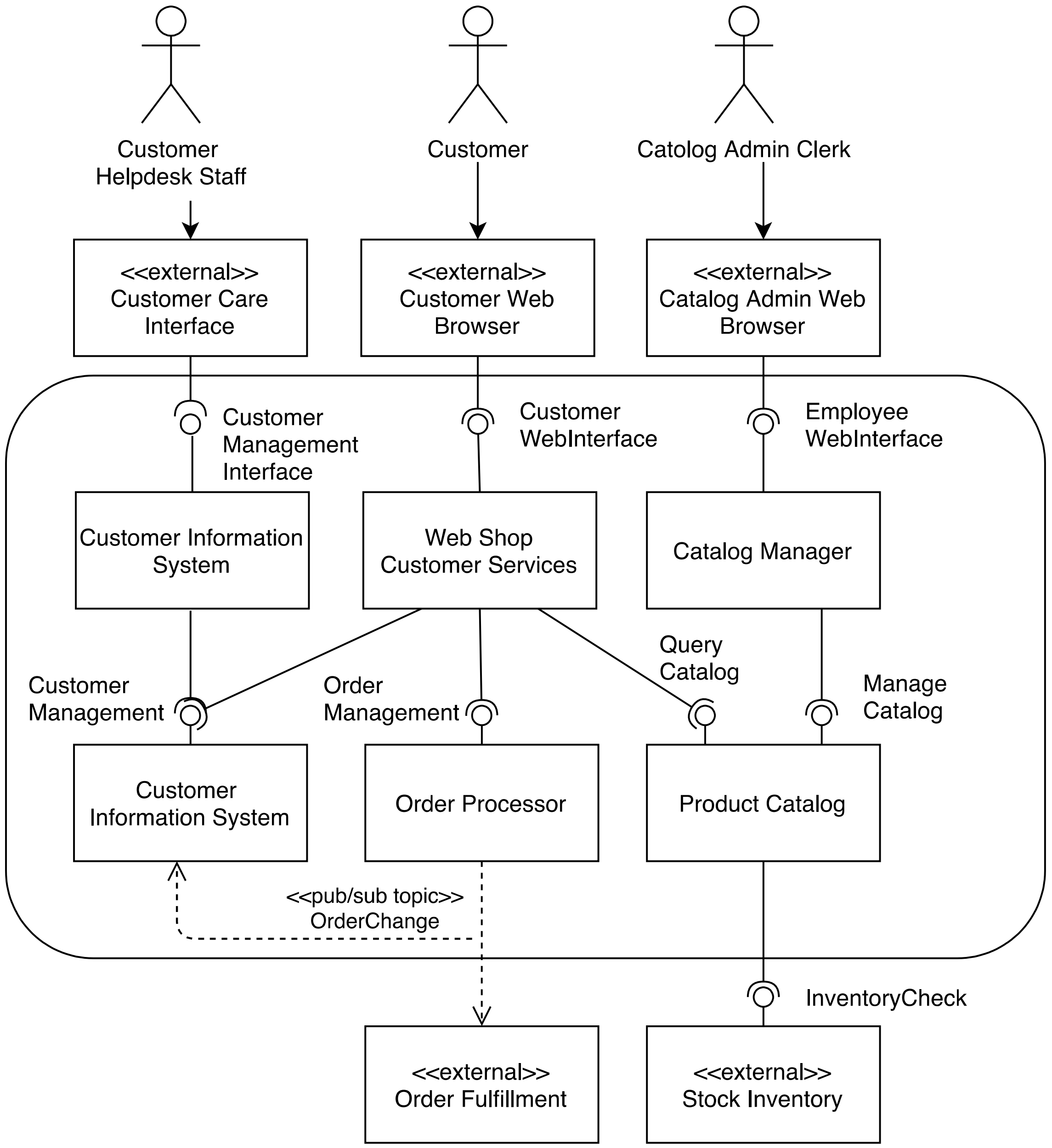
- Goals concerned with protecting asset against harm
  - asset = any resource of value to stakeholders, can be tangible (e.g. cash) or intangible (information, reputation)
  - type of harms include ...
    - … disclosure (**C**onfidentiality)
    - … modification (**I**ntegrity)
    - … unavailability (**A**vailability)
- Examples
    - Avoid disclosure of medical record
    - Avoid illegitimate modification of student’s results
    - Avoid unavailability of bank account

(based on Haley et al., Security requirements engineering: A framework for representation and analysis, *IEEE TSE, Jan 2008)*

Software Security Engineering Activities


1. Identify Sensitive Resources
2. Define Security Policy
3. Identify Threats to the System
4. Design Security Implementation
5. Assess Security Risks
[unacceptable]
[acceptable]

Example: an e-commerce website


Customer Helpdesk Staff
Customer
Catolog Admin Clerk
<<external>> Customer Care Interface
<<external>> Customer Web Browser
<<external>> Catalog Admin Web Browser
Customer Management Interface
Customer WebInterface
Employee WebInterface
Customer Information System
Web Shop Customer Services
Catalog Manager
Customer Management
Order Management
Query Catalog
Manage Catalog
Customer Information System
Order Processor
Product Catalog
<<pub/sub topic>> OrderChange
InventoryCheck
<<external>> Order Fulfillment
<<external>> Stock Inventory

Identify sensitive assets and security goals

E.g. as a simple table (based on Rozanski & Woods, 2011)

Assets	Sensitivity	Security Goals
Customer Account Records	Personal information of value for identity theft and violation of privacy	Confidentiality, Integrity, Availability
Product Catalog Records	Defines what is for sale and its description; if maliciously changed could harm business	Integrity, Availability
Product Pricing Records	Defines pricing for catalog items; if maliciously or accidentally modified, could harm the business or allow fraud	Integrity, Availability

Sometimes it is useful to develop a security goal model showing parent goals and subgoals of some important security goals

Define the business security policy

Defines who has what access to which assets and what operations on these assets are logged in an audit trail

Asset\Agent	Data administrator	Customer helpdesk	Catolog Clerk	Product Price Administrator
Customer Account Records	Full access with audit	Full access to non-sensitive data only	No access	No access
Product Catalog Records	Full access with audit	Read access	Write with audit	Read access; Write with audit
Product Pricing Records	Full access with audit	Read access	Read access	Write access with audit

Sometimes a goal model is use to relate security policies to security goals.

Identify Threats

Identify who may want to violate the security policy, why, and what are the likelihood and severity of such violations

Attacker	Objective	Likelihood of Attempts	Severity if Successful
Identity thieves	Obtain customers credit card details	High	High
Rogue customer	Obtain goods without paying or at lower price	Medium	Medium
Rogue or disgruntled employee	Hurt company reputation or financially Financial gain	Medium	High

- Consider insider as well as outsider threats
- Consider potential violation of each security goal

Model Threats with Attack Trees

Example

Attacker's Objective: Obtain customer credit card details

- Extract details from system database
  - Crack DB password or exploit DB vulnerability
  - Obtain details from DB admin staff
- Extract details from the Web interface
  - Crack customer password
  - Phishing attack
  - Exploit web server vulnerability
- Obtain details from customer service staff
- Obtain details from customer by posing as customer service staff

Design the security implementation using architectural tactics for security

Mitigation Tactics

- Authentication mechanisms
  - Password, smart card, biometric, single-sign-on
- Access control
  - Typically using 3rd party security infrastructure
- Limit access using firewall
- Secrecy
  - Needed to protect sensitive information that leaves our system
  - Use secure network link (SSL/TLS); encrypt data on client machine

Detection and Recovery Tactics

- Intrusion detection system
- Restrict access when attack
- Use availability tactics (see later) to recover from failures

Common Problems and Pitfalls

Inadequate Security Requirements

- No clear security requirements
- Complex security policies
- Ignoring the insider threat

Architecture pitfalls

- Technology-driven approach & piecemeal security
- Using ad hoc or unproven security technology
- Security embedded in the application code
- Lack of administration facilities
- System does not fail securely
- Security as an afterthought

Checklist for Security Requirements

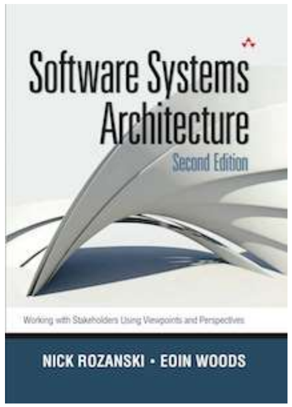
- What are the system's security goals?
  - What are the sensitive resources?
  - What are the confidentiality, integrity, availability, and accountability goals?
- What are the system's security policies?
  - Do these policies satisfy the security goals?
  - Are these policies as simple as possible?
- What are the system's security threats?
  - Have you considered insider as well as outsider threats?
  - Have you consider how the system's deployment environment will alter the threats?

Checklist for Architecture Definition

- What security mechanisms does your system include to prevent or reduce each security threat?
- What security mechanisms does your system include to detect and recover from security breaches?
- Have you considered end-to-end security?
  - Is there a security gap?
  - What is the system's weakest point?

Checklist for Architecture Definition (2)

- Are you using proven third-party security infrastructure as much as possible?
- Are your security mechanisms cost-effective?
- Have you considered impact on other qualities?
  - e.g. impact on performance and usability


Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EOIN WOODS

Chapter 26

Performance and Scalability

The ability of the system to predictably execute within its mandated performance profile, and to handle increased performance volume if required

Performance & Scalability Concerns

- **Response time**: the time it takes for a transaction to complete in the context of a defined work load
  - E.g. Under normal load, the time needed to authorise an online payment should be less than 3 seconds in at least 90% of cases
- **Throughput**: the number of transactions the system is capable of handling per unit of time (second, minute, or hour)
  - E.g. The system must be able to process at least 10,000 transactions per minute
- **Predictability:** similar transactions should complete in a similar amount of time
  - Sometimes a more desirable quality than absolute performance

Performance & Scalability Concerns (2)

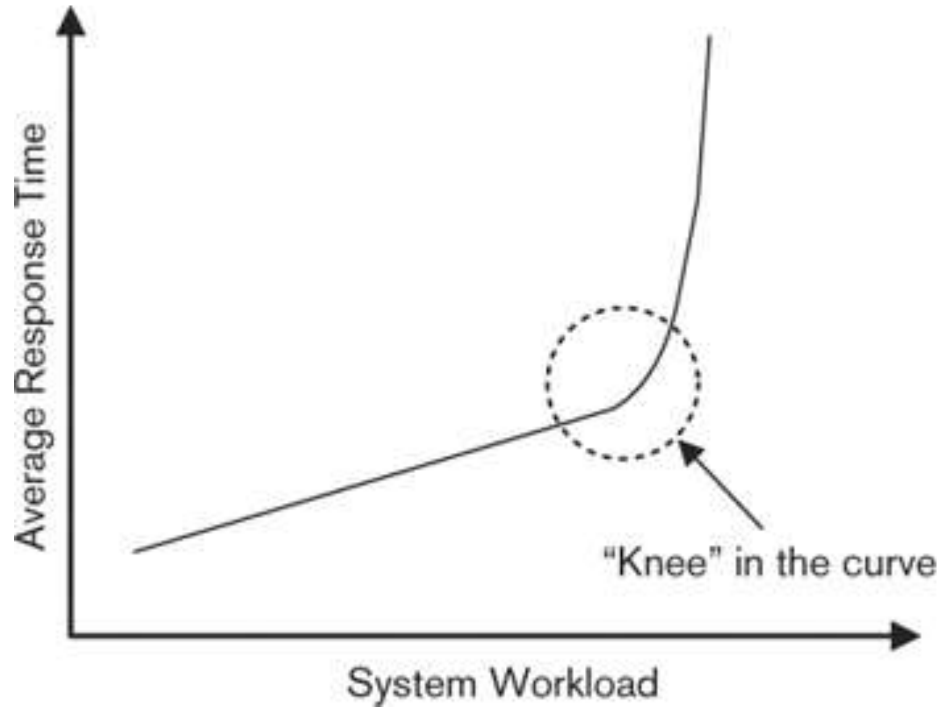
- **Scalability**
  - Short definition: the ability of a system to handle an increased workload.
  - Long definition: “the ability of a system to maintain the satisfaction of its quality goals to levels that are acceptable to its stakeholders when characteristics of the application domain and the system design vary over expected. operational ranges” (Duboc et al., IEEE TSE 2013)
- E.g
  - The ability of a web search engine to return results in a constant time as the number of simultaneous queries and number of indexed pages increase.
  - The ability of an air traffic control system to keep safe separation distance between airplane as the number of airplanes it has to manage increases.

Performance & Scalability Concerns (3)

- Hardware Resource Requirements
  - Deciding how much and what type of hardware the system will need
  - Must be considered early because of cost, acquisition time, and physical constraints (e.g. floor space)

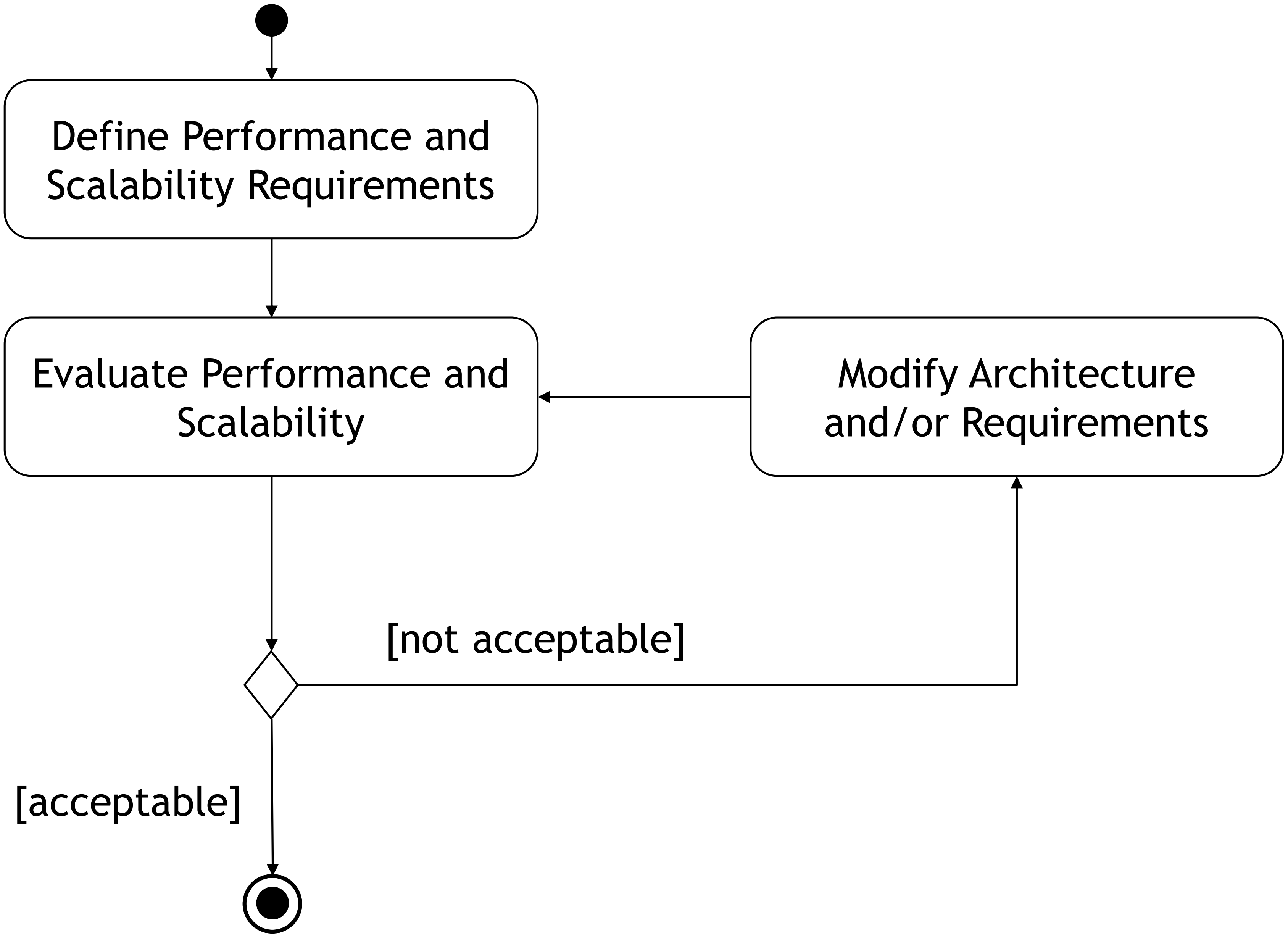
Performance & Scalability Concerns (4)

- **Performance breakdown**: nearly all computer systems have a workload beyond which performance degrades abruptly



This behaviour is generally caused by one or more system resources becoming so overloaded that they can no longer work effectively (e.g. swamped memory, swamped network connection)

Performance Engineering Activities


Define Performance and Scalability Requirements
Evaluate Performance and Scalability
Modify Architecture and/or Requirements
[not acceptable]
[acceptable]

Define Performance and Scalability Requirements

1. Identify business-oriented performance goals

e.g. for a financial system: “*be fast enough to support a workload of 20,000 back-office transactions per day*”

2. Derive software performance requirements

- response times for specific functions under a given load

  e.g. *95% of orders should be process within 3 sec when the system is under a load of up to 500 orders per second.*

- throughput requirements

  e.g. *system must be able to handle up to 500 orders per sec*

- scalability requirements

  e.g. *response time should stay under 3 seconds when in 6 months the load is expected to rise to 2000 orders per sec.*

Example: a Business-Oriented Performance Goal

GOAL Achieve[Accurate Incident Information]

Responsibility CAD, Call Handler, Public

Desired Behaviour

GIVEN an incident has occurred
WHEN the ambulance service receives an emergency call reporting the incident
THEN within 1 minute
The CAD has accurate information about the incident location and number and types of casualties

Fit Criteria

- Throughput: the ambulance service must be able to handle up to 100 calls per minute
- Responsiveness: for at least 90% calls, incident information is obtained and entered in CAD in less than 1 minute

Software performance requirements will be derived from this performance goal and domain assumptions

Responsiveness and user experience

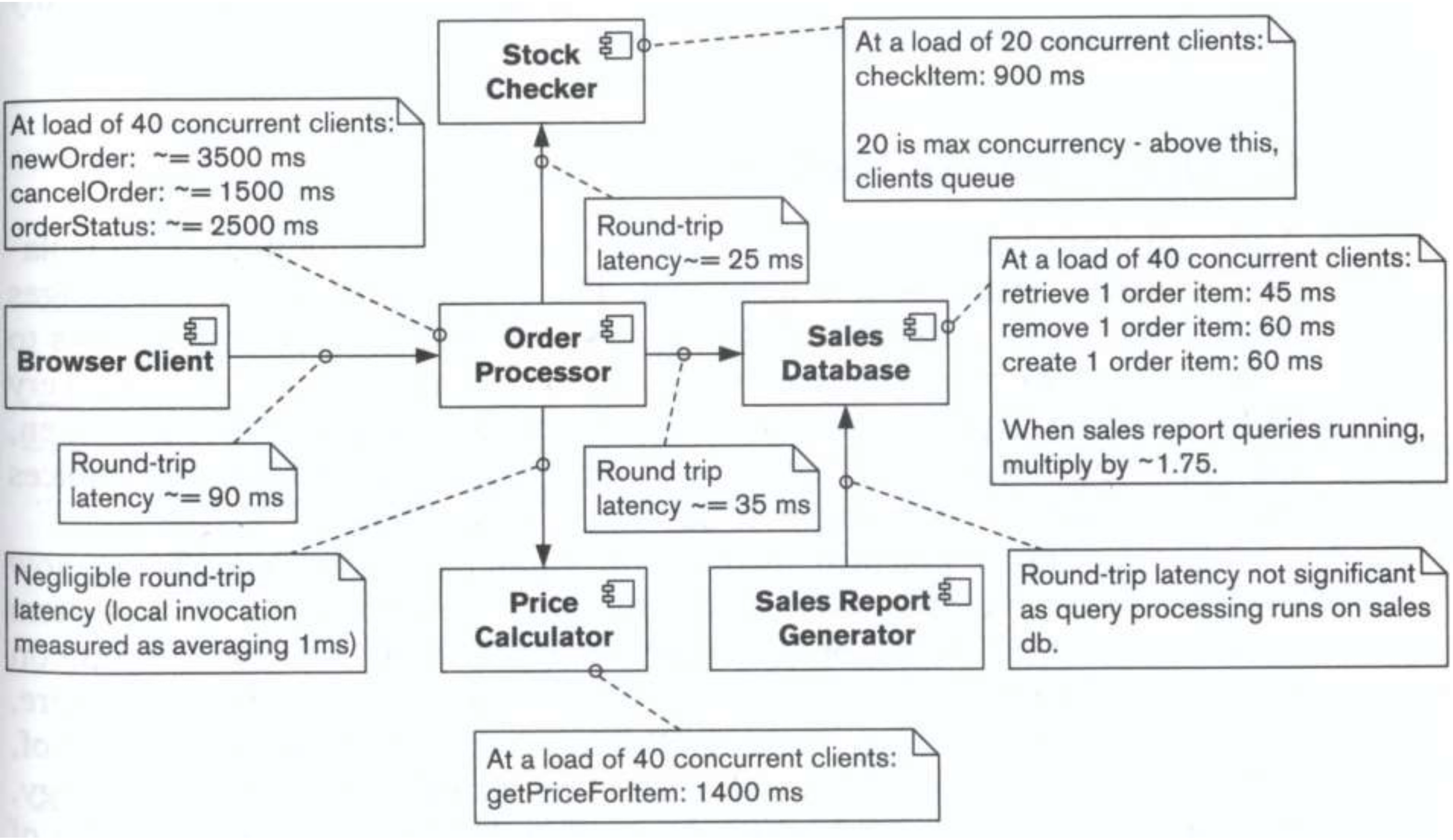
Response time limits	User Experience
0.1 second	Limit for which the user feels the system is **reacting instantaneously**
1 second	Limit for the user's **flow of thought** to stay uninterrupted even though the user will feel the delay
10 seconds	Limit for **keeping the user's attention** focused on the dialogue

Nielsen, Jakob. Usability engineering. Elsevier, 1994.

Evaluate Performance and Scalability

- Using performance models
  - Models vary from simple back-of-the-envelope calculations to complex simulations and statistical models.
  - Estimating model parameters (e.g. predicted response times of each component under a given load) relies on past experiences, quick prototyping, or intelligent guesswork.
  - Main purpose is to identify bottleneck and estimate load at which system performance would break down rather than precise performance metrics.
  - But never sure the system will behave as modelled.
- Using performance testing
  - Can start early on prototypes and architecture skeleton to calibrate and validate performance model

Example Performance Model


Stock Checker
At a load of 20 concurrent clients: checkItem: 900 ms
20 is max concurrency - above this, clients queue
At load of 40 concurrent clients: newOrder: ~= 3500 ms cancelOrder: ~= 1500 ms orderStatus: ~= 2500 ms
Round-trip latency~= 25 ms
At a load of 40 concurrent clients: retrieve 1 order item: 45 ms remove 1 order item: 60 ms create 1 order item: 60 ms
When sales report queries running, multiply by ~1.75.
Browser Client
Order Processor
Sales Database
Round-trip latency ~= 90 ms
Round trip latency ~= 35 ms
Negligible round-trip latency (local invocation measured as averaging 1ms)
Price Calculator
Sales Report Generator
Round-trip latency not significant as query processing runs on sales db.
At a load of 40 concurrent clients: getPriceForItem: 1400 ms

Architectural Tactics for performance & scalability

- "Throw more hardware at the problem"
  - Scaling up: replace existing hardware with higher-capacity components (e.g. servers with faster processors)
    - Simple but it can be costly and is limited to largest machine available
  - Scaling out: add more components (e.g. add more servers)
    - More cost-effective but works only if the system is designed to take advantage of additional hardware

Architectural Tactics for performance & scalability

- Reduce Contention via Replication
  - A resource contention is a situation where multiple tasks require access to a shared resource simultaneously (e.g. a an application server, a database server)
  - Performance and scalability can be improved by replicating the resource
    - E.g. multiple instances of a web-server; replicating a database across multiple geographical regions
- Partition and Parallelize
  - Some large lengthy processes can be partitioned into smaller process that can be executed in parallel and their results combined (e.g. MapReduce)
  - Not always applicable and reduce response time at the cost of requiring more processing resources

Architectural Tactics for performance & scalability

- Reuse Resources and Results (Caching)
  - For operations that are computationally expensive or need access to a relatively slow external service
  - Create a cache that store results and reuse the results when they are required again

- Distribute processing over time
  - Many systems have different loads at different time of the day with large differences between peaks and troughs
  - Analyze the peak time load to see whether some of the workload can be moved to other time

Architectural Tactics for performance & scalability

- Degrade Gracefully
  - E.g. reject requests when system is overloaded and ask to try again later; prioritize certain requests over others; cap number of request a client can make per unit of time; …
  - Needs proactive monitoring to be able to quickly detect and respond to performance concerns
- Make Design Compromises
  - Highly modular, loosely coupled designs have performance cost
  - Compromising the design modularity can improve performance but will make system more difficult to maintain and negatively impact other qualities (e.g. availability and security)

Common Pitfalls

Analysis problems

- Imprecise performance and scalability goals
- Overconfidence in positive results from simplified performance models and performance testing
- Invalid assumptions about domain or technology

Architecture design problems

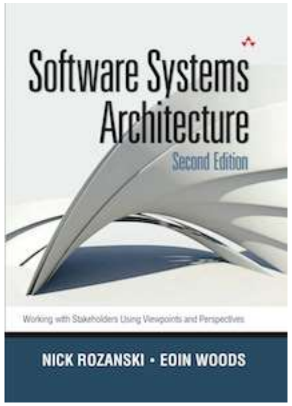
- Inappropriate partitioning (“God element”)
- Transactions with high overhead (network latency, data serialization, security processing, database access, etc.)
- Disregard differences between local and remote invocations
- Database contention
- Concurrency-related contention
- Careless allocation of resources: automatic resource management (e.g. garbage collection) is not free

Checklist for Performance Requirements

- What are your system's response time and throughput requirements?
- Are these requirements reasonable?
- Are your performance targets defined in the context of a particular load on the system?

Checklist for Architecture Definition

- What architectural tactics are you using to achieve the required performance levels? Why?
- What performance-related assumptions have you made about your system?
- What are the potential performance bottlenecks in your architecture?
- What workload can your system handle before breaking down?

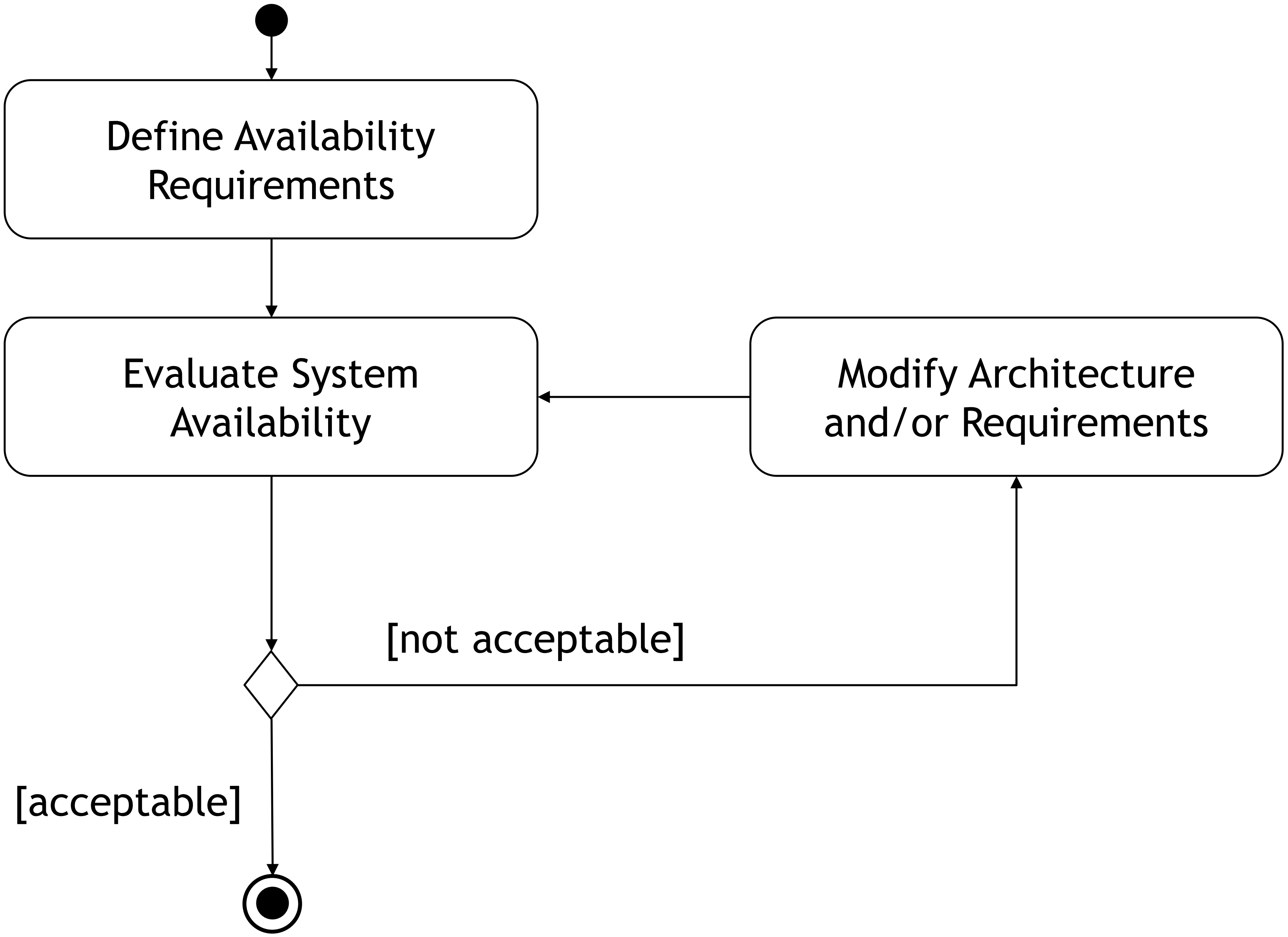

Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS

Chapter 27

Availability and Resilience

The ability of the system to be fully or partly operational as and when required, and to effectively handle failures that could affect system availability.

Availability Engineering Activities


Define Availability Requirements
Evaluate System Availability
Modify Architecture and/or Requirements
[not acceptable]
[acceptable]

Availability

the proportion of time the system is up and running and available to provide a service to users

Availability	Downtime per year
90%	40 days
99%	4 days
99.9%	9 hours
99.99%	50 minutes
99.999%	5 minutes
99.9999%	30 seconds

Stakeholders’ availability concerns

- Classes of services
  - A system offers different type of services to its users
  - Not all services need the same level of availability
  - Not all services need to be available at all time
- For a given service, availability concerns include
  - how often and when failure occurs
  - how fast the system recovers from failure
  - what degraded services are provided during failures
- Planned downtime
  - IT operation stakeholders need occasional windows of time for hardware and software updates

Documenting Availability Requirements

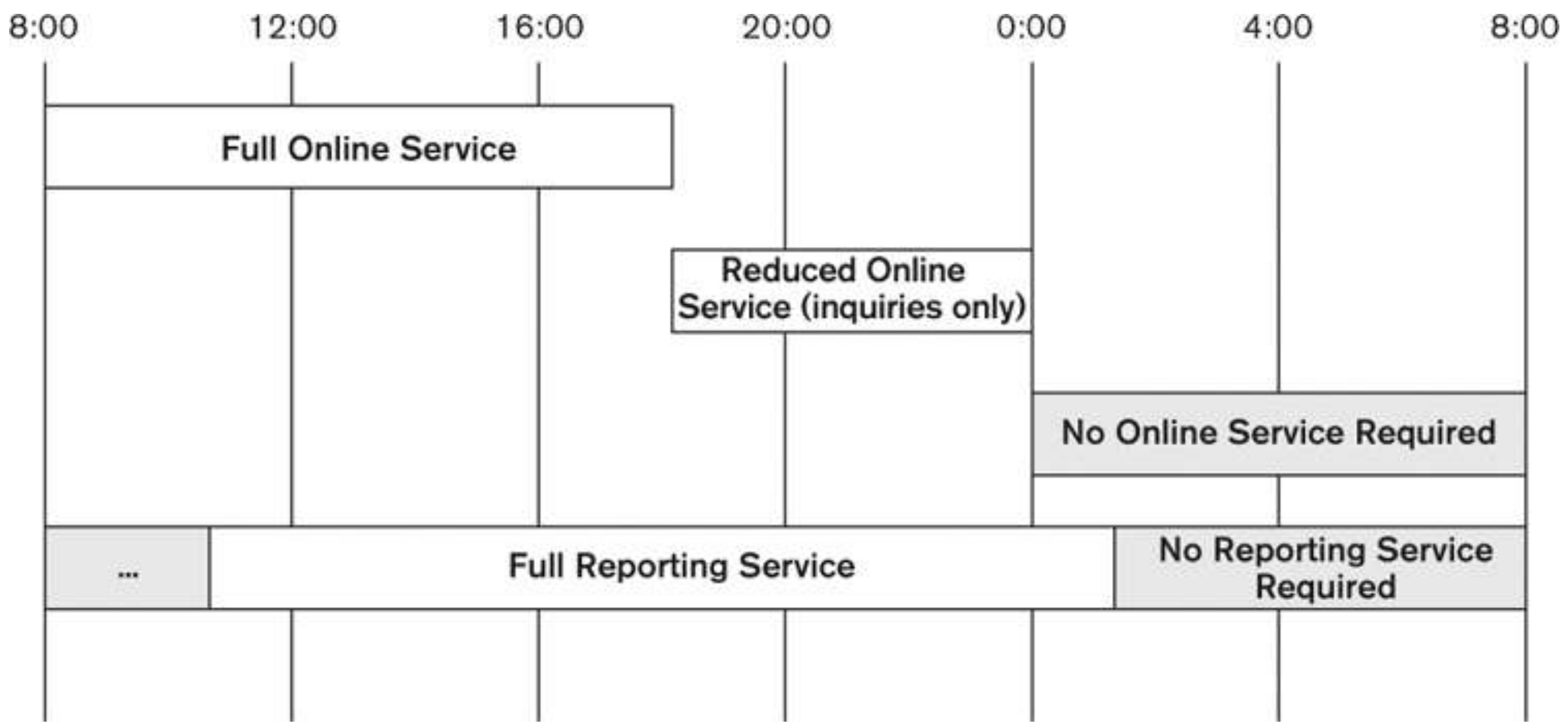
A simple table is usually effective

Goals (Services)	Availability Fit Criteria	Failure Handling Requirements
Achieve[Emergency Call Answered]	downtime: less than 5 sec/day	
Achieve[Known Nearest Available Ambulance Allocated]	crashes: less than once per month	• Achieve[Allocation Service Restored within 30 seconds after Crash] • Achieve[Manual Ambulance Allocation While CAD Allocation Service is Down]
Achieve [Weekly Performance Report Generated]	• at least 90% of reports generated on time • no report more than a week late	

Producing an availability schedule

- The availability schedule specifies periods during which the different type of services must be available

E.g. for an online enterprise application software


8:00
12:00
16:00
20:00
0:00
4:00
8:00
Full Online Service
Reduced Online Service (inquiries only)
No Online Service Required
...
Full Reporting Service
No Reporting Service Required

Resilience

- Disaster Recovery: system recovery after a disaster in the physical environment (e.g. fire)
  - Typically remote backups, standby machines, and failover process
- Business continuity: recovery of the whole business after a major disruption
  - Involves system recovery plus ensuring availability of staff, working environment, and communication infrastructure

Evaluating System Availability

- For hardware

$$availability = \frac{MTBF}{MTBF + MTTR}$$

MTBF = mean time between failure; *MTTR* = mean time to repair

- For software availability
  - Evaluate how failure of each component would impact rest of the system and how it would affect users
  - Identify single points of failure with disastrous consequences
  - Sometimes, architects use software reliability models for quantitative evaluation.
- Evaluate whether planned downtime is sufficient
  - Do you have enough time for all required upgrades and internal processing (backups, reconciliation, large batch jobs, etc.)

Architectural Tactics for Availability

- Replication strategies
  - Select fault-tolerant hardware (e.g. RAID storage, routers, standby printers, etc.)
  - Use high-availability clustering: redundant computers deployed in parallel with *failover* and *load balancing* mechanisms
  - Allow for component replication: deploy multiple instances of a functional element on different nodes

Architectural Tactics for Availability

- Design for Failure (i.e. assume failure *will* happen)
  - Decouple architecture so as to limit impact of a failing component
  - Design effective processes and management tools to identify, diagnose and recover from failure
  - Fail gracefully or hide failures from users when possible
  - Run fire drills (e.g. Netflix Chaos Monkey)
- Identify backup and disaster recovery solutions
  - Design data backup *and restore* strategies; use transaction logs
  - Design strategies for business continuity

Problems and Pitfalls

- Overambitious availability requirements

  Risk reduction: ensure high availability requirements are backed by real business needs and worth the costs.

- Single points of failure

  Risk reduction: identify single points of failure in functional and deployment views, then consider if cost-effective to improve their reliability (e.g. through hardware duplication)

- Cascading failure

  Risk reduction:

  – ensure all components handle overload gracefully

  – isolate components from impacts of failure of others

- Unavailability through overload

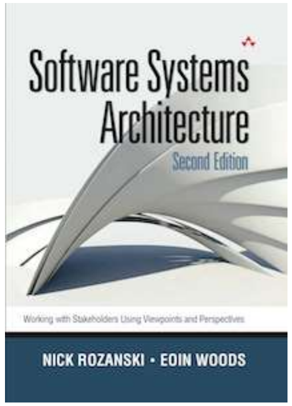
  Risk reduction: ensure components continue to work or degrade gracefully when load increases

Checklist for Availability Requirements

- What are the system’s availability requirements?
- Are they driven by business needs?
- Do they consider different classes of services?
- Do they strike a realistic balance between cost and business needs?

Checklist for Architecture Definition

- What architectural tactics does your system use to achieve the required availability requirements?
  - How do you prevent or reduce failures?
  - How do you recover from failures?
  - Does your system have single points of failure?
  - Does your system have risks of cascading failures?
- What are the justification for using these tactics?
- What are the impacts of your availability decisions on cost, security performance, and functionality?


Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS

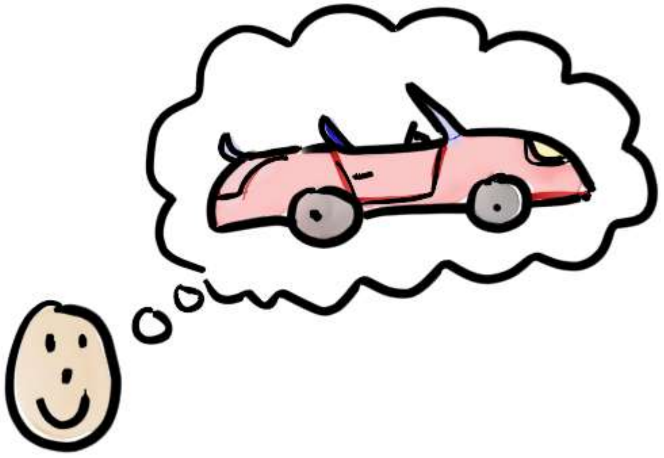
Chapter 28

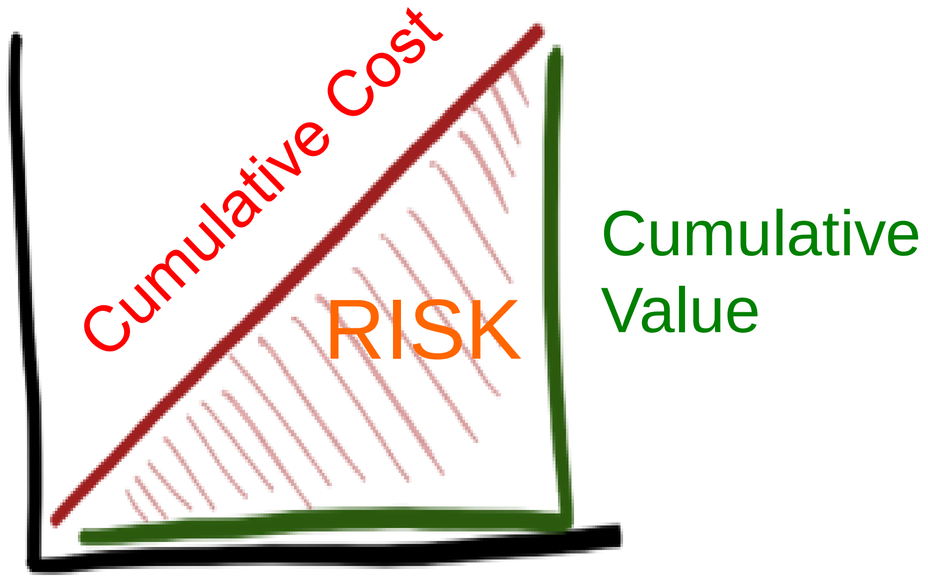
Evolution

The ability of the system to be flexible in the face of the inevitable change that all systems experience after deployment, balanced against the cost of providing such flexibility

Big Bang delivery means high risk

- No business value during development
- No requirements validation during development



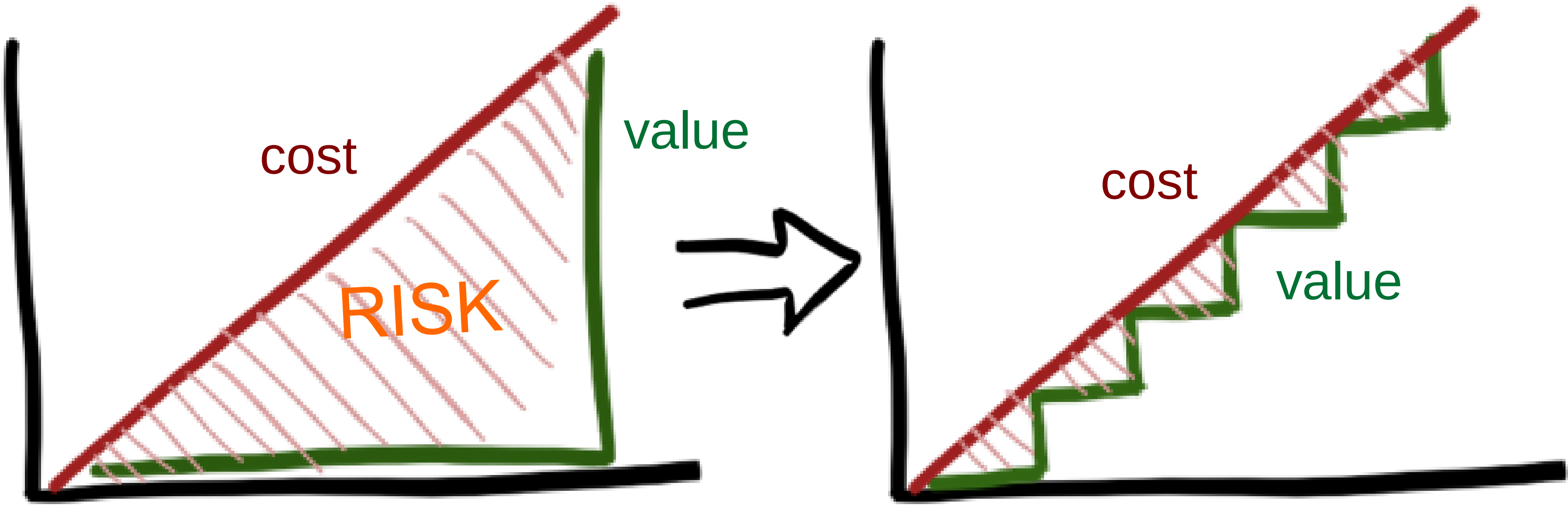


Figures from Henrik Kniberg's talk
"What is Agile?"

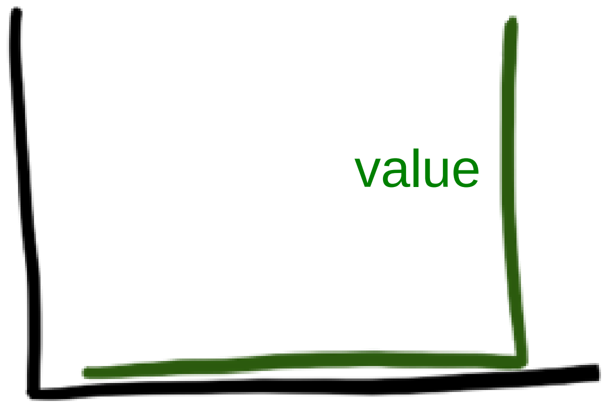
Break the project into smaller parts

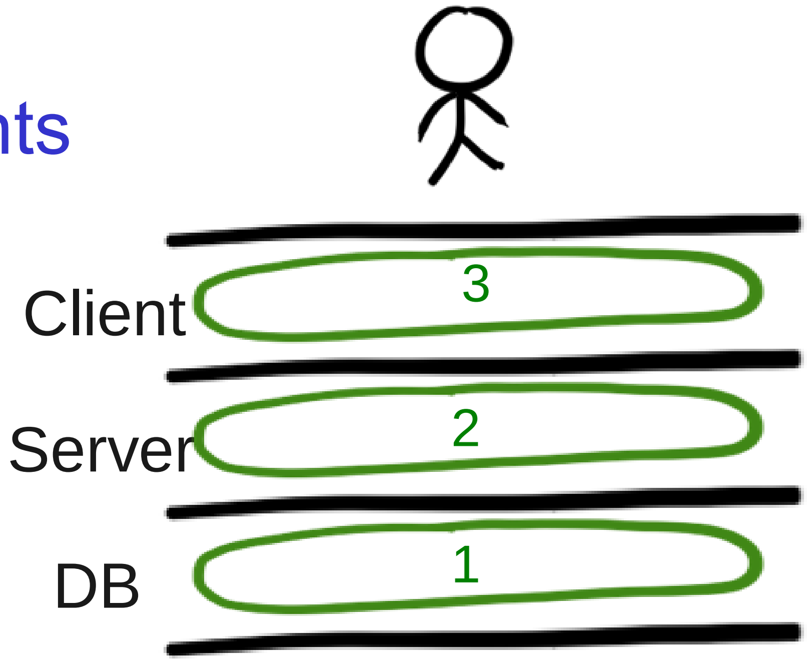
You must find a way to deliver incrementally and iteratively so that

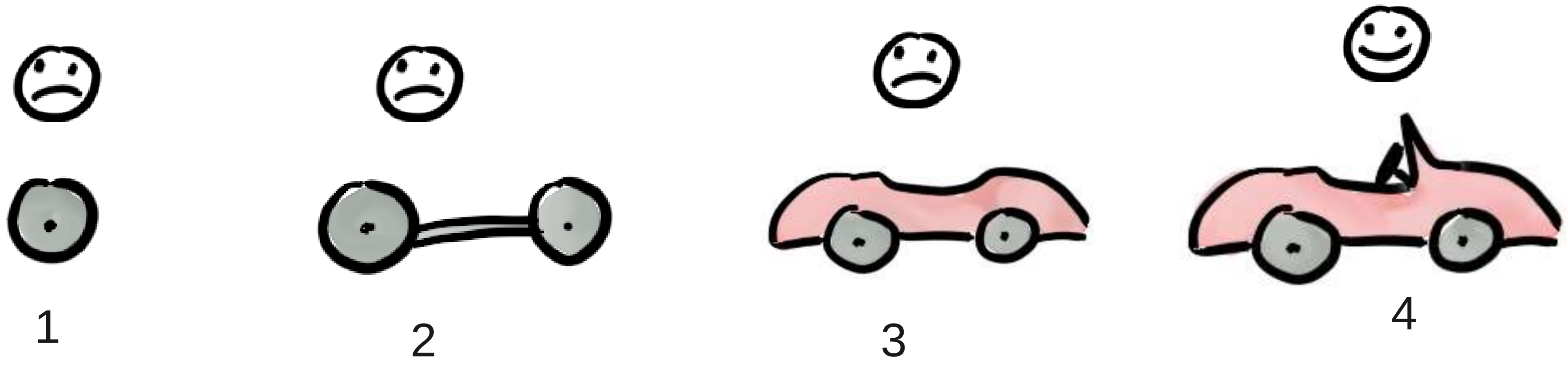
- you deliver business value earlier
- you can validate your requirements and assumptions faster
- you can more easily adapt to changes



Not ”horizontal” increments

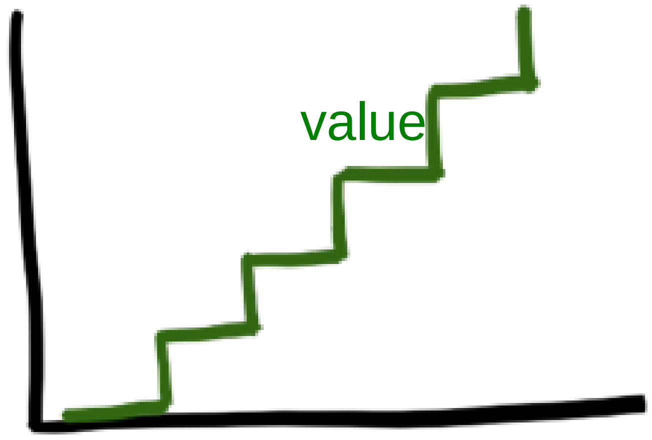


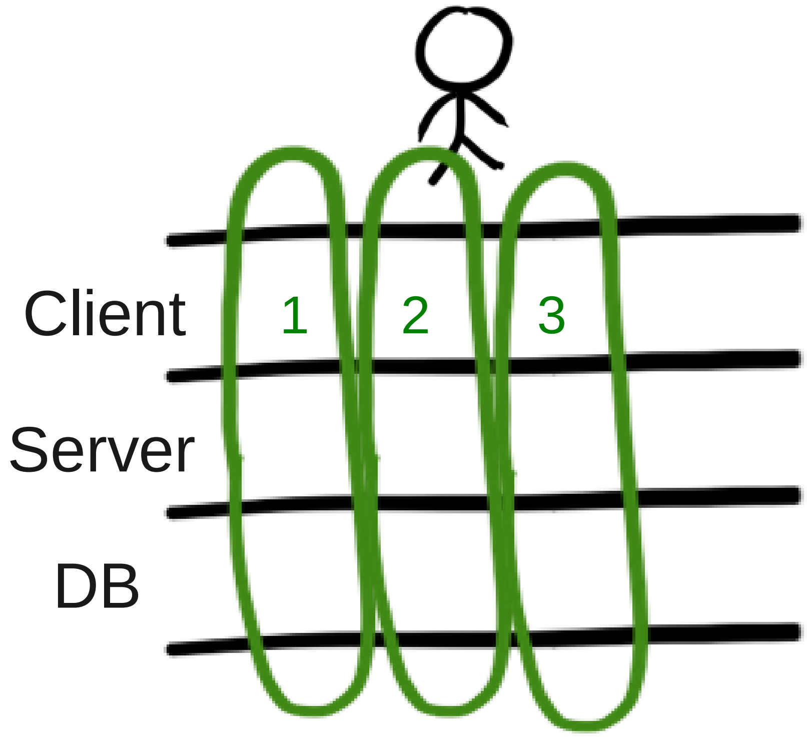


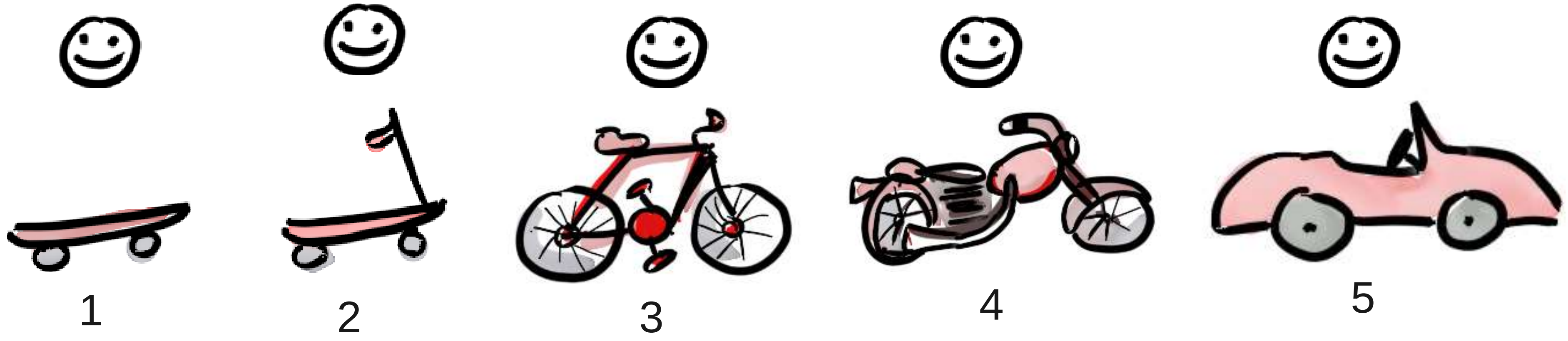


Henrik Kniberg

”Vertical” increments!







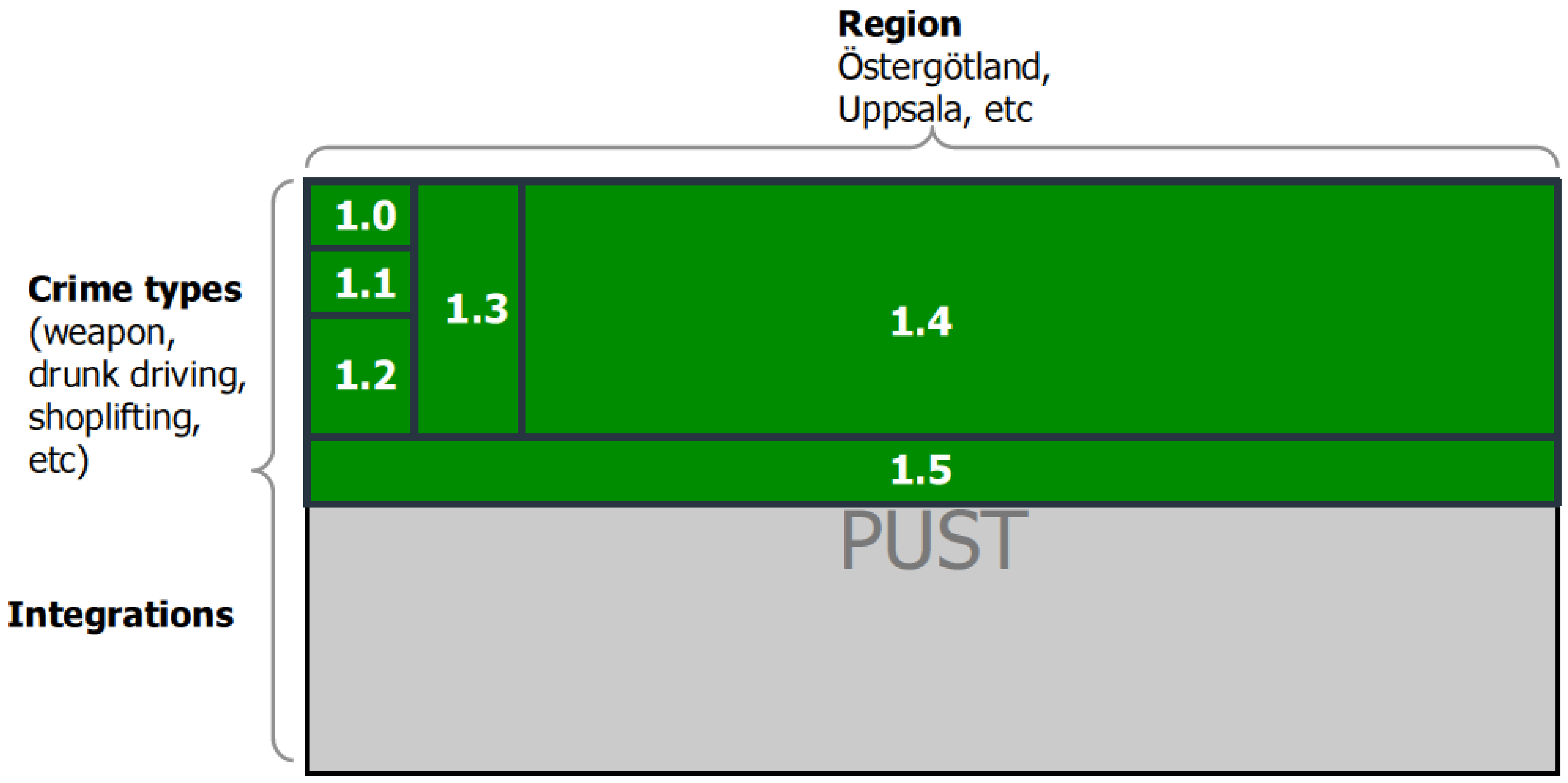
Henrik Kniberg

How to identify increments

- Identify a minimal system
  - “Minimal System” in D. Parnas, Designing Software for Ease of Extension and Contraction, 1979
  - “Minimum Viable Product” in lean product development
  - A system that provides value to stakeholders and allows you to test user behaviours and technology capabilities
- Define a possible product development roadmap
  - The roadmap should allow exploring different paths depending on feedback from previous iterations

Example: Swedish Police mobile technology

from Henrik Kniberg, Lean from the trenches, 2011

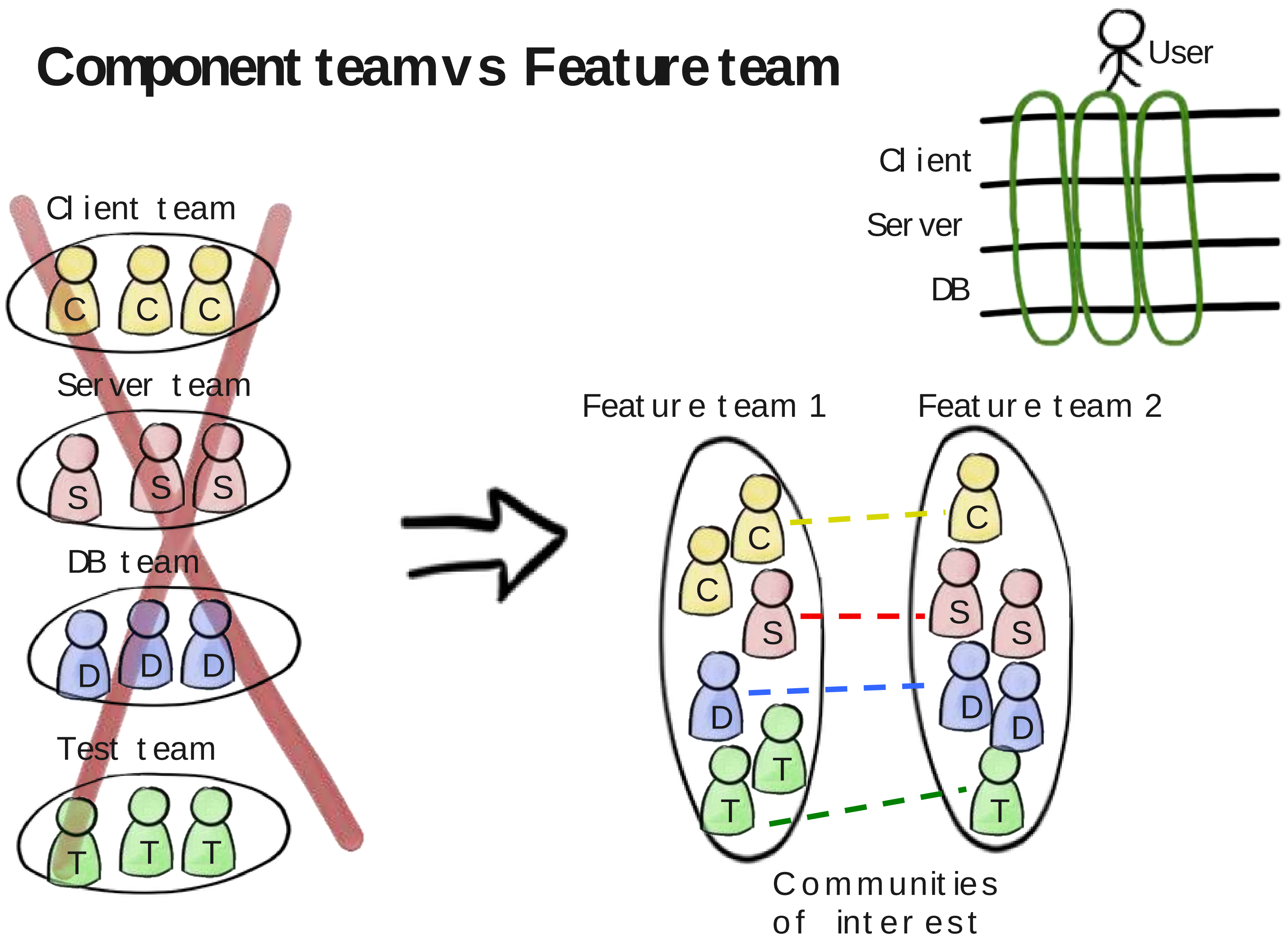


Conway’s Law

"Any organization that designs a system […] will inevitably produce a design whose structure is a copy of the organization's communication structure.”

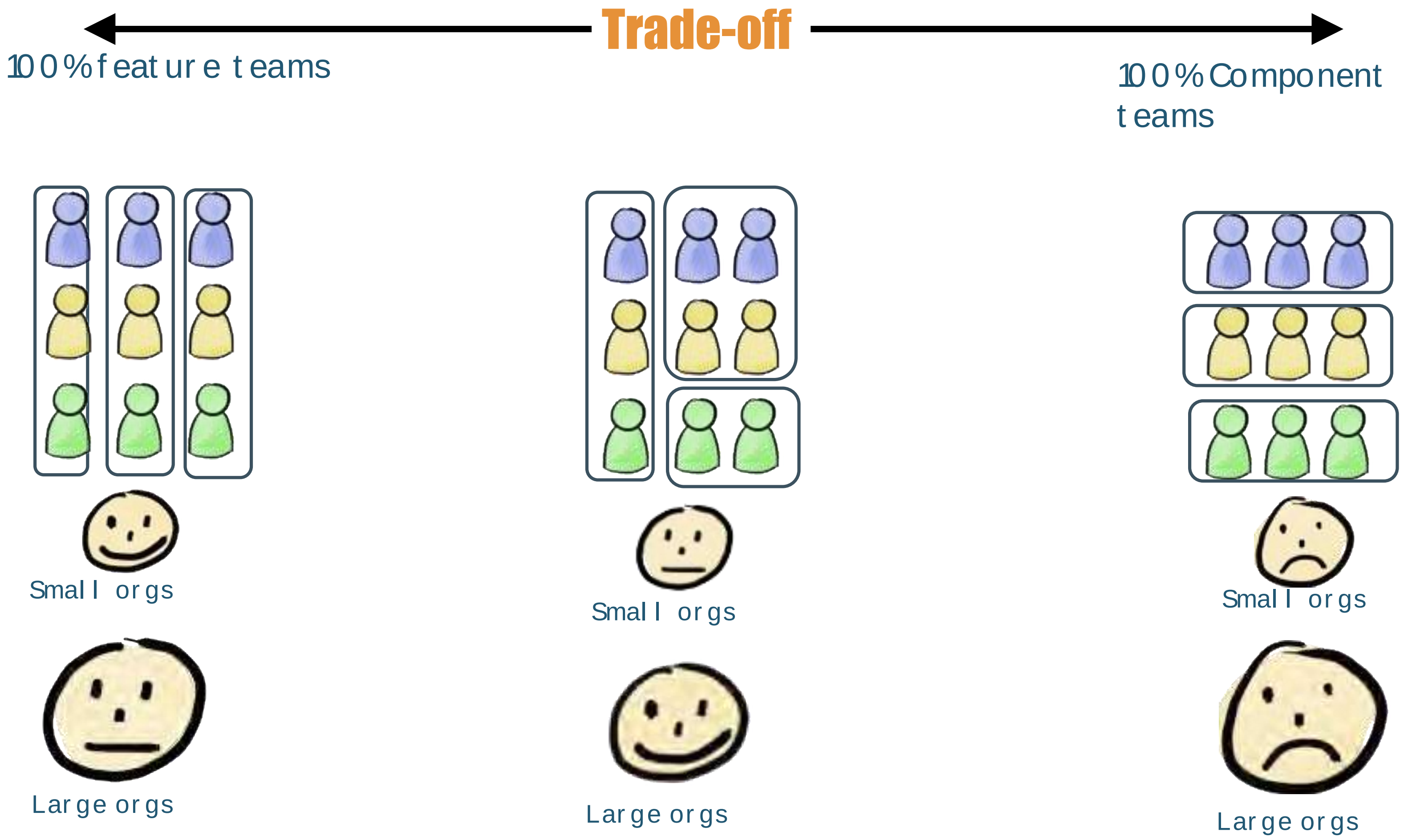
i.e. the software structure mirrors the organization’s communication structure.

How to organize teams



Henrik Kniberg

Team types - finding the right balance



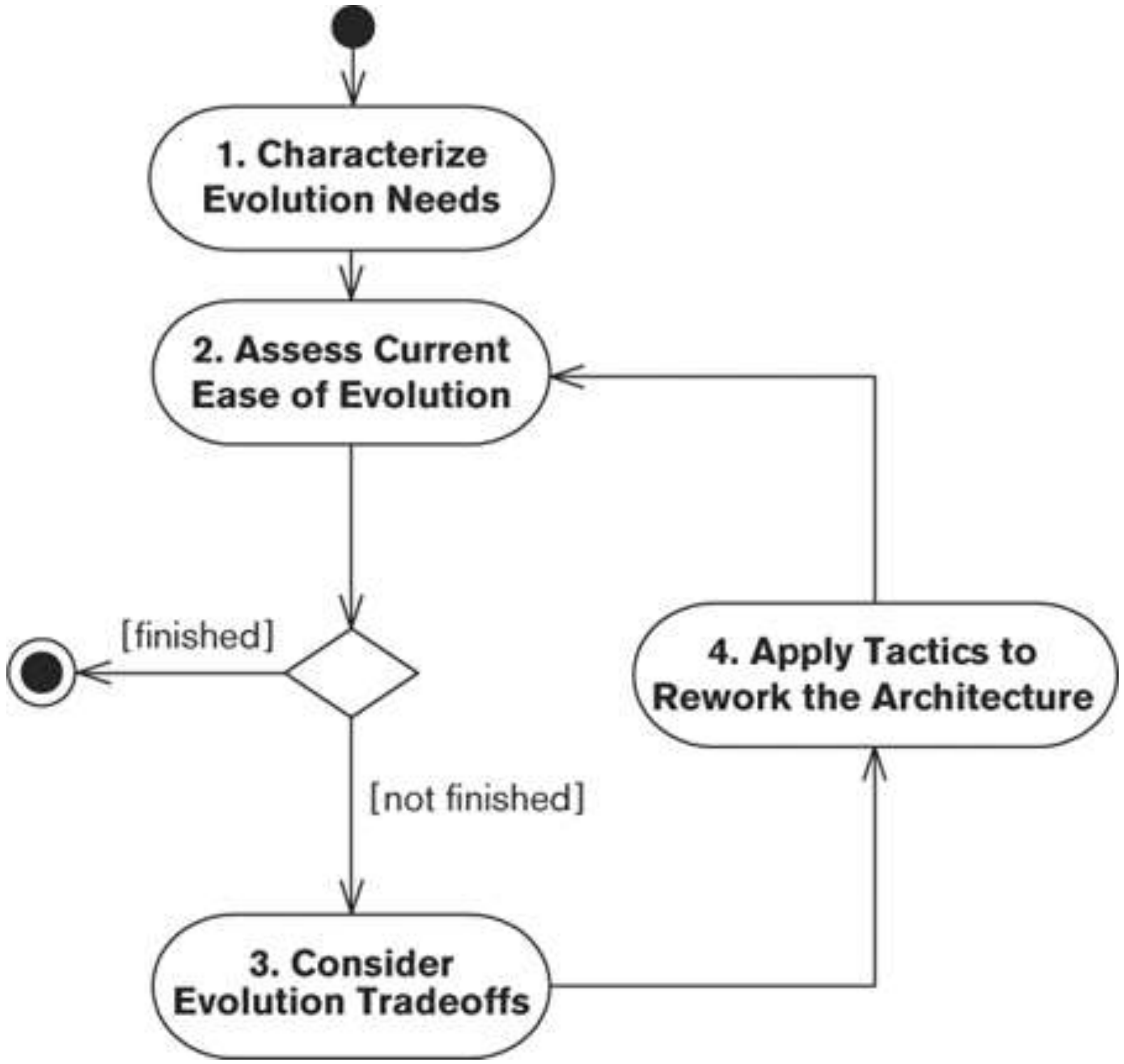
Henrik Kniberg

Stages in the life of a software product

(Bennett & Rajlich, Software maintenance and evolution: a roadmap, 2000)

1. Initial development stage
   - acquisition of knowledge about application domain and user needs
   - definition of a software architecture that will facilitate or hinder future changes
2. Evolution Stage
   - Only successful software evolve!
   - Aim: respond to changing needs and changing environment
   - preserving architectural integrity and team knowledge are critical to evolution
3. Servicing Stage
   - loss of either architecture integrity or team knowledge
   - only small patches are possible
4. Phase-out: software in operation, but no more servicing
5. Close-down: software no longer used

Software Evolution Activities


1. Characterize Evolution Needs
2. Assess Current Ease of Evolution
[finished]
[not finished]
3. Consider Evolution Tradeoffs
4. Apply Tactics to Rework the Architecture

1. Characterize Evolution Needs

You must start by indentifying the likely changes and their characteristics

- Type of change
  - Functional, Platform, Integration, Growth
- Magnitude of change
  - estimation of required effort; local vs. global change
- Likelihood of change
  - how likely will the change actually be required?
- Timescale
  - likely to be needed soon, or far in the future?

⇒ Focus your attention on changes that have high magnitude and high likelihood in the near future.

2. Assess the Current Ease of Evolution

For top priority changes, work through an evolution scenario:

If the likely change becomes reality

- How much of the system must be changed?
  - use the architecture functional, deployment and development views
- How difficult and risky will the changes be?

3. Consider Evolution Tradeoffs

- Alternative strategies supporting changes.
  Two extremes:
  - Design the most flexible system possible so as to make all future changes cheaper (e.g. meta-model architecture)
  - Design the simplest system possible so as to avoid wasting effort in providing unneeded flexibility (extreme agile approach)
- Find the right balance by estimating change likelihood and cost of changes for different architectural choices

4. Architectural Tactics for Evolution

- Design for change
  - Contain changes: define a modular architecture where all components have well-defined and cohesive responsibilities
  - Create flexible interfaces
  - Use architectural styles and design patterns that facilitate change: Layering, Inversion of Control
- Use tools to make changes reliable
  - Configuration management, automated build process, dependency analysis, automated release process, automated testing, continuous integration.
- Preserve development environment
  - It can be very hard to recreate the development environment (exact set of compilers, patches, libraries, build tools, etc.) if dismantled and the details are not recorded.

Problems and Pitfalls

- Prioritizing the wrong evolution dimension
- Designing for changes that never happen
  - i.e. building unnecessary complexity and costs into the system

- Overlooking impact of flexibility on other qualities
  - Flexibility sometimes improves but sometimes hinders other qualities such as performance, availability and security
- Overreliance on specific hardware or software
  - Using a specialized third-party component (hardware or software) can facilitate development but may create problems if component becomes unavailable or outdated.
- Ad hoc release management
- Lost development environments

Checklist for Evolution Requirements

- What are the changes your system will most likely have to deal with?
- What are the likelihood and timescale of these changes?

Checklist for Architecture Definition

- How well does your architecture cope with each of these changes? What architectural elements will be impacted? What is the effort involved?
- What design principles are you using to facilitate these changes?
- What are the justification for these principles?
- What are the impacts on cost and other qualities?

Cost

Architectural Tactics (not covered explicitly in the book)

- Check that you are not over-engineering
  - Can you simplify functionality?
  - Can you weaken the quality requirements?
  - Can you simplify the architecture?
- Reuse existing assets
- Reduce hardware cost through better software performance and reliability
- Reduce administration cost
  - provide efficient administration support functions
  - reduce number of servers and geographical locations to be serviced

Summary: Architecture and Quality Requirements

- 4 important quality requirements
  - Security
  - Performance and scalability
  - Availability and resilience
  - Evolution
- Architectural perspectives: guidelines for
  - defining quality requirements
  - evaluating architecture models with respect to quality requirements
  - improving the architecture to support the quality requirements
- Other perspectives: accessibility, internationalization, usability, development resources, …