ARCHITECTURAL PERSPECTIVES

OBJECTIVES

- explain what architectural perspectives are
- describe the main four perspectives
- how to apply perspectives to views
- understand the value of architectural patterns and tactics

CONTENTS

- **architectural perspectives**
- applying perspectives to views
- architectural tactics & patterns

QUALITY PROPERTIES

- so far we have been identifying structures : what the system had to do
- not considering how system had to do it
- the "how" is referred to as the "quality properties" of the system

QUALITY PROPERTIES

QUALITY PROPERTIES

- quality properties are the non-functional characteristics of the system
  - performance
  - efficiency
  - security
  - maintainability
  - availability
  - and many more ..

QUALITY PROPERTIES

- quality properties are crucial to stakeholders
  - slow functions don't get used
  - unavailable systems cause business interruption
  - security problems cause headlines
  - unmaintainable systems become irrelevant
- addressing QPs is key architectural task
  - understanding stakeholder "real" needs
  - trading off between conflicting needs
  - need a framework for thinking about QPs

PERSPECTIVES

- perspectives : collection of patterns, templates and guidelines to ensure the system has the right quality properties
  - a store of knowledge and experience
  - a guide to the architect based on proven practice
- our initial core set
  - performance and scalability
  - security
  - availability and resilience
  - evolution
- also: accessibility, development resource, internationalization, geographical distribution, regulation, usability ..

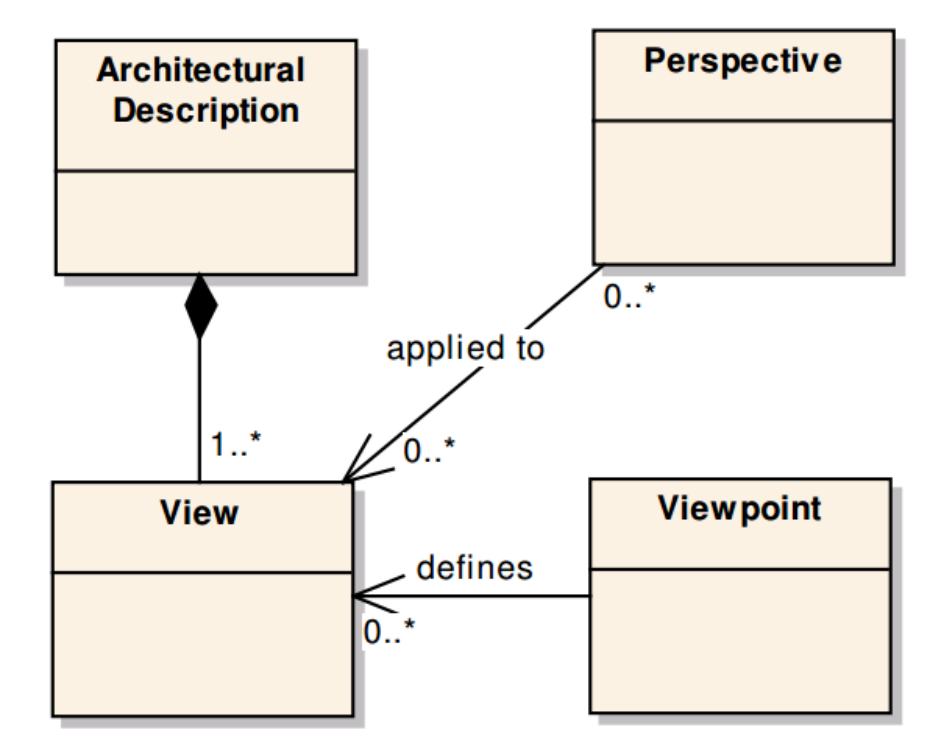
VIEWPOINTS & PERSPECTIVES

	Viewpoint	Perspective
Focus	a type of structure	a quality property
Result	a view – model(s) a primary arch structure	changes to views supporting artefacts
Guidance	models to create advice based on practice	a process for application advice based on practice

VIEWPOINTS & PERSPECTIVES

- you apply perspectives to the architecture to ensure QPs are acceptable and guide its development

e.g. performance and scalability applied to functional view, security applied to information view



DESCRIPTION OF PERSPECTIVES

- **desired quality** : definition
- **applicability** to views : which of your views are most likely to be impacted by applying the perspective
- **concerns** : that the perspective addresses
- **activities** : how to apply the perspective to your architecture
- **tactics** : an established approach or solution you can use
- **problems and pitfalls** : to be aware of

TACTICS

- architectural tactic : an established approach or solution you can use to achieve a particular QP
  - e.g. redundancy increases availability by error handling
- how to apply tactics
  - identify the relevant tactics
  - evaluate (dis)advantages for implementing the key-drivers
  - evaluate their relationship (e.g. contradiction)
  - make appropriate design decisions

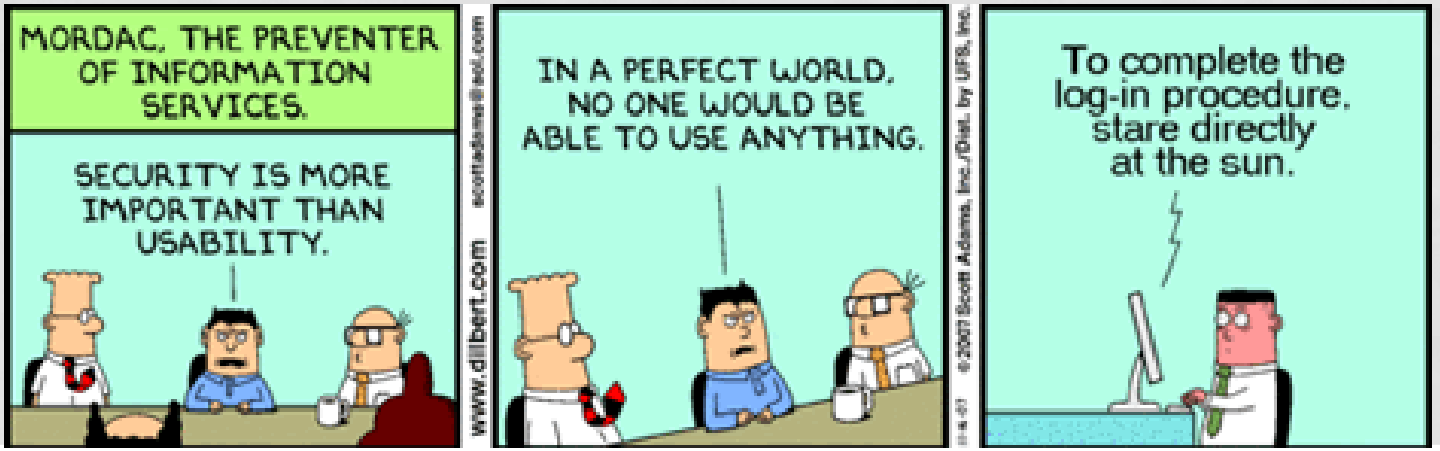
PERFORMANCE AND SCALABILITY

concerns : processing volume, response time, responsiveness, throughput, predictability

tactics : optimize repeated processing, reduce contention via replication, prioritize processing, consolidate related workload, distribute processing over time, minimize the use of shared resources, partition and parallelize, use asynchronous processing, make design compromises

SECURITY

- **concerns** : authentication, authorization, confidentiality, integrity, accountability, availability, intrusion detection, recovery
- **tactics** : apply recognized security principles, authenticate the principals, authorize access, ensure information secrecy, ensure information integrity, vulnerability analysis, application of security technology



AVAILABILITY AND RESILIENCE

- **concerns** : classes of service, planned/unplanned downtime, MTBF, MTTR, disaster recovery, redundancy, clustering, failover

- **tactics** : select fault-tolerant h/w, use h/w clustering and load balancing, log transactions, apply software availability solutions, select fault-tolerant software, identify backup and disaster recoveries solutions

EVOLUTION

- **concerns** : magnitude of change, dimensions of change, likelihood of change, timescale for change, development complexity, preservation of knowledge, reliability of change

- **tactics** : contain/encapsulate change, create flexible interfaces. apply change-oriented architectural styles, build variation points into the software (use patterns), achieve reliable change, preserve development environments, achieve reliable change (configuration management, automated testing, ci/cd)

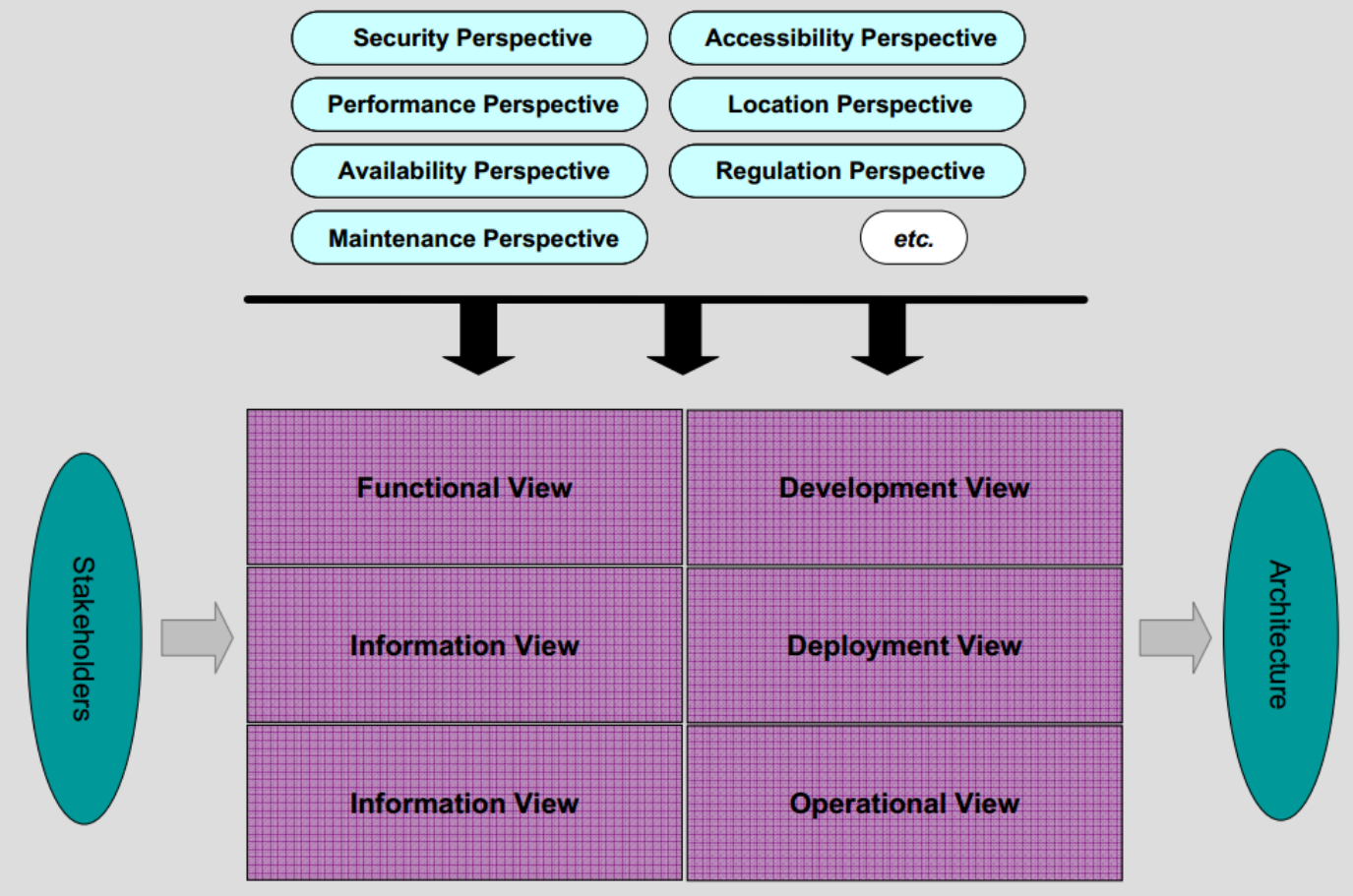
MORE PERSPECTIVES ...

- accessibility
  - can the system be used by people with disabilities?
- development resource
  - can the system be built within people, time, budget constraints?
- internationalization
  - is the system independent of language, country and culture?
- location
  - will the system work, given its geographical distribution (multiple locations) ?
- regulation
  - does the system meet required regulatory constraints?
- usability
  - can people use the system effectively?

CONTENTS

- architectural perspectives
- **applying perspectives to views**
- architectural tactics & patterns

APPLYING PERSPECTIVES TO VIEWS



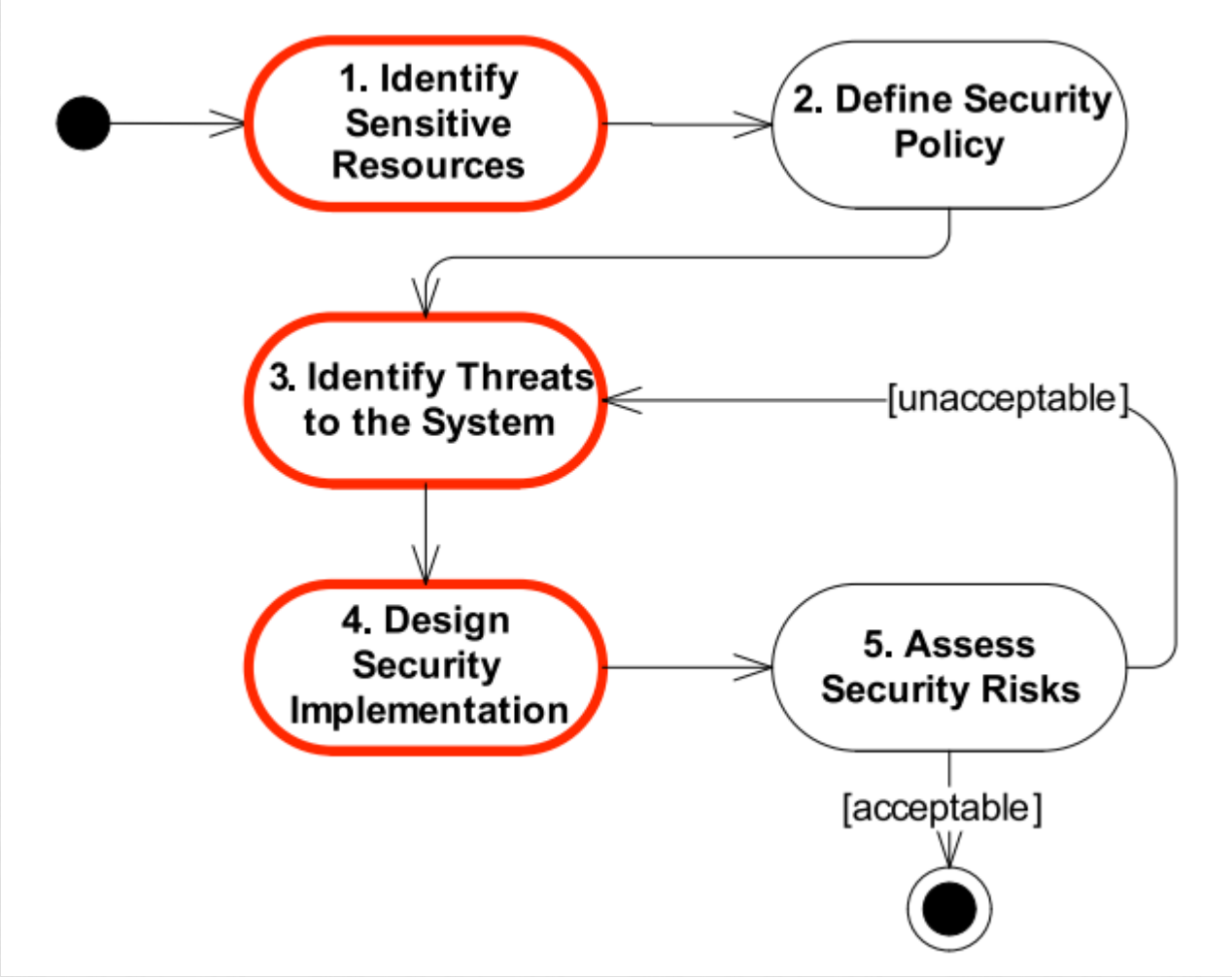
APPLYING PERSPECTIVES TO VIEWS

- which perspectives would help you to achieve your quality properties ?
- where may you have conflicts if applying different relevant perspectives ?

APPLYING PERSPECTIVES TO VIEWS

	Security	Performance	Availability	Evoluion
Operational				
Concurrency		Shared resources, blocking, queuing, coordination		
Information	Access control, access classes, object-level security			
Functional				Extension points, flexible interfaces, meta-approaches

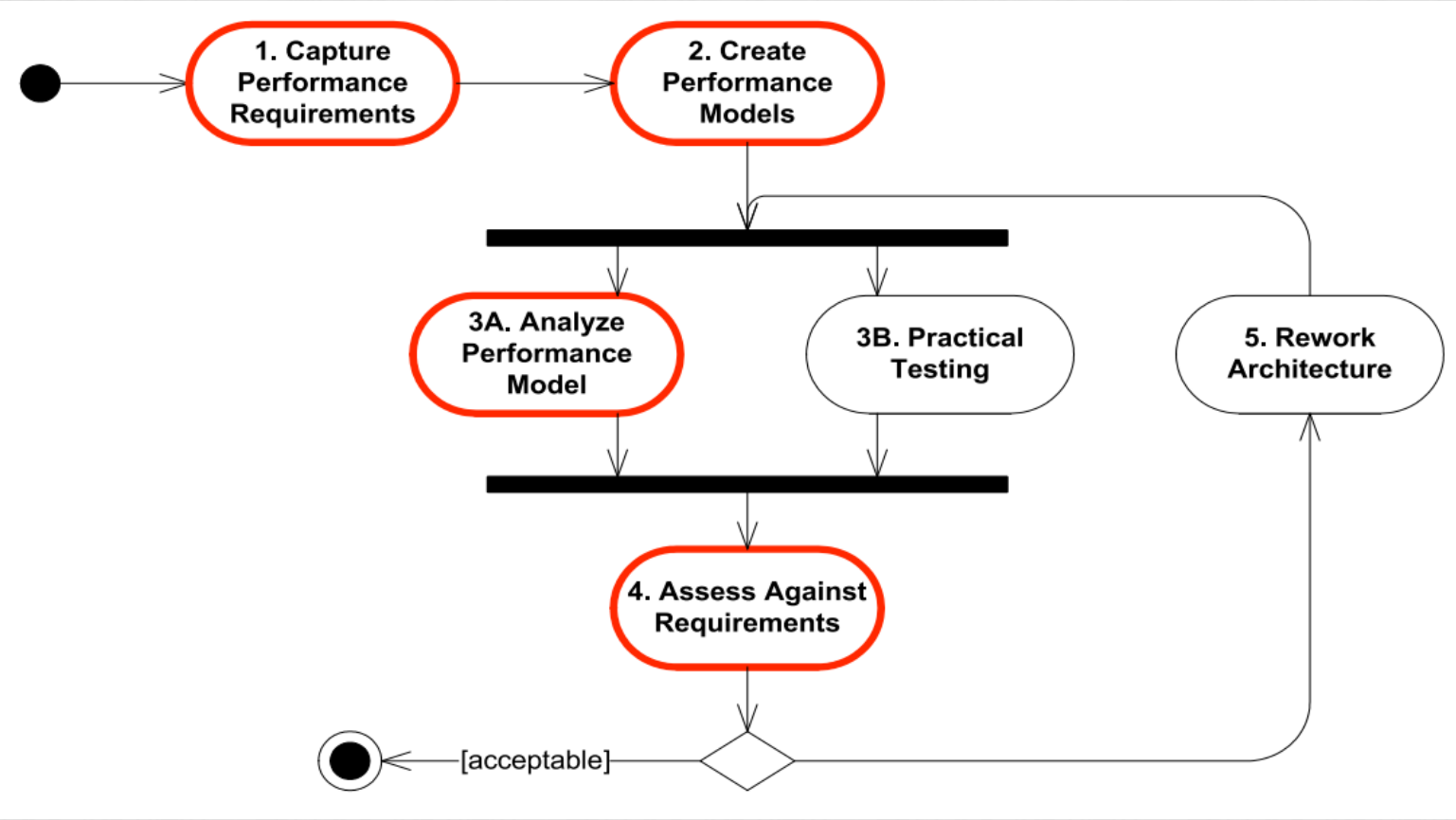
APPLYING SECURITY PERSPECTIVE ("ACTIVITIES")



APPLYING SECURITY PERSPECTIVE ("ACTIVITIES")

- identify sensitive resources
  - e.g. data in the database
- identify security threads
  - operators stealing backups
  - admins querying data, seeing names
  - bribing investigating officers
  - internal attack on the database via network
- design security implementation
  - backups : encrypt data in the database
  - bribery : add audit trail for data access
  - network attacks : harden database, firewall, IDS
  - consequents : more complexity, bad for performance, more operational costs...

APPLYING PERFORMANCE AND SCALABILITY PERSPECTIVE



APPLYING PERFORMANCE AND SCALABILITY PERSPECTIVE

- capture p&s requirements
  - e.g. response times, throughput and scalability
- create performance models
  - what are the key performance metrics ?
  - what are the performance bottlenecks ?
  - models : calculations , statistical models, simulation programs
- analyze models
- assess against requirements

CONTENTS

- architectural perspectives
- applying perspectives to views
- **architectural tactics & patterns**

TACTICS & PATTERNS

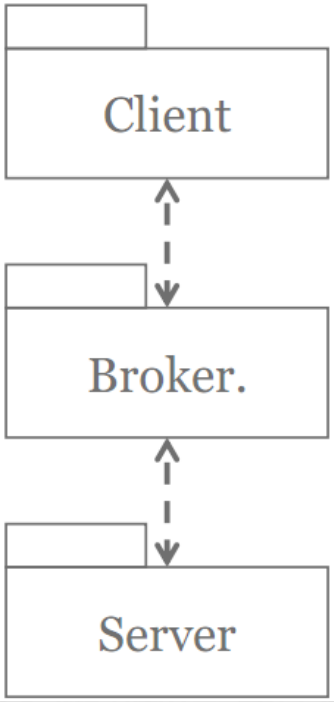
- an architectural pattern often implements multiple tactics, usually related to different qualities, in a coherent way

- architectural tactic : an established approach or solution you can use to achieve a particular QP
  - e.g. redundancy increases availability by error handling

PATTERNS IMPLEMENT TACTICS

Drivers	Modifiability	Reliability
Tactics	**Localize changes**	**Fault detection**
	Prevention of ripple effects	**Fault prevention**
Sub-tactics	**Semantic coherence**	**Heartbeat**
	Information hiding	**Process Monitor**

Broker is a natural solution for reliability & accommodates modifiability



LEVELS OF SOFTWARE PATTERNS

- architecture (system level)
  - pipes & filters
  - layers
  - client/server
  - peer-to-peer
  - publisher/subscriber
  - asynchronous messaging
  - tuple space ...
- design (subsystem level)
  - strategy
  - observer
  - decorator
  - factory, ...
- language idioms (block level)
  - how to loop through a list of items in Java
  - how to handle exceptions in Java

PATTERN CATEGORIES

- creational patterns
  - involve object instantiation and provide a way to decouple client from objects it needs to instantiate
  - e.g. singleton pattern ensures a class has only one instance
- structural patterns
  - let you compose classes or objects into lager structures
  - e.g. decorator patterns is used in java.io classes
- behavioral patterns
  - concerned with how classes and objects interact and distribute responsibility
  - e.g. observer pattern, a design where observable and observers are loosely coupled

PATTERN CATEGORIES