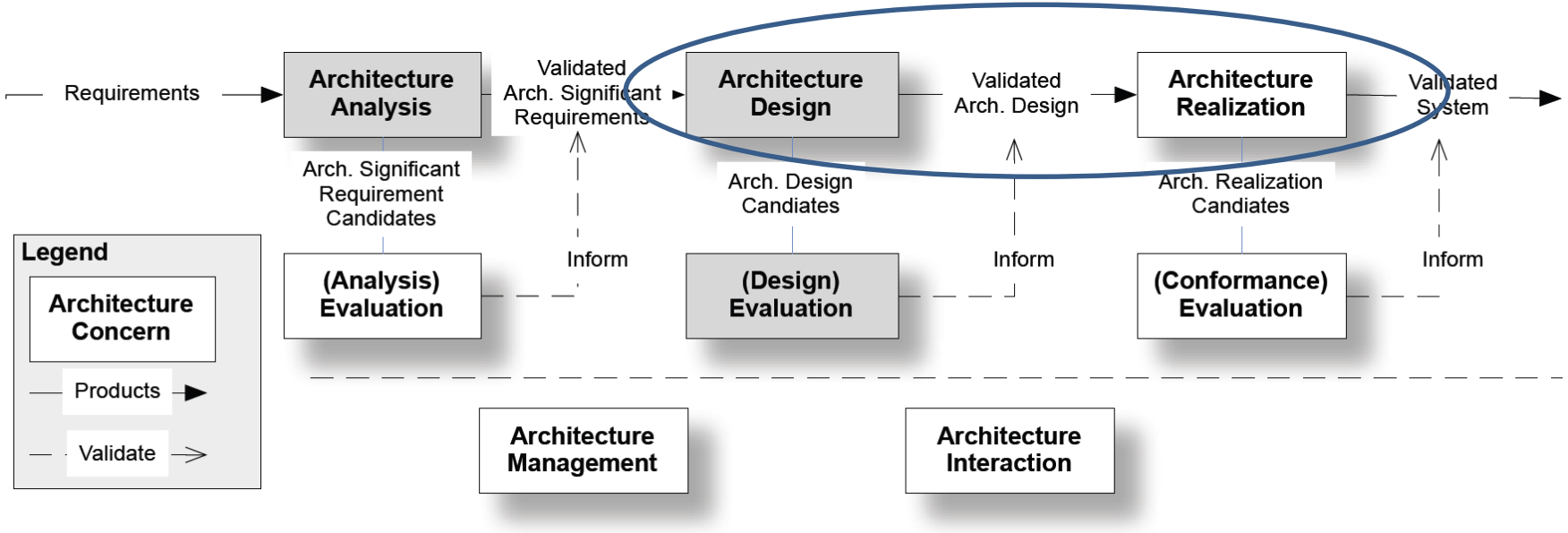
AARHUS UNIVERSITET

Software Architecture in Practice

Tactics

AARHUS UNIVERSITET

Context



AARHUS UNIVERSITET

Introduction

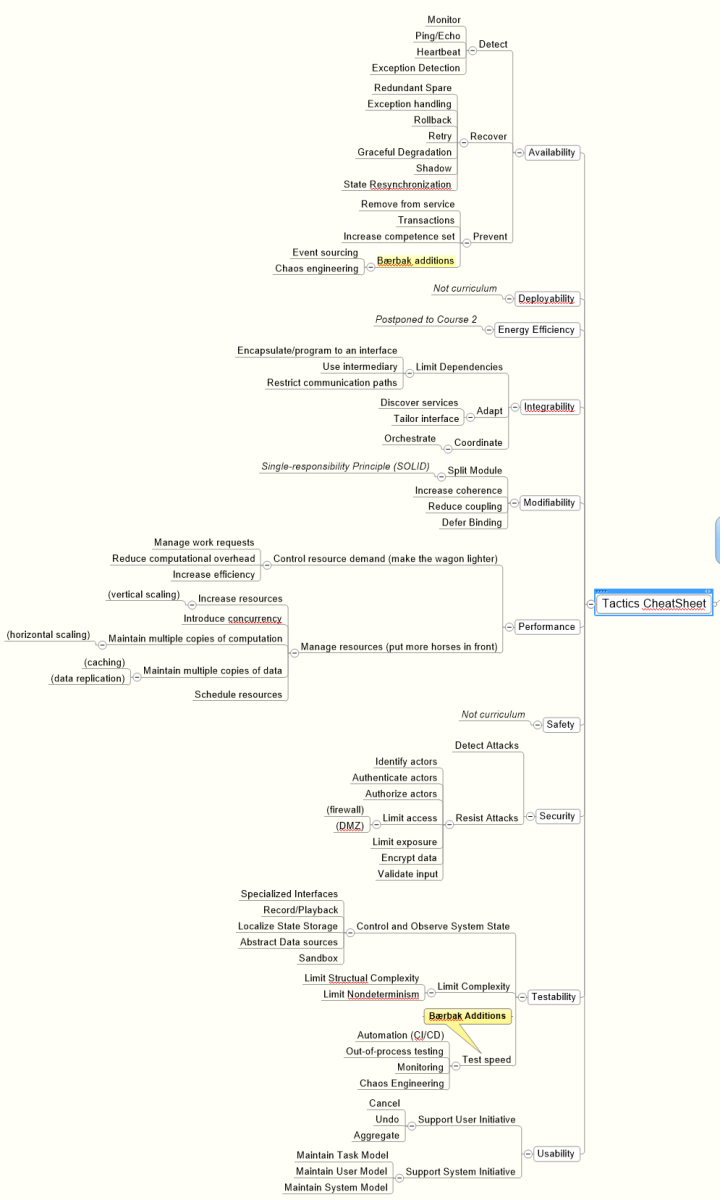
- **Tactics**: A design decision that influences the achievement of a quality attribute response [§3.4]
  - i.e. control the response measure in a positive direction

- The following presentation is *selected picks* from the book
  - Representing my key interests or central tactics
    - Testability, modifiability, availability
  - … and representing areas of my relatively missing expertise
    - Security, safety ☹

The Overview

AARHUS UNIVERSITET

- Well, quite a lot of tactics…



AARHUS UNIVERSITET

Availability

- **Availability**: Property of software that it is ready to carry out its task when you need it to be.

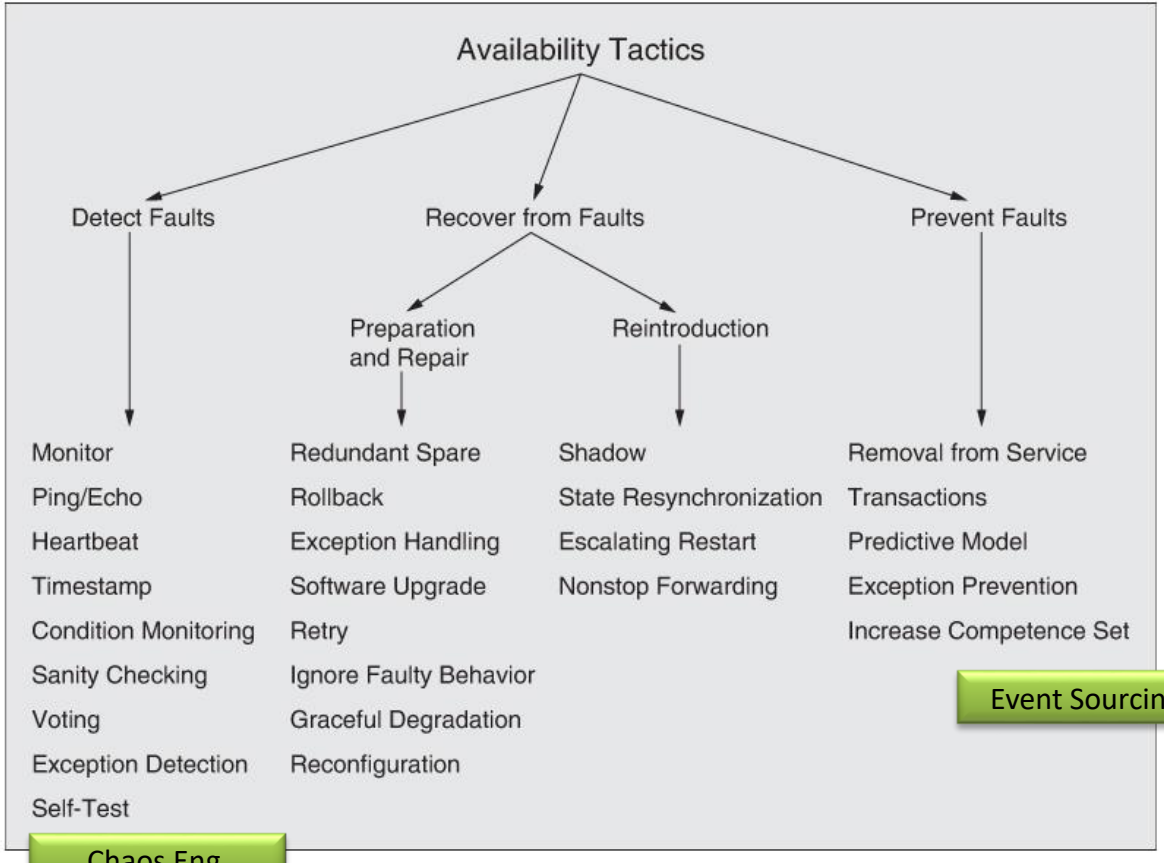
- Lots of metrics…

Table 5.1. System Availability Requirements

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

  - MTBF: Mean Time Between Failures (Airbus 380)
  - MTTR: Mean Time To Repair (NetFlix)
  - MTBF / (MTBF+MTTR)

Tactics

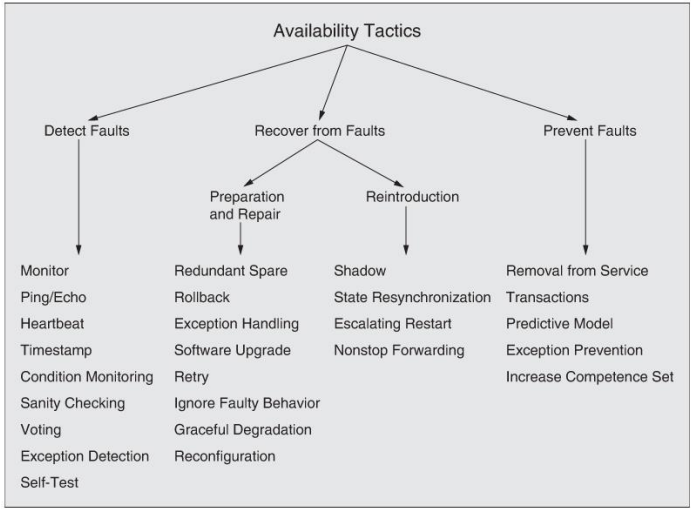

Availability Tactics
Detect Faults
Recover from Faults
Prevent Faults
Preparation and Repair
Reintroduction
Monitor
Ping/Echo
Heartbeat
Timestamp
Condition Monitoring
Sanity Checking
Voting
Exception Detection
Self-Test
Redundant Spare
Rollback
Exception Handling
Software Upgrade
Retry
Ignore Faulty Behavior
Graceful Degradation
Reconfiguration
Shadow
State Resynchronization
Escalating Restart
Nonstop Forwarding
Removal from Service
Transactions
Predictive Model
Exception Prevention
Increase Competence Set
Chaos Eng.
Event Sourcing

Tactics Categories

- Detect Faults
  - Our system needs to know that something has failed!
  - Examples
    - Monitor
      - Overview processes and network to detect anomalies
    - Ping-Echo
      - Send out ‘are you alive’ signals that much be answered
    - Heartbeat
      - MongoDB’s replica set emit heart beats to fellow instances
    - Exception Detection
      - MongoDB’s Java driver will throw an exception in case a write request cannot be fulfilled

Tactics Categories

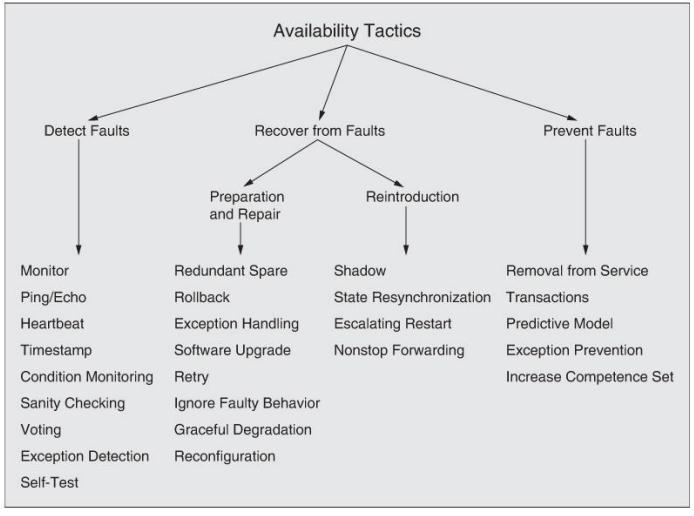
- Recover from Faults / Prep and repair
  - Once detected, how do we get back on track?
  - Examples
    - Redundant Spare
      - Someone to take over if main service fails
    - Rollback
      - Go back in time to known good state
    - Retry
    - Exception handling
    - Graceful Degradation
      - Run with cached data (off-line)



AARHUS UNIVERSITET

Tactics Categories

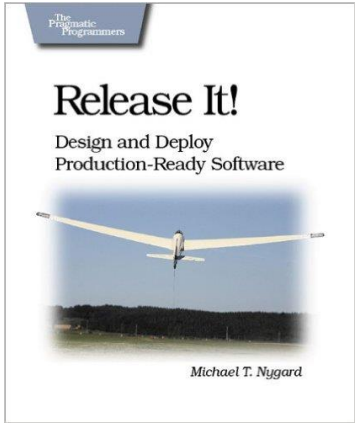
- Recover from Faults / Reintroduction
  - How to get failed components working again
  - Examples:
    - Shadow
      - Failed component operate in ‘shadow’ mode until ‘safe to reintroduce’
    - State resynchronization
      - MongoDB slave 1 needed two days to get back



AARHUS UNIVERSITET

Tactics Categories

- Prevent Faults
  - Can we avoid it in the first place
  - Examples:
    - Transactions: ACID in RDB
    - Increase competence set: Read Nygard's book and employ the techniques ☺ - Safe failure modes
      - (or follow my MSDO course ☺)
    - Removal from service
      - Docker swarm health checks



AARHUS UNIVERSITET

Evolving World

- *Tactics* are of course not a static set, and Bass et al. do not present a comprehensive catalogue.

- One missing tactic is **‘event sourcing’,** I find…
  - (Fowler, 2005) (https://martinfowler.com/eaaDev/EventSourcing.html)
- Definition: *Event Sourcing ensures that all changes to application state are stored as a sequence of events.*
  - I.e. only use **CR**ud of the database CRUD operations.
- Another is *chaos engineering*
  - Will return to that in second course

Summary

- **Availability**: Property of software that it is ready to carry out its task when you need it to be.

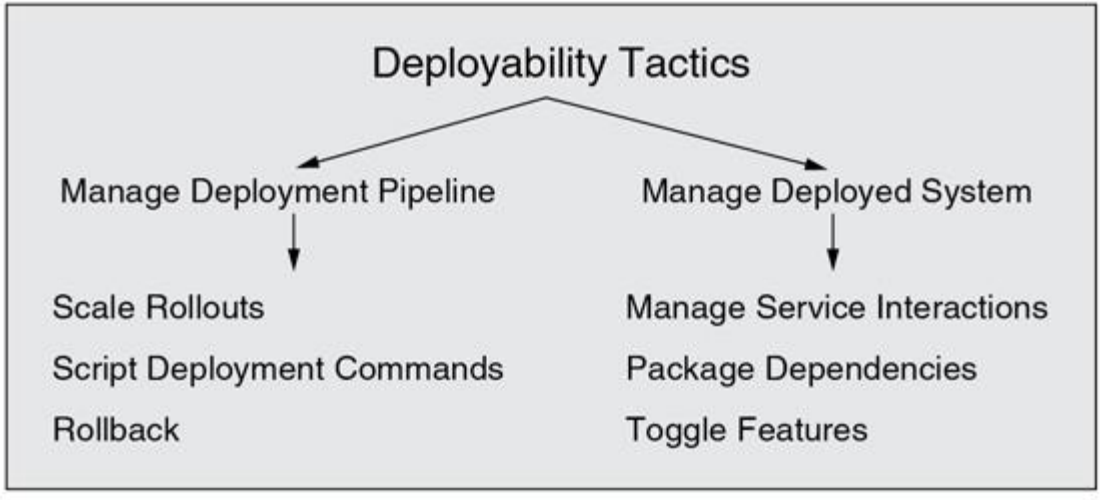
- There are *lots* of tactics
  - Some are simple, directly in programming language
    - Like exceptions
  - Some are wildly complex
    - Passive redundancy (classified as a pattern)
  - Look for the complex ones in external libraries

Deployability

- **Deployability:** Concerned with *the time and effort for software to be allocated to an environment for execution*

- Uber data: 5.000 deployments per week
  - That is, revised software deployed every 2 minutes around the clock…

Tactics



- Many of these are covered by a proper Continuous integration / Continuous deployment (CI/CD) pipeline
- Not part of this course. Follow the MSDO fagpakke ☺.

Energy Efficiency

- **Energy Efficiency:** Concerned with the system's ability to conserve/minimize power consumption while providing it's services

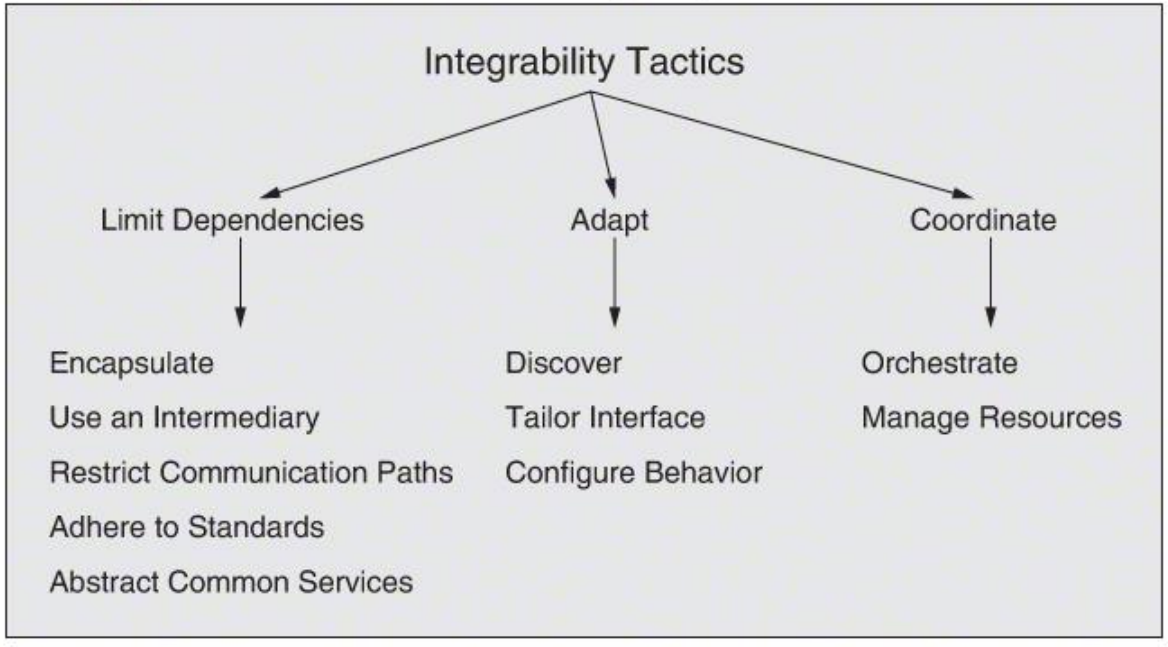
- Will be the focus in the next enkeltfag…

Integrability

- **Integrability:** Concerned with the costs and risks of integrating separately developed components (so the resulting system behaves correctly)

- Modern day development seldom only rely on in-house developed code
  - Maven Repository, NuGet, nmp, … (compile-time)
  - Single sign-on and many other external services (run-time)

Tactics



- Reduce risk and cost of adding new components, reintegrating changed ones, and integrating sets…

AARHUS UNIVERSITET

Tactics Categories

- Limit Dependencies
  - Encapsulate – *Program to an Interface // Façade*
    - Never ever code directly to the concrete API, wrap it in a façade
      - That is: not “SELECT studentId from StudentTable where name=“arne””
      - But “int studentId = dbStrategy.fecthFromName(“arne”)”
  - Use Intermediary – *Decoupling / avoid hard coupling*
    - Pub-sub message broker (decouple producer and consumer)
    - Service discovery (Use DNS to lookup IP, ObjectManager pattern)
    - Adapter pattern (adapt interface)
  - Restrict Communication Paths
    - Clear convention about how components interact
      - Ala ‘program to interface’

AARHUS UNIVERSITET

Tactics Categories

- Limit Dependencies
  - Adhere to standard
    - REST is no just random GET/POST messages! Read the books!
    - Follow this course and name your modules according to patterns and styles
  - Abstract common services
    - *Program to an interface*
      - int studentId = dbStrategy.fecthFromName(“arne”)”
      - *Can be recoded to a Redis DB or a MongoDB in a few hours*
      - Compare this to finding 1.200 places in the code that sends SQL commands to the DB…

AARHUS UNIVERSITET

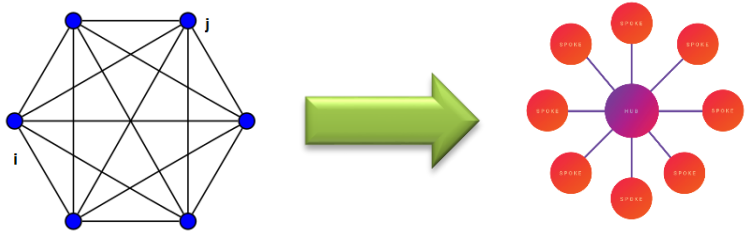
Tactics Categories

- Adapt
  - Discover – *Decouple hard bindings between components*
    - Lookup the actual receiver based upon an abstract, global, name
      - DNS lookup, ObjectManager, dependency injection (abstract factory)
  - Tailor interface - *adapter, decorator, proxy patterns*
    - Add/hide capabilities in existing interface while keeping API stable
    - *Extensible interface*
      - *Add version number in message*
      - *JSON and XML are pretty flexible, you can always add stuff, and older clients may then ignore additional attributes*
  - Configure behavior
    - Make component adaptable (through configuration) to more than one type of communication
      - Like HTTP media type – do you talk JSON or XML?

AARHUS UNIVERSITET

Tactics Categories

- Coordinate
  - Orchestrate - *Mediator pattern*
    - Replace a ‘fully connected graph’ with a ‘hub-and-spoke’ one



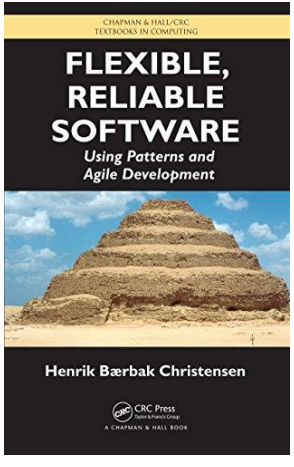
  - Manage resources
    - Use resource managers instead of direct access
      - Like thread pools, database connection pools
    - Can thus control fairness, avoid exhaustion, etc.

Modifiability

- **Modifiability**: Concerned with the ease with which the system supports change

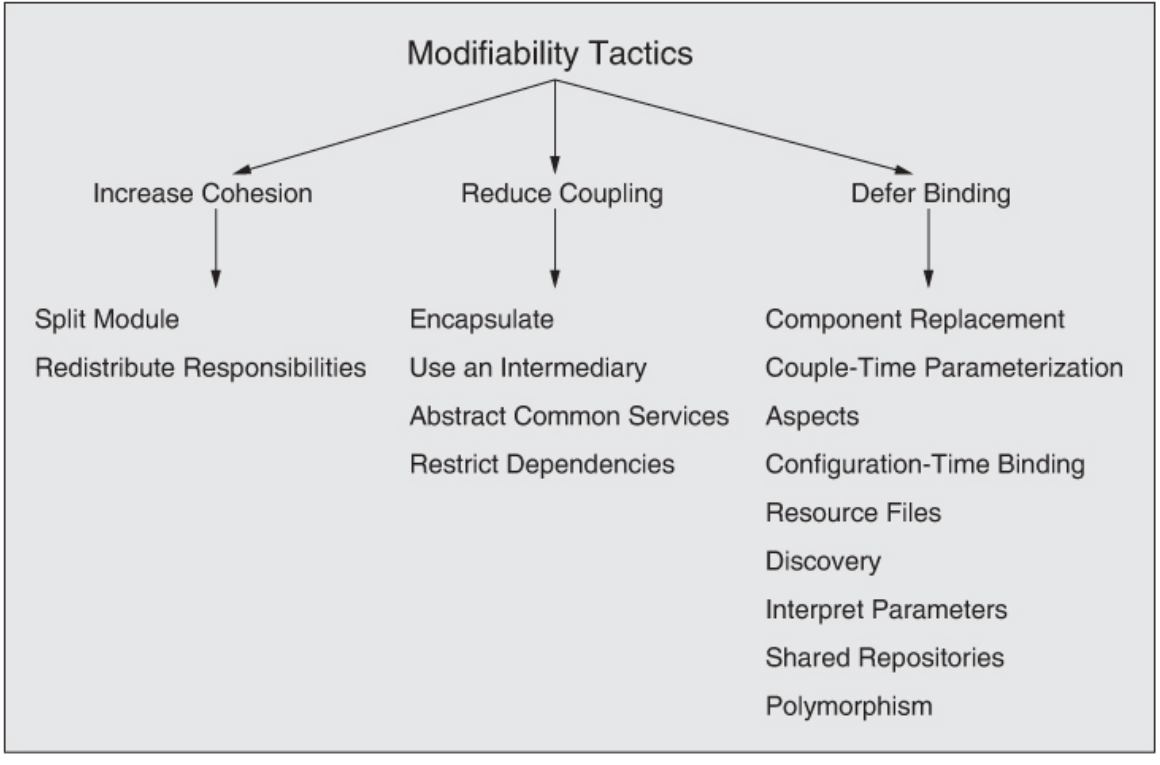
- Focus in many classic software engineering courses and books ☺

- *I actually assume you master these tactics…*

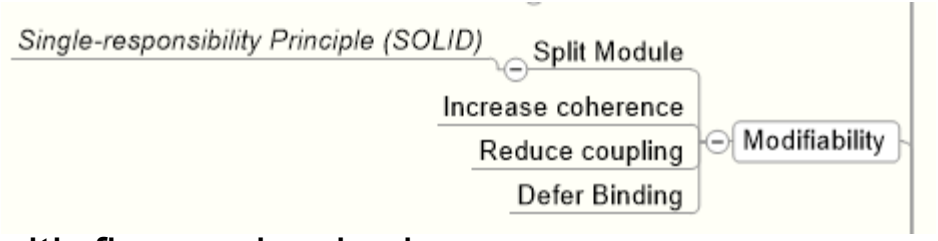


Tactics

- The full set…



Tactics

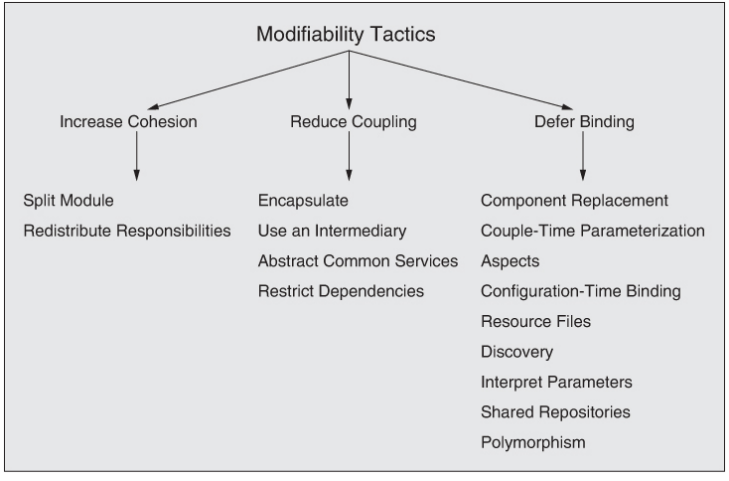


- My top picks
  - Split module
    - Avoid ‘The Blob’, replace with fine-grained roles
    - See my chapter on ‘Role-based design’ / SOLID / Interface-Segregation Principle
  - Increase coherence
    - The old friends ‘coupling and cohesion’ – make small, highly cohesive roles
  - Reduce coupling
    - Avoid fully connected object graphs in favor of hierarchical and/or Mediator designs (hub-and-spokes)
  - Defer binding
    - Use dependency injection to decouple roles

Tactics

- Bærbak: *Program to an interface, Favor object composition…*

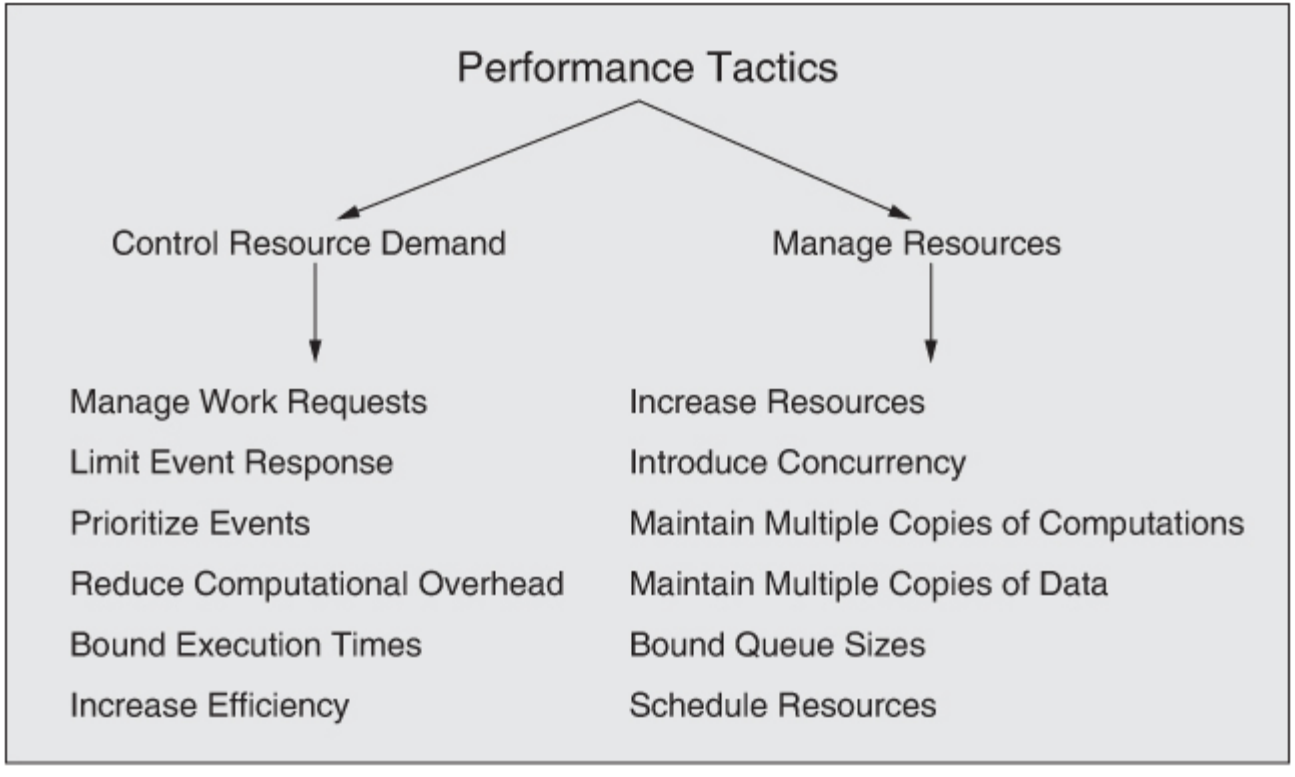
- Many Design Patterns address modifiability …



Performance

- **Performance**: Concerned with ability to meet timing requirements

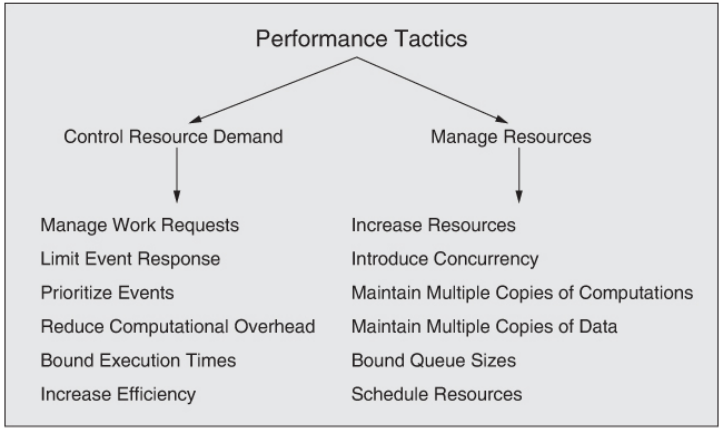
Tactics


Performance Tactics
Control Resource Demand
Manage Resources
Manage Work Requests
Limit Event Response
Prioritize Events
Reduce Computational Overhead
Bound Execution Times
Increase Efficiency
Increase Resources
Introduce Concurrency
Maintain Multiple Copies of Computations
Maintain Multiple Copies of Data
Bound Queue Sizes
Schedule Resources

Tactics

- Either lower demand or increase muscles ☺
  - ***Make the wagon lighter*** *or* ***add more horses in front****…*
- 'Control Demand' examples
  - Increase efficiency: better algorithms
  - Reduce overhead: avoid too much network traffic
- 'Manage resources' examples
  - Increase: Vertical scale (up)
  - Multiple copies data: cache
  - Multiple copies comp:
    - Horizontal scaling (scale out)

Will also return in Course 2



Evolving World

- Quite a few of my MSDO groups looked into
  - CQRS: Command Query Responsibility Segregation

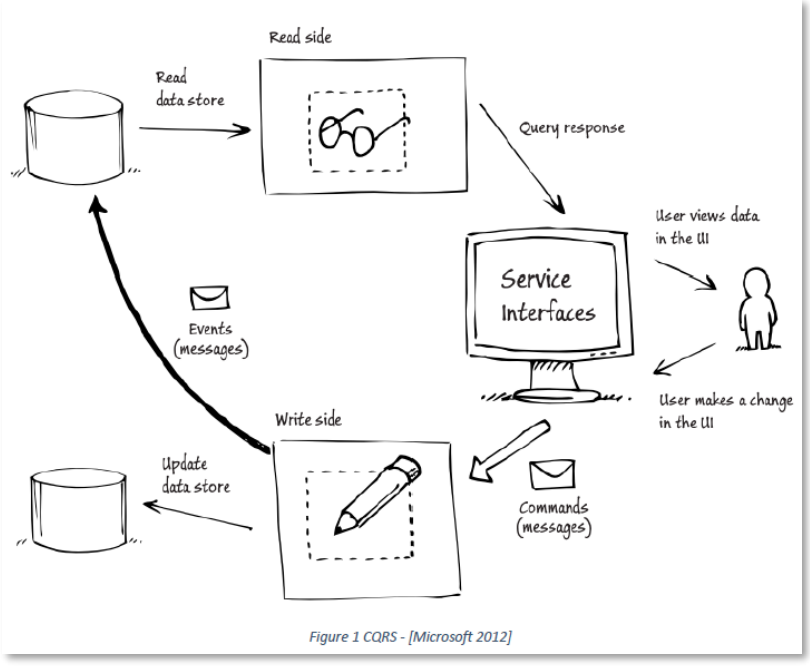


Figure 1 CQRS - [Microsoft 2012]

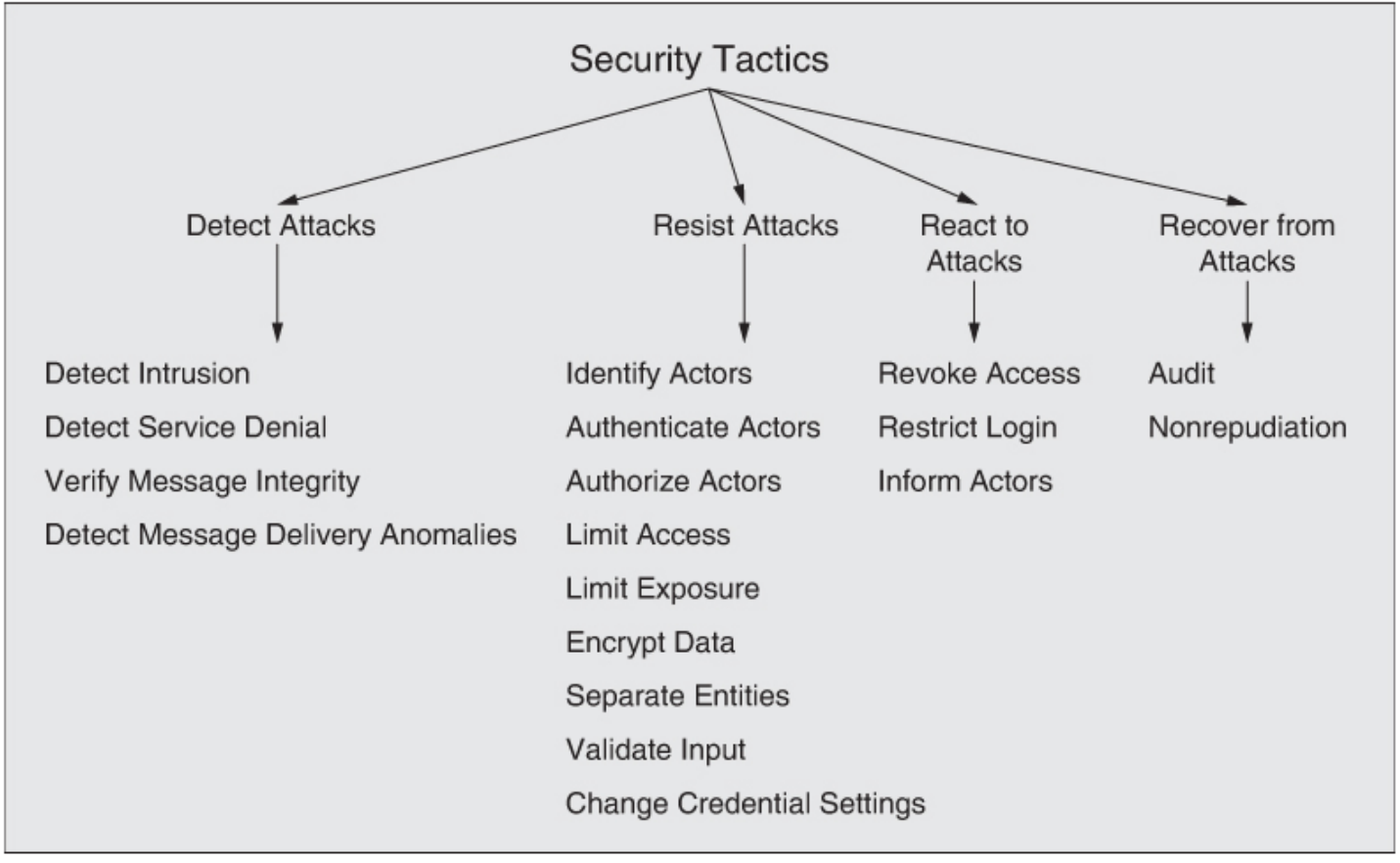
Security

- **Security**: Concerned with ability to protect data and information from unauthorized access while still providing access to people/systems that are authorized

Disclaimer: I have little 'hard' experience in this field...

But Master i IT has quite a few fagpakker on the topic!

Tactics


Security Tactics
Detect Attacks
Resist Attacks
React to Attacks
Recover from Attacks
Detect Intrusion
Detect Service Denial
Verify Message Integrity
Detect Message Delivery Anomalies
Identify Actors
Authenticate Actors
Authorize Actors
Limit Access
Limit Exposure
Encrypt Data
Separate Entities
Validate Input
Change Credential Settings
Revoke Access
Restrict Login
Inform Actors
Audit
Nonrepudiation

Tactics

- Detect Attacks
  - Basically all subtactics are concerned with monitoring the system and comparing behavior with ‘known behavior’ and report anomalies, as indications of ongoing attacks
- Resist Attacks
  - Identify, Authenticate, Authorize actors
    - “login”, allow only access to those authorized to view data
  - Limit access, exposure
    - Firewalls, DMZ, limit access through a single point/API
  - Encrypt data (communication and/or data at rest)

Tactics

- Resist Attacks
  - Validate input
    - Client side input must be sanitized before applied
      - Typical SQL injection attacks

WarStory:
'Validate Input' tactic even for a lecturing web site

  - Change Credential Settings
    - *How many of you have changed the 'admin' password of your home router?*

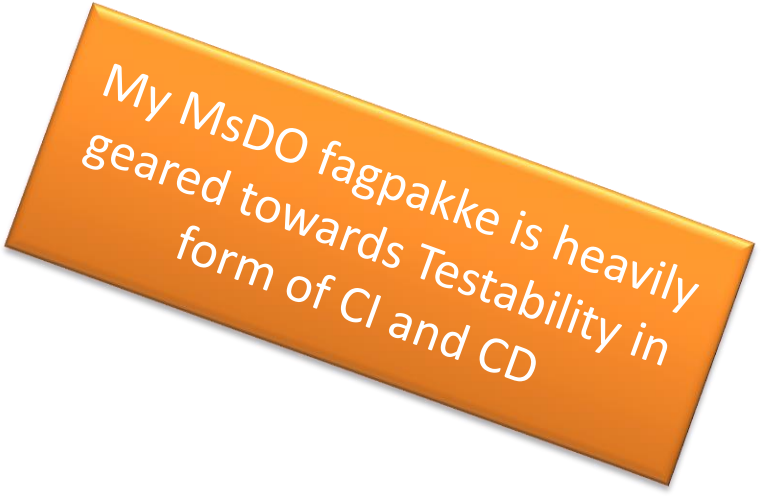
Tactics

- Lots of tactics
  - General rule: **Never do security yourself**
    - You *will* get it wrong, so…
    - Pick commonly accepted best-of-breed technique
      - OpenID, OAuth2, HTTPS, JWT, SAML, SSL, TSL, …
    - *Read the books!!!*
      - Protocols are complex and you *will* get it wrong
  - Rely on reviewed/trusted implementations

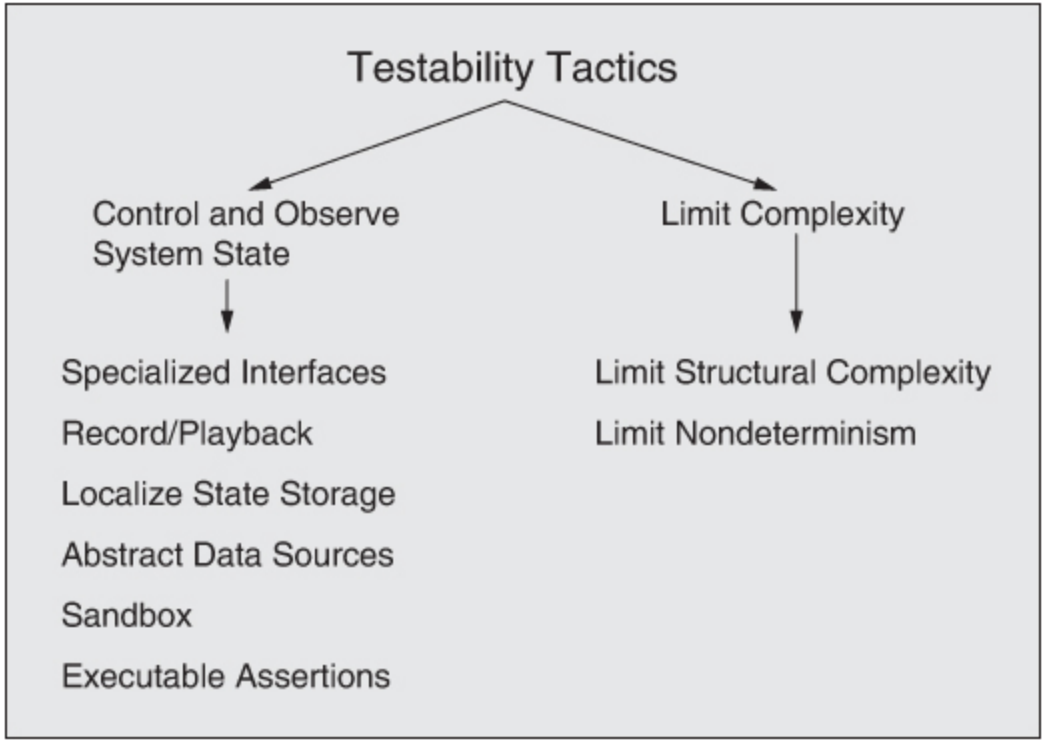
- And many issues are *organizational!*
  - Do not open that spam mail, please…

Testability

- **Testability**: Concerned with the ease with which the software can be made to demonstrate its faults

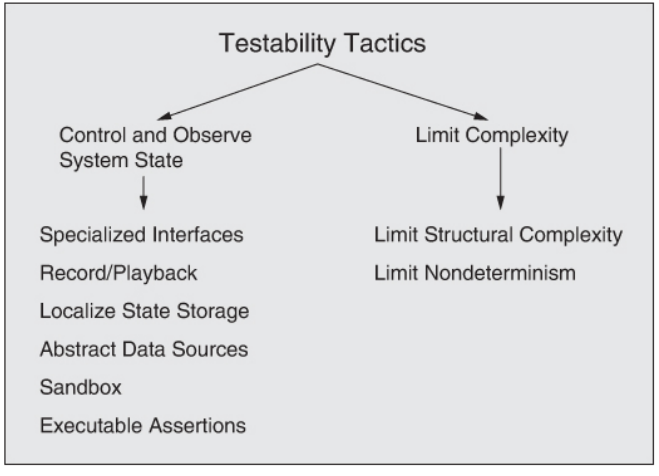


Tactics


Testability Tactics
Control and Observe System State
Limit Complexity
Specialized Interfaces
Record/Playback
Localize State Storage
Abstract Data Sources
Sandbox
Executable Assertions
Limit Structural Complexity
Limit Nondeterminism

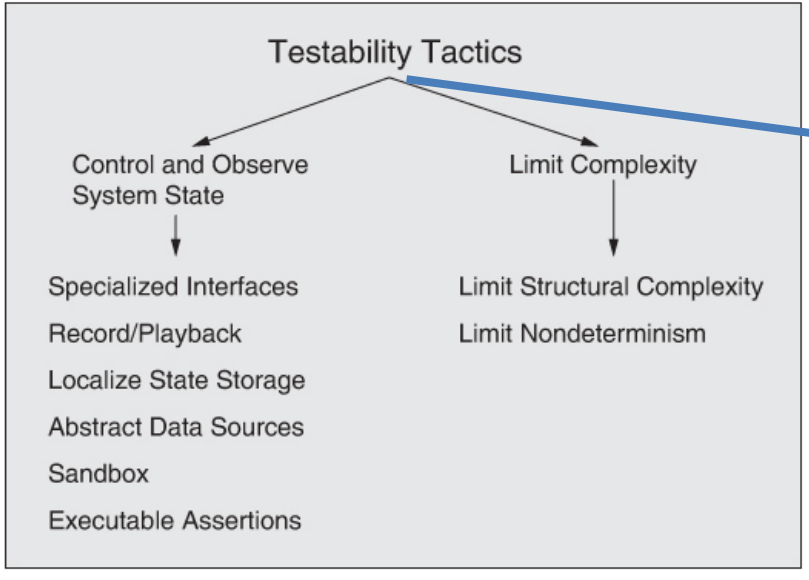
Tactics

- Specialized interfaces (Expansion Interface POSA 4)
  - Tests often deliberately breaks encapsulation!
    - Allow it, but only through dedicated test interfaces!
- Sandboxing: *isolate from real world*
  - *Program to an interface* and allow test doubles to be injected instead of DBs, external hardware, etc.
  - Use virtual machines
    - Staging environments
      - Docker swarm, Kubernetes, … ☺
- Limit complexity
  - *Program to interface, favor object composition* ☺



Evolving World

- Also here I find a tactic missing (or a category)



‘Ease of demonstrate faults’ equals **speed!**

Test Process
- Automation (CI/CD)
- Out-of-process testing
- Monitoring
- Chaos Engineering

Tactic

- Test Process Tactics
  - **Automation:** Ensure that tests are executed automatically/programmatically, not by hand
    - xUnit frameworks
    - Continuous Integration servers on dedicated branches
  - **Out-of-process testing:** Ensure your automated tests can test *integration with remote services*.
    - Service Doubles, stubbing, docker automation, …
    - Test Containers: Spawning docker containers under Junit control

Tactic

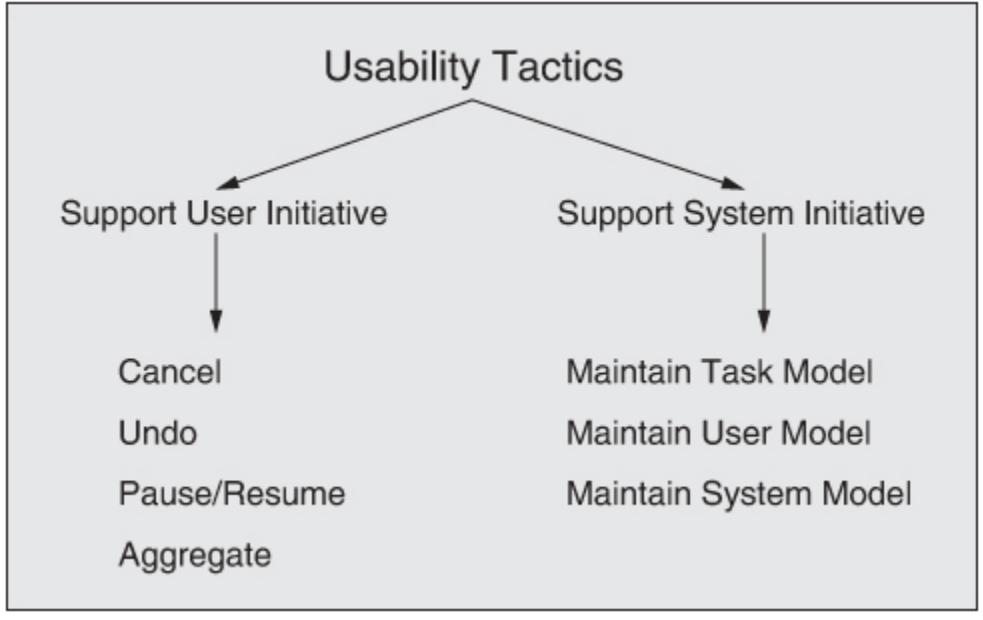
- Test Process Tactics
  - **Monitoring:** Monitor production systems and report anomalies
    - Monitor log messages
  - **Chaos Engineering:** Simian army to produce failure conditions in prod., workshops to brainstorm and test failure situations

”My suggestions” are of course not part of exam curriculum!

Usability

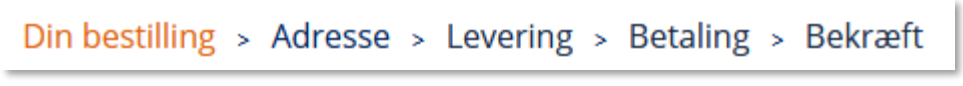
- **Usability**: Concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides

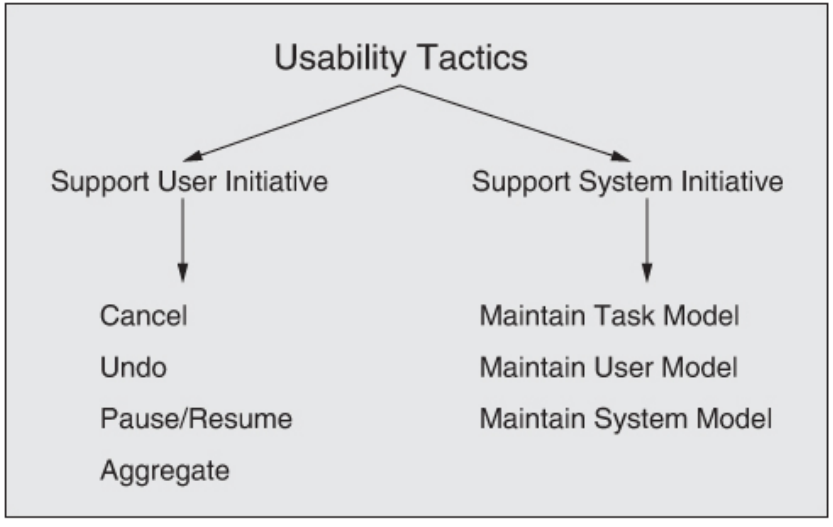
Tactics


Usability Tactics
Support User Initiative
Support System Initiative
Cancel
Undo
Pause/Resume
Aggregate
Maintain Task Model
Maintain User Model
Maintain System Model

Tactics

- Undo and cancel ! Record macros
- Model Task, User, System
  - In a way that allows users to learn and experiment
    - Ex: Buy item in web shop = Task model is a sequence of steps
      - Shipping options
      - Shipping address
      - Billing





Summary

Definition: **Tactic** is a design decision that influences the achievement of a quality attribute response

- Some are high level architectural decisions
  - Redundant spare, …
- Some are low level programming decisions with strong architectural influence
  - Reduce module size, introduce concurrency, …

AARHUS UNIVERSITET

My Architecture Perception

- Some say
  - ”Architecture is the ability to draw 7 boxes with lines between them”
- I say
  - ”Architecture is any decision that influence QAs”
  - Which leads to the corollary
    - *Architectural work span the entire spectrum of decisions from the highest level all the way down to programming statement level!*
- Klaus Marius Hansen, Architect at Microsoft
  - ”I spent quite a lot of time programming architectural prototypes…”

AARHUS UNIVERSITET

Training?

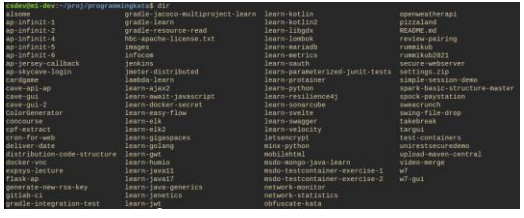
- You need to train to become an expert…
  - From Japan and martial arts, a kata is an individual training exercise, where the emphasis lies on proper form and technique.
  - *I myself do quite a lot of 'programming kata's*

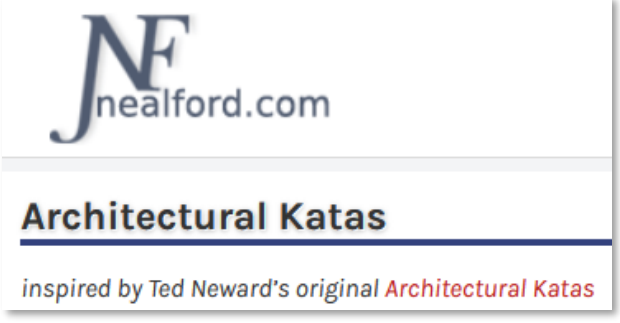
"How do we get great designers?
Great designers design,
of course."

Fred Brooks

"So how are we supposed to get great architects, if they only get the chance to architect fewer than a half-dozen times in their career?"

Ted Neward





Overview

- Cheat sheet I

- *Find it on the week plan*