Progark

Notater

Part one: Introduction

- **Chapter 1:** What is software architecture?
- **Chapter 2:** Why is software architecture important?
- **Chapter 3:** The many contexts of software architecture

Chapter 1 - What is Software Architecture?

1.1 - What is Software Architecture Is and What It Isn't

- **Definisjon of software architecture:** "The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both"

- **Architecture is a set of software structures:**
  - Structure: is simply a set of elements held together by a relation.
  - There are three categories of architectural structures:
    - Modules
    - Component-and-connector
    - Allocation
  - A structure is architectural if it supports reasoning about the system and the system's properties. The reasoning should be about an attribute of the system that is important to some stakeholder. These include **functionality** achived by the system, the **availability** of the systemin the face of faults, the difficulty of making **spesific changes** to the system, the **responsiveness** of the system to user requests, and many others.

- **Architecture is an abstraction:**
  - An architecture is foremost an abstraction of a system that selects certain details and suppresses others
  - An abstraction is essential to tainming the complexity of a system. We simply cannot, and do not want to, deal with all of the complexity all of the time.

- **Every software system has a software architecture:**
  - Every system can be shown to compromise elements and relations among them to support some type of reasoning. In the most trivial case, a system is itself a single element - an uninteresting ans probably no useful architecture, but an architecture nevertheless.

- **Architecture includes behaviour:**
  - The behaviour of each element is part of the architecture insofar as that behavior can be used to reason about the system. This behaviour embodies how elemnts interact with each other, which is clearly part of our definition of software architecture.

- **Not all architectures are good architectures.**

- **System and enterprise architecture (OBS, boka legger ikke vekt på disse)**
  - System architecture: A systems architecture, re is a representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and a concern for the human interaction with these components. That is, system architecture is concerned with a total system, including hardware, software and humans.
  - Enterprise architecture: is a description of the structure and behavior of an

organization's processes information flow, personnel, and organizational subunits, aligned with the organization's core goals and strategic direction.

1.2 - Architectural structures and views

- **Structures and views**
  - View: is a representation of a coherent set of architectural elements, as written by and read by system stakeholders. It consists of a representation of a set of elements and the relation among them.
  - Stucture: is the set of elements itself, as they exist in software or hardware.

- **Three kind of structures**
  - Module: embody decisions as to how the system is to be structured as a set of code or data units that have to constructed pr procured.
  - Component-and-connector: embody decisions as to how system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).
  - Allocation: embody decisions as to how the system will relate to nonsoftware structures in its environment (such as CPU's, file systems, networks, devlopment teams, etc..)

- **Structures provide insight**

- **Some useful module structures:**
  - Decomposition structure: The units are modules that are related to each other by the is-a-submodule-of relation, showing how modules are decomposed into smaller modules recursively until the modules are small enough to be easily understood.
  - Uses structure: In this important but overlooked structure, the units here are also modules, perhaps classes. The units are related by the "uses" relation, a specialized form of dependency.
  - Layer structure: The modules in this structure are called layers. A layer is an abstract virtual machine that provides a cohesive set of services through a managed interface.
  - Class structure: The module units in this structure are called classes. The relation is "inherts from" or "is an instance of".
  - Data model: describes the static information structure in terms of data entities and their relationships. For example, in a banking system, entities will typically include Account, Customer, and Loan.

- **Some useful component-and-connector structures**
  - Service structur:
  - Concurrency structure:

- **Some useful allocation structures**
  - Deployment structure: shows how software is assigned to hardware processing and communication elements.
  - Implementation structure: this structure shows how software elements (usually modules) are mapped to the file structure(s) in the system's development, integration, or configuration control environments.
  - Work assignment structure: this structure assigns responsibility for implementing and

integrating the modules to the teams who will carry it out.

- **Relating structures to each other**

- **Fewer is better**

- **Which structures to choose?**

	Software Structure	Element Types	Relations	Useful For	Quality Attributes Affected
Module Structures	Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control	Modifiability
	Uses	Module	Uses (i.e., requires the correct presence of)	Engineering subsets, engineering extensions	"Subsetability," extensibility
	Layers	Layer	Requires the correct presence of, uses the services of, provides abstraction to	Incremental development, implementing systems on top of "virtual machines"	Portability
	Class	Class, object	Is an instance of, shares access methods of	In object-oriented design systems, factoring out commonality; planning extensions of functionality	Modifiability, extensibility
	Data model	Data entity	{one, many}-to-{one, many}, generalizes, specializes	Engineering global data structures for consistency and performance	Modifiability, performance
C&C Structures	Service	Service, ESB, registry, others	Runs concurrently with, may run concurrently with, excludes, precedes, etc.	Scheduling analysis, performance analysis	Interoperability, modifiability
	Concurrency	Processes, threads	Can run in parallel	Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed	Performance, availability
Allocation Structures	Deployment	Components, hardware elements	Allocated to, migrates to	Performance, availability, security analysis	Performance, security, availability
	Implementation	Modules, file structure	Stored in	Configuration control, integration, test activities	Development efficiency
	Work assignment	Modules, organizational units	Assigned to	Project management, best use of expertise and available resources, management of commonality	Development efficiency

1.3 - Architectural patterns

- **Architectural patterns:** provide packed strategies for solving some of the problems facing a system.

- **Module type pattern:**
  - Layered pattern:

- **Component-and-connector pattern:**
  - Shared-data (or respository) pattern: comprises components and connectors that create, store, and access persistent data.
  - Client-server pattern: The components are the clients and the servers, and the connectors are protocols and messages they share among each other to carry out the system's work..

- **Allocation patterns:**
  - Multi-tier pattern: how to distribute and allocate the components of a system in distinct subnets of hardware and software, connected by some communication medium.
  - Competence center and platform: which are patterns that specialize a software system's work assignment structure.

1.4 - What makes a good architecture?

Chapter 2 - Why is software architecture important?

1. An architecture will inhibit or enable a system's driving quality attributes.
2. The decisions made in an architecture allow you to reason about and manage change as the system evolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances communication among stakeholders.
5. The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. An architecture defines a set of constraints on subsequent implementation.
7. The architecture dictates the structure of an organization, or vice versa.
8. An architecture can provide the basis for evolutionary prototyping.
9. An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.
10. An architecture can be created as a transferable, reusable model that forms the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.
12. By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.
13. An architecture can be the foundation for training a new team member.

2.1 - Inhibiting or enabling a system's quality attributes

- **Hvilke kvalitetsattributter vil man oppnå?** man må legge opp arkitekturen etter hvilke kvalitetsattributter man ønsker å oppnå. Ønsker man at et system skal ha høy sikkerhet, så må man legge arkitekturen deretter osv.
- **En arkitektur er ikke en suksessformel i seg selv:** Om man har en bra arkitektur så er det ikke en selvfølge at det er bra funksjonalitet og kvalitet. Når man skal evaluere må man se på hele livssyklusen fra planlegging (arkitektur) til implementasjon.
- **Det er viktig å se det store bildet:** En god arkitektur er nødvendig, men er ikke den eneste faktoren som påvirker kvaliteten. Kvalitetattributene må bli tatt hensyn til gjennom hele designprosessen, implementasjonsfasen og sluttfasen (deployment). Ingen

2.2 - Reasoning about and managing change

- 3 forskjellige typer endringer i programvare:
    - **A local change:** modifying a single element.
    - **A nonlocal change:** requires multiple element modifications (feks a new feature that affects multiple elements).
    - **An architectural change:** affect the fundamental ways in which the elements interact with each other and will probably require changes all over the system (feks changes in the architecture pattern)

2.3 - Predicting system qualities

- **Bestemme design før utvikling:** ved å bestemme arkitektur på forhånd kan man ofte forutse hvilke kvalitetsattributter man kan oppnå. Som sagt garanterer ikke en arkitektur at man oppnår kvalitetsattributter, men den er en veiledenede retning (feks. ved å følge et pattern så vet man hva man oppnår i stedet for å ta random designvalg der man ikke vet hvor man ender opp).

2.4 - Enhancing communication among stakeholders

- **Hvordan man kommuniserer:** når man kommuniserer med en kunder kan man ikke snakke om tekniske løsninger på fagspråk siden kunden mest sannsynlig ikke har noe teknisk erfaring.
- **Hvordan bruke arkitekturen i kommunikasjon med kunde?** En arkitektur er en abstraksjon av selve systemet som gjøre det lettere å ha oversikt over store komplekse systemer. En kunde har ofte en del krav og bekymringer til systemet, og derfor kan en overordnet oversikt (arkitektur) ofte hjelpe med kommunikasjonen mellom arkitekten og kunden.
- **Hvilke bekymringer har en kunde i forhold til system?**
    - **Budsjett:** er arkitekturen rimelig i forhold til oppsatt budsjett?
    - **Mål:** får kunden oppfyllet alle målene/kravene ?
    - **Generelt funksjonalitet**
    - **Tidsbruk:** går prosjektet som planlagt eller ligger det dårlig ann tidsmessig.

2.5 - Carrying early design decisions

- **Sett av besluttninger.** Vi kan se på arkitekturen som et sett av besluttninger. Feks når man maler et bilde så vil første malingsstrøk påvirke sluttresultatet.
- **Ta gode besluttninger tidlig:** Poenget er at det som bestemmes tidlig ikke er like lett å endre senere i utviklingssyklusen, derfor må gode valg bli tatt tidlig fordi det danner alt grunnlag.
Feks hvis man maler med akryldmaling, men heller ville hatt vannbasert maling, så vil det bli vanskelig å endre fordi malingen er tørket og ikke kan fjernes.

2.6 - Defining constraints on an implementation

- **Sett av elementer:** Implementasjonen må bli implementert som er sett av beskrevede elementer. Hver beskrivelse av elementene er en begrensning på implementeringen.

2.7 - Influencing the organizational structure

- **WBS:** tildele forskjellige grupper ansvaret for å utvikle en del av systemet. Man deler ofte et system opp i en work-breakdown structure. Da er det naturlig at man deler opp "arbeidsgrupper" etter denne strukturen, der hver gruppe får ansvaret for hvert sitt subsystem.

2.8 - Enabling evolutionary prototyping

- **Prototyping:** når man har definert en arkitektur kan man begynne å lage en prototype. En prototype ("skeletal system") viser litt infrastruktur - hvordan elementene initialiserer, kommuniserer, deler data, aksesserer ressurser, reporterer feil, aktivitetslog, etc.
- **Reduserer risiko** med prototyping

2.9 - Improving cost and schedule estimates

- Estimater av Kostnader og tidsplan estimater er viktige verktøy for prosjektmaganger.
- **Estimere kostnader:** Top-down (arkitektens og prosjekt managerens estimat på kostnader) og bottum-up (utviklerenes estimat på kostnader). Begge sider er viktige estimater!

2.10 - Supplying a transferable, reusable model

- **Gjenbruk er en fin ting:**
    - code kan bli gjenbrukt → spare tid + penger
    - kravene som underbygger arkitekturen → sikkerhet
    - Erfaring fra infrastruktur → sikkerhet
    - "Early-decisions" konsekvenser → kan stille en større garanti til noe som allerede er testet og kan da med trygget ta gode tidlige besluttninger.
- **Software produkt line**: produkter med samme gjenbruk av komponenter.

2.11 - Allowing incorporation of independently developed components.

2.12 - Restricting the vocabulary of design alternatives

- Hvordan velge blant uendelige muligheter? med å være disiplinert kan man ta valg som utelukker andre valg slik at man bare står igjen med et lite utvalg å velge mellom.

2.13 - Providing a basis for training

Chapter 3 - The many contexts of software architecture

Contexts of software architecture:

- **Technical:** what technical role does the software architecture play in the system or

systems of which it's a part?
- **Project life cycle:** how does a software architecture relate to the other phases of a software development life cycle?
- **Business:** how does the presence of a software architecture affect an organization's business environment?
- **Professional:** what is the role of a software architect in an organization or a development project?

3.1 - Architecture in a technical context

- Architectures inhibit or enable the achievement of quality attributes
- Architectures and the technical environment

3.2 - Architecture in a life-cycle context

- Software development:
    - Warerfall development
    - Iterative development
    - Agile development
    - Model-driven development
- Making a business case for the system
- Understanding the architecturally significant requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing ot evaluating the architecture
- Implementing and testing the system based on the architecture
- Ensuring that the implementation conforms to the architecture

3.3 - Architecture in a business contect

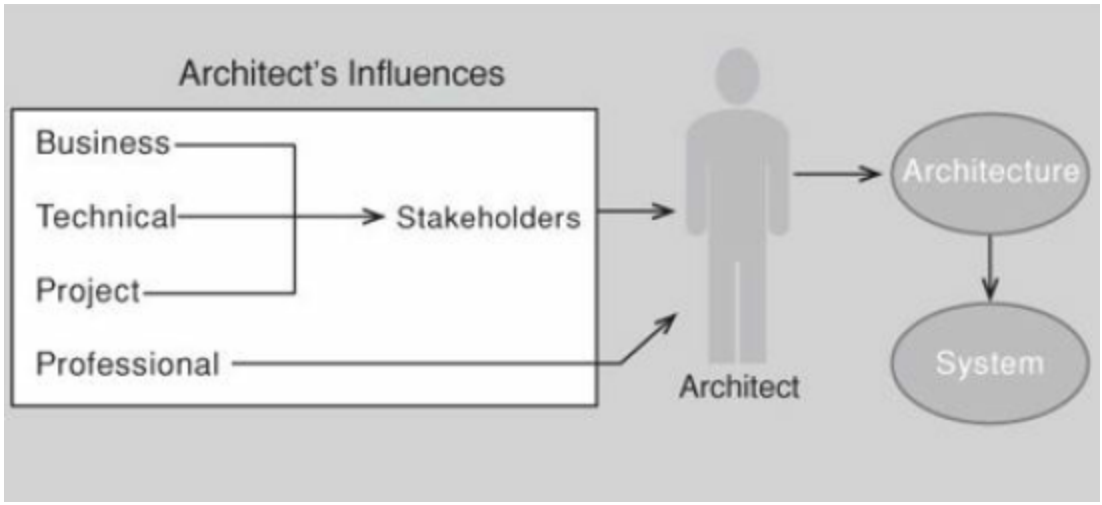
- Architecture and business goals
- Architecture and the development organization

3.4 - Architecture in a professional context

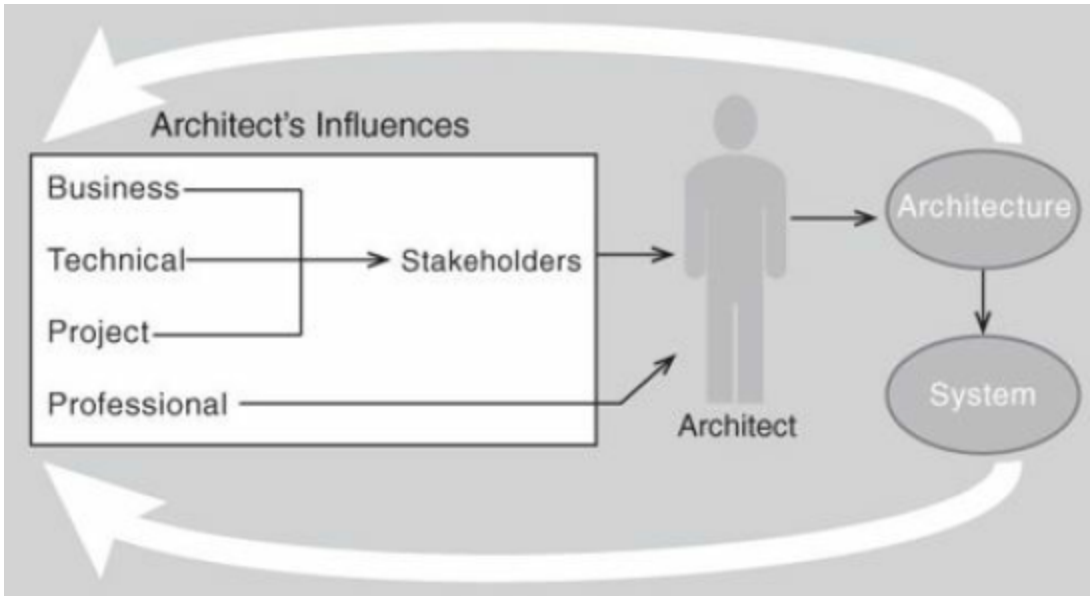
- Architects' background and experience

3.5 - Stakeholders (Se liste)

3.6 - How is architecture influenced



3.7 - What do architectures influence



- Technical context
- Project context
- Business context
- Professional context

Part two: Quality attributes

- **Chapter 4:** Understanding quality attributes
- **Chapter 5:** Availability
- **Chapter 6:** Interoperability
- **Chapter 7:** Modifiability
- **Chapter 8:** Performance
- **Chapter 9:** Security
- **Chapter 10:** Testability
- **Chapter 11:** Usability
- **Chapter 12:** Other quality attributes
- **Chapter 13:** Architectural tactics and patterns
- **Chapter 14:** Quality attribute modeling and analysis

Chapter 4 - Understanding Quality Attributes

Quality Attribute: is a measurable or testable property of a system that is used to indicate how well the system satisfies the need of its stakeholders

4.1 - Architecture and Requirements

- **Requirements for a system:**
    - Functional requirements
    - Quality attribute requirements
    - Constraints

4.2 - Functionality

- First of all, functionality does not determine architecture
- **What if functionality were the only thing that mattered?** then you wouldn't have to divide the system into pieces at all; a single monolithic blob with no internal structure would do just fine. Instead, we design our systems as structured sets of cooperating architectural elements - modules, layers, classes, services, databases,apps, threads, peers, tiers, and on and on.

4.3 - Quality Attribute Considerations

- **Definition:** One definition is that functionality describes what the system does and the quality describes how well the system does its function.
- **There are two categories of quality attributes on which we focus:**
    1. The first is those that describe some property of the system at runtime, such as availability, performance or usability
    2. The second is those that describe some property of the development of the system, such as modifiablity or testability
- **The achivement of one Quality Attribute:** the achivement of one quality attribute will have an effect, sometimes positive and sometimes negative, on the achivements of others.

4.4 - Specifying Quality Attribute Requirements

- The parts of a Quality Attribute scenatio:
  - Source of stimulus
  - Stimulus
  - Environment
  - Artifact
  - Response
  - Response measure

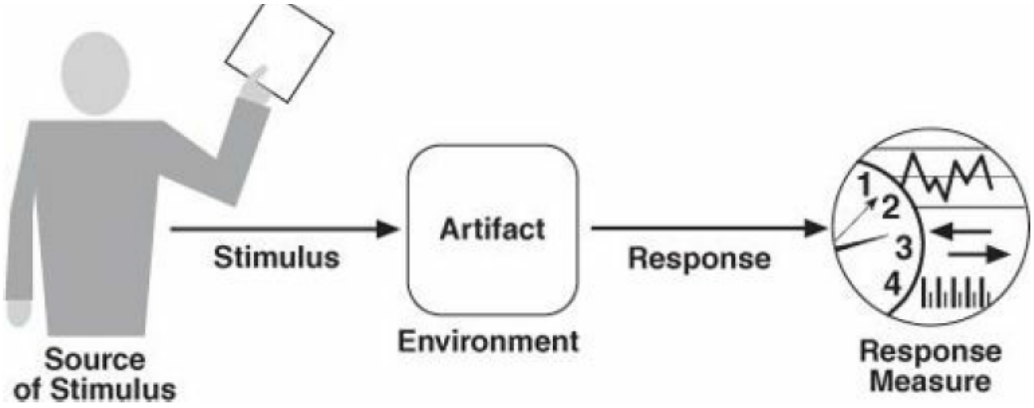
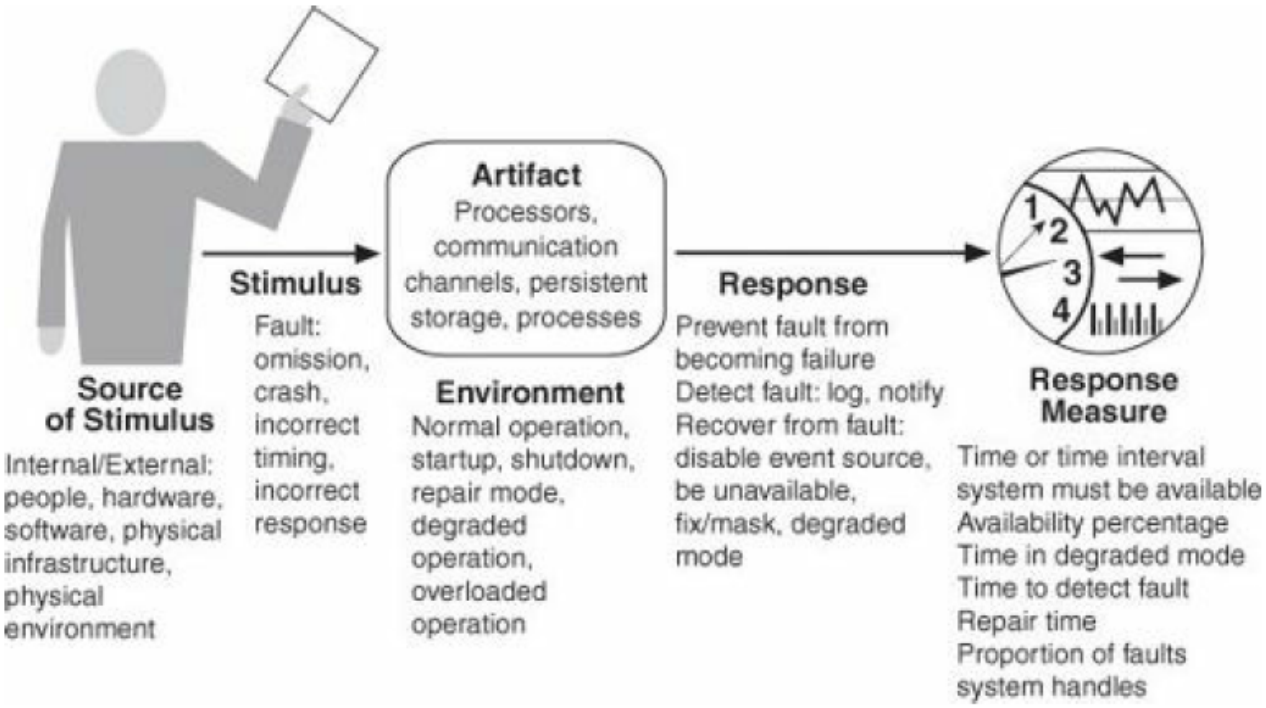


Figure 4.1. The parts of a quality attribute scenario



4.5 - Achieving Quality Attributes through Tactics

- **Tactics:** are techniques that an architect can use to achieve the required quality attribute. A tactic is a design decision that influences the achievement of a quality attribute response - tactics directly affect the system's response to some stimulus.

4.6 - Guiding Quality Design Decisions

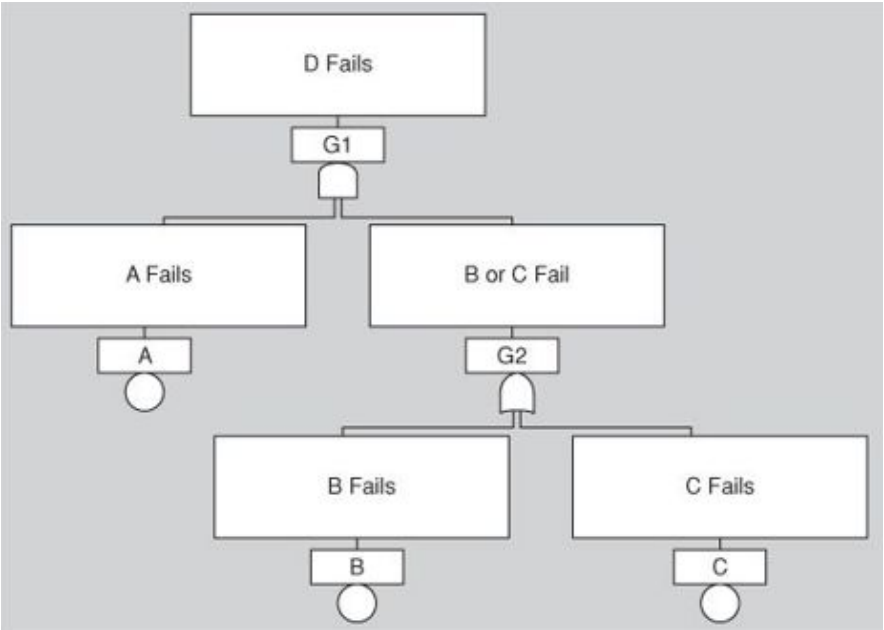
- **The seven categories of design decisions are:**
    - Allocation of responsability
    - Coordination model
    - Data model
    - Management of resources
    - Mapping among architectural elements
    - Binding time decisions
    - Choice of technology

Chapter 5 - Availability

- **Availability:** refers to the ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required value over a specified time interval."
- **Dependability:** is the ability to avoid failures that are more frequent and more severe than is acceptable.
- **Values for acceptable system downtime:** (%)
- **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

- **Planning for failure:** se for seg på forhånd hvilke failures som kan oppstå. Vi kommer ikke utenom failures
- **Hazard analysis:** is a technique that attempts to catalog the hazards that can occur during the operation of a system. It categorizes each hazard according to its severity:
    - Catastrophic, Hazardous, Major, Minor, No effect
- **Fault tree analysis:** is an analytical technique that specifies a state of the system that negatively impacts safety or reliability, and then analyzes the system's context and operation to find all the ways that the undesired state could occur.



(Figure: D fails if A fails and either B or C fails)

5.1 - Availability General Scenario

- **Source of stimulus:** We differentiate between internal and external origins of faults or failure because the desired system response may be different.
    - **Stimulus:** A fault of one of the following classes occurs:
    - **Omission:** A component fails to respond to an input.
    - **Crash:** The component repeatedly suffers omission faults.
    - **Timing**: A component responds but the response is early or late.
    - **Response:** A component responds with an incorrect value.
- **Artifact:** This specifies the resource that is required to be highly available, such as a processor, communication channel, process, or storage.
- **Environment:** The state of the system when the fault or failure occurs may also affect the desired system response. For example, if the system has already seen some faults and is operating in other than normal mode, it may be desirable to shut it down totally. However, if this is the first fault observed, some degradation of response time or function may be preferred.
- **Response:** There are a number of possible reactions to a system fault. First, the fault must be detected and isolated (correlated) before any other response is possible. (One exception to this is when the fault is prevented before it occurs.) After the fault is detected, the system must recover from it. Actions associated with these possibilities include logging the failure, notifying selected users or other systems, taking actions to limit the damage caused by the fault, switching to a degraded mode with either less capacity or less function, shutting down external systems, or becoming unavailable during repair.
- **Response measure:** The response measure can specify an availability percentage, or it can specify a time to detect the fault, time to repair the fault, times or time intervals during which the system must be available, or the duration for which the system must be available.

Chapter 6 - Interoperability

Chapter 7 - Modifiability

Chapter 8 - Performance

Chapter 9 - Security

Chapter 10 - Testability

Chapter 11 - Usability

Chapter 12 Other Quality Attributes

Chapter 13 - Architectural Tactics and Patterns

Chapter 14 - Quality Attribute Modeling and Analysis

Part three: Architecture in the life cycle

Part four: Architecture and business

- **Chapter 23:** Economic Analysis of Architectures
- **Chapter 25:** Architecture and Software Product Lines

Part five: The brave new world

- **Chapter 26:** Architecture in the Cloud
- **Chapter 27:** Architectures for the Edge