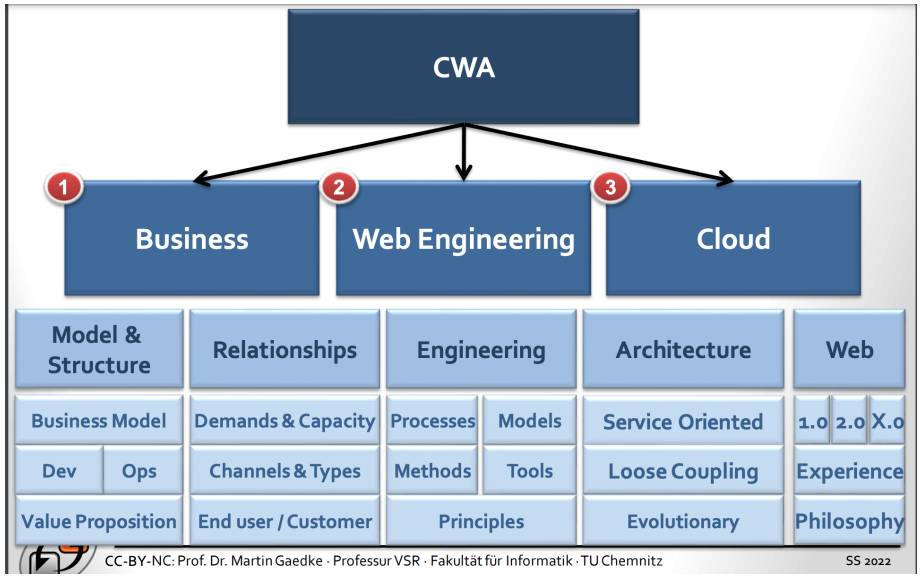
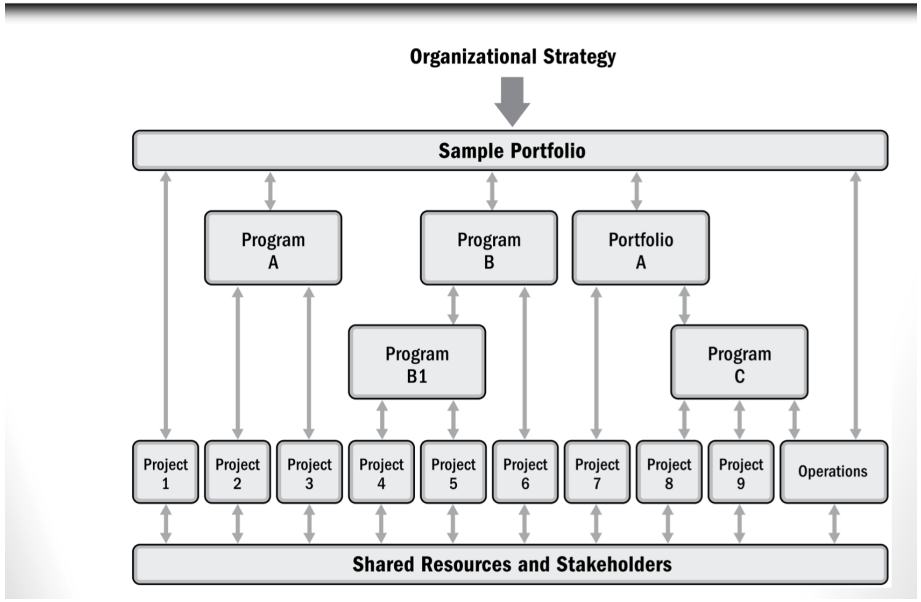


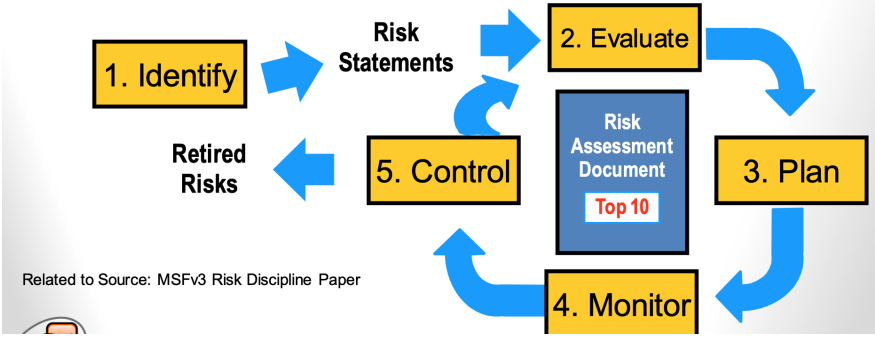
Project, Program, Portfolio



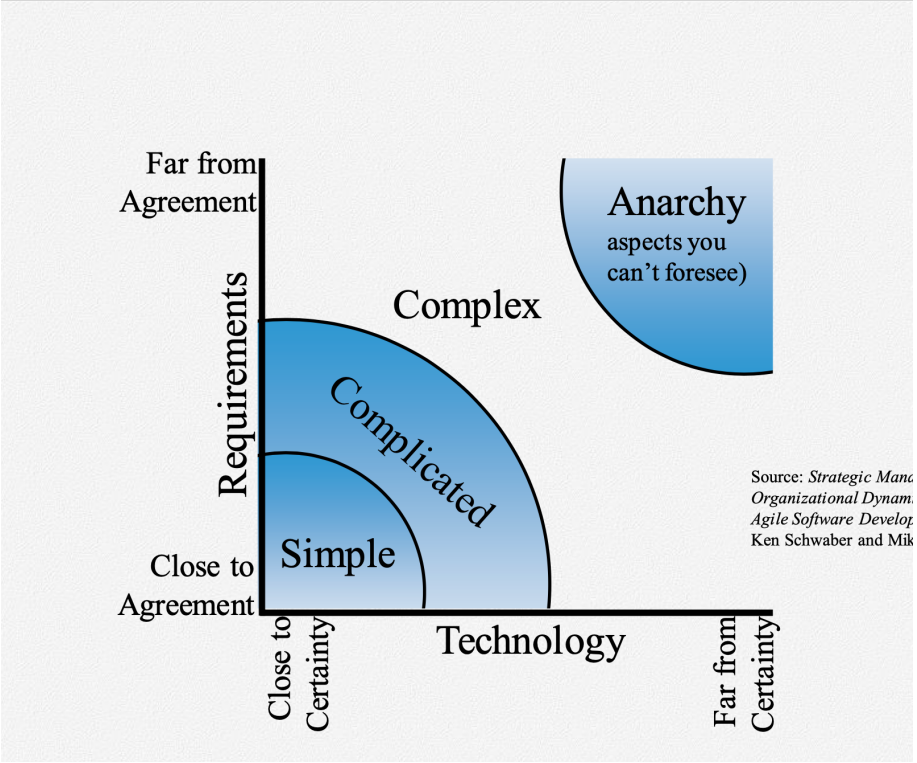
Project management	Web Engineering
Objectives and Stakeholder Analysis	Requirements Analysis
Risk Analysis	requires input/expertise from Web Engineering
Identification and organization of activities	Web Engineering Processes
Schedule/Milestones	Planning, Releases etc.
Progress Monitoring	Web Engineering Process visibility

Risk Management Process

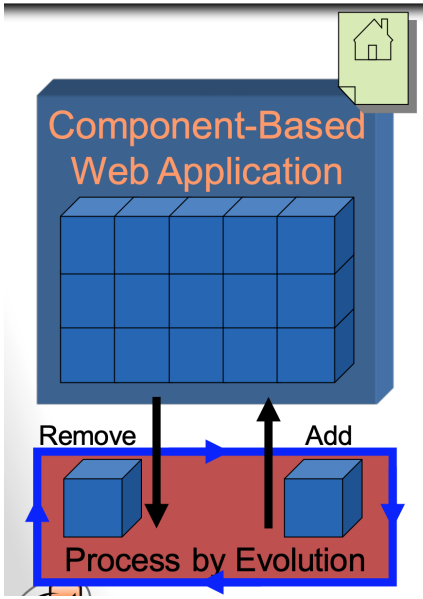
- **Risk** – Any event that could potentially have a negative impact on the project
  - Remember: Risk is not a problem as refers to the future!
  - Evaluate: Quality analysis (prioritize effect on project objective) & Quantity analysis (probability and estimating implications)



We focus on Complex Problems

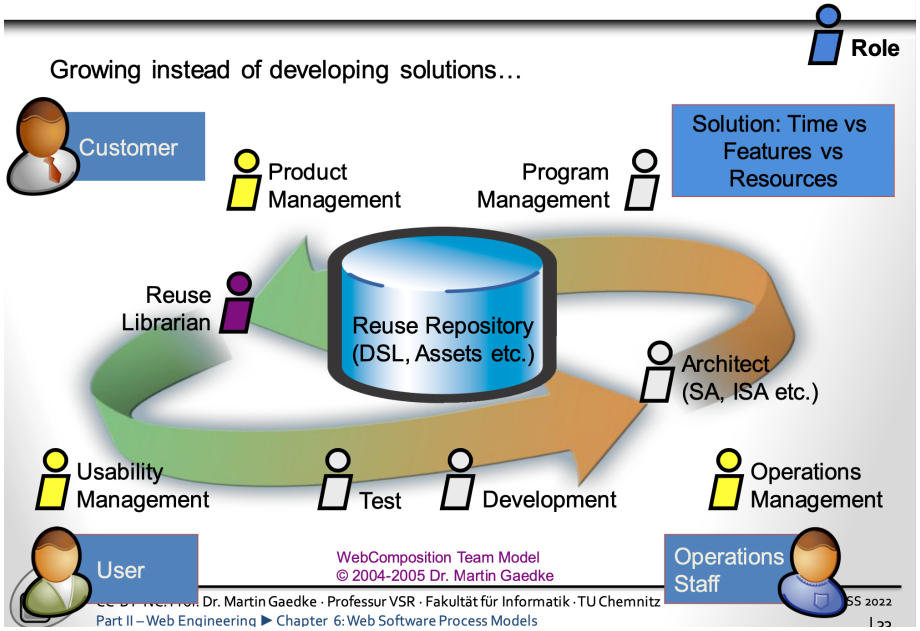


Reuse-Oriented Approaches

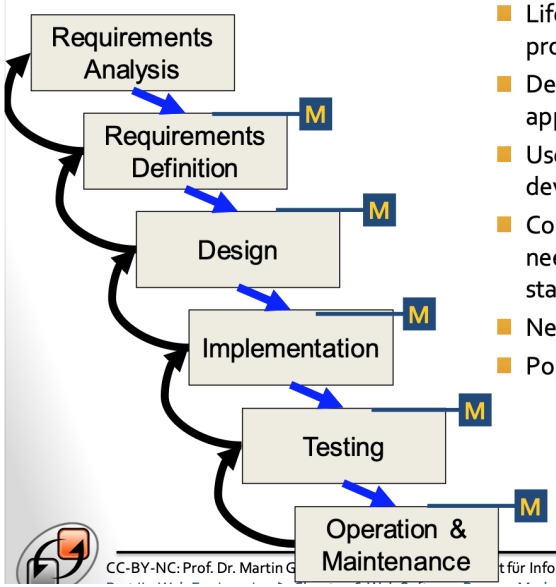


- Web Engineering in context of Reuse
- Product is assembly from **reusable components**
  - Idea: All needed Components exist
- These Approaches focus on being **agile** in the context of
  - **Producer Reuse**
  - **Consumer Reuse**

Evolution-oriented Team

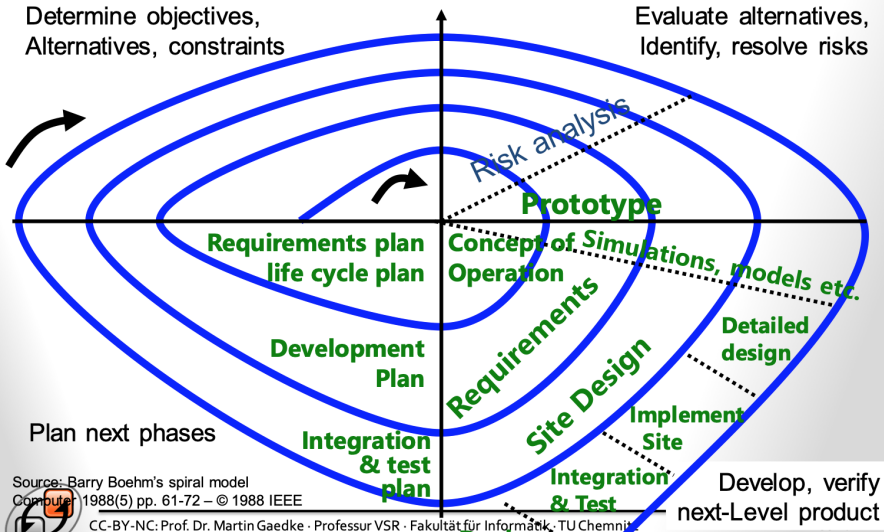


The Waterfall Model



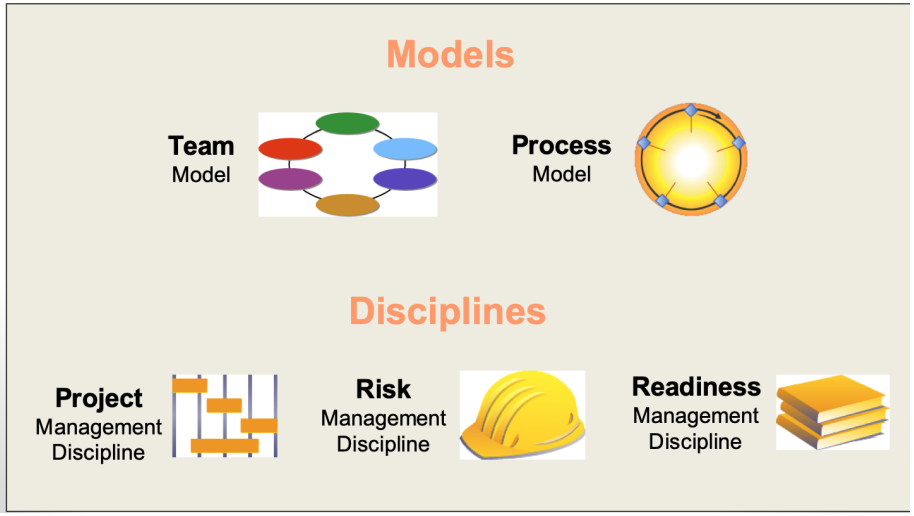
- Life Cycle Model – software process as number of stages
- Derived from other engineering approaches (late sixties)
- Used for practical Web Application development in several variants
- Completion of stage / milestone needed to proceed to following stage (M)
- Negative: Problems in early stages
- Popular for management

Spiral Model (Risk-Driven)

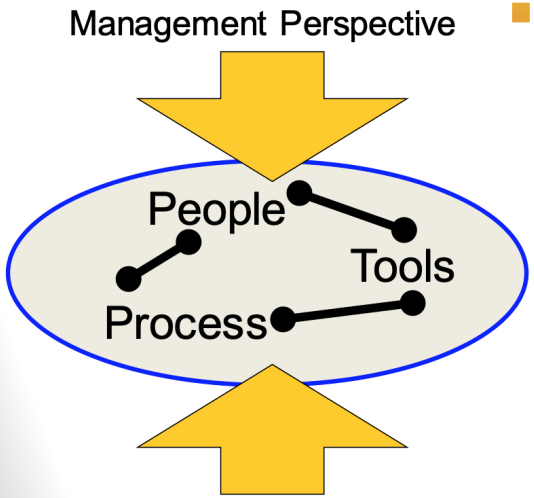


Source: Barry Boehm's spiral model
Computer 1988(5) pp. 61-72 – © 1988 IEEE

Key MSF Components (MSF v3)



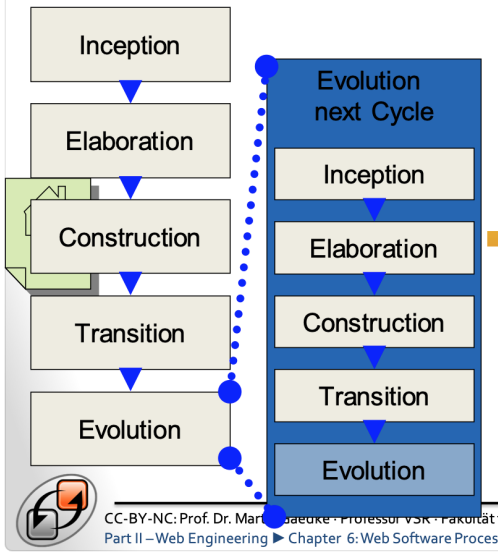
Rational Unified Process®



- RUP
  - ▶ Unified Software Development Process
  - ▶ Jacobson, Booch und Rumbaugh

Source: Philippe Kruch
A Rational Developme
Crosstalk, 9(7), July 19

Management Perspective



- Development Life *Cycle*
  - ▶ Inception:
    - vision
    - business case
    - scope of the project
  - ▶ Elaboration: Planning
  - ▶ Construction
  - ▶ Transition: Deliver, Training
- Evolution
  - ▶ Life of software after initial development cycle

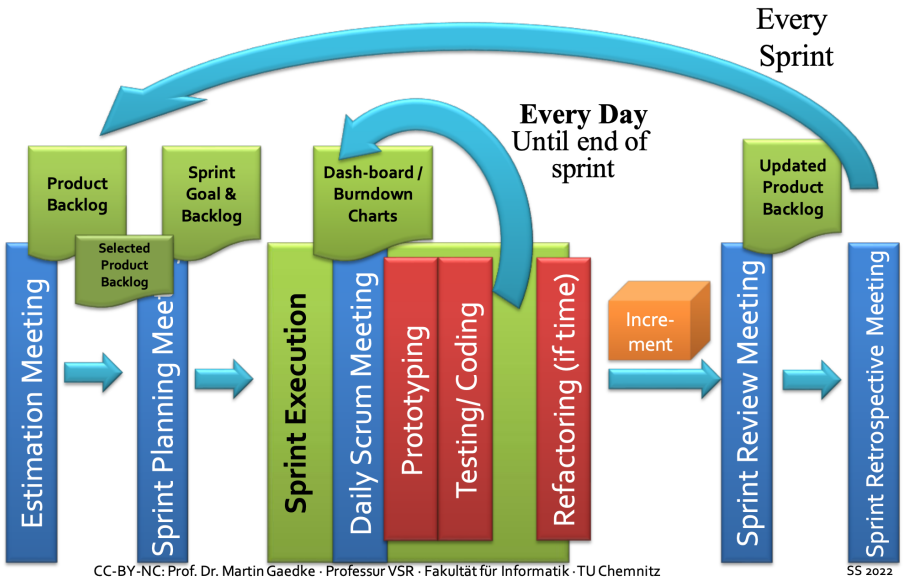
Source: Philippe Kruchten, *A Rational Development Process*, Crosstalk, 9(7), July 1996

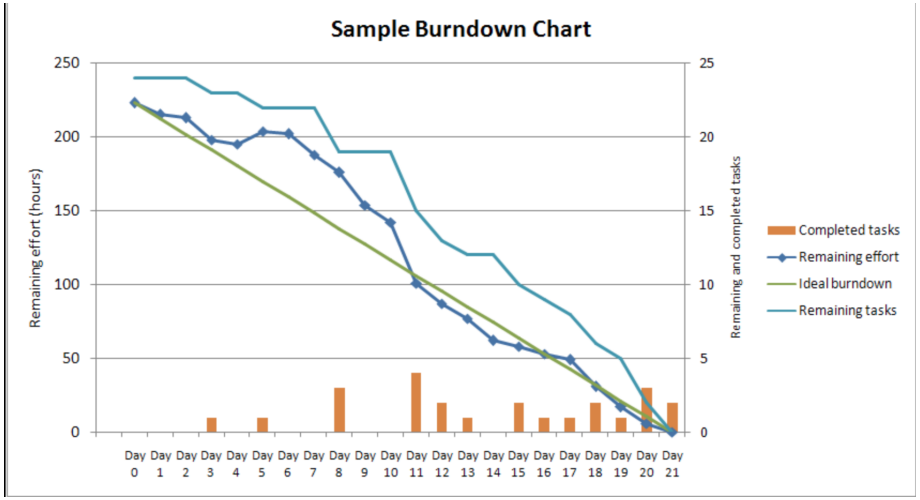
Conceptos	RUP	MSF	XP	SCRUM
Definición	Análisis diseño implementación y documentación de sistemas orientados a objetos	Combina el concepto de cascada basado en el alcance de los hitos, con el modelo en espiral basado en los entregables iterativos e incrementables.	Basado en la adaptación, mayor flexibilidad, dinámica y funcional.	Gestión y desarrollo de software basado en un proceso iterativo e incremental
Adaptar el proceso de desarrollo	X	X	X	X
Alto nivel de abstracción	X			X
Fomento del aprendizaje de experiencias	X	X		
Centrarse en la arquitectura	X			X
Interacción continua con el cliente	X		X	X
Código estándar			X	
Colaboración entre equipo	X	X	X	
Demostrar resultados Iterativamente e Incrementalmente	X	X	X	X
Diseño simple			X	
Modelar el software	X			
Enfoque continuo en la calidad	X	X	X	X
Enfoque en los riesgos	X	X	X	X
Permanecer ágil y esperar los cambios	X	X	X	X
Dirigido por casos de uso	X			
Para pequeños grupos de trabajo		X	X	X
Para grandes grupos de trabajo	X			X
Para proyectos complejos	X			X

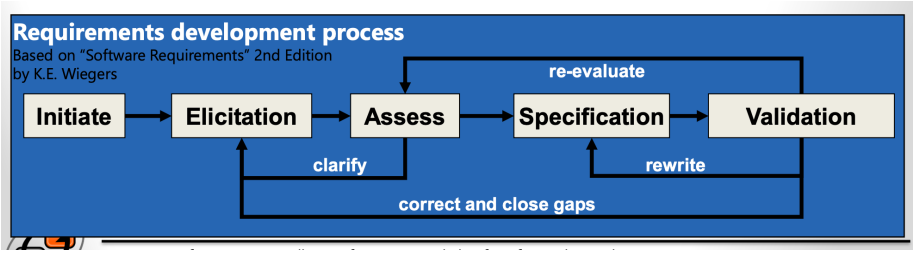
Agile Manifesto

- Principles & Values defined by Manifesto for Agile Software Development
  - ▶ Individuals & interactions > processes & tools
  - ▶ Working software > comprehensive documentation
  - ▶ Customer collaboration > contract negotiation
  - ▶ Responding to change > following a plan
- **Manifesto acknowledges the value of the right items, but focuses the value on the left more**
- For further information, cf.: http://agilemanifesto.org/

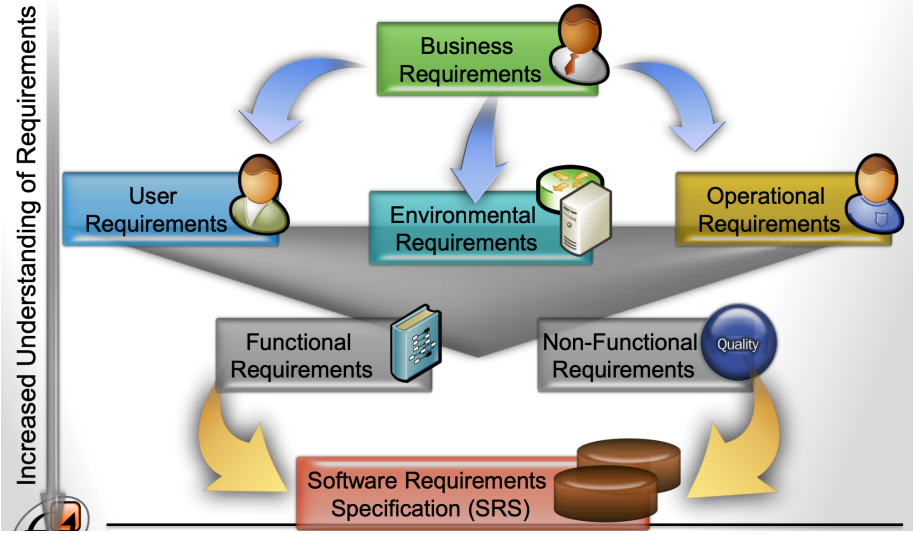
Scrum 'applied' in more detail...







The Requirements Big Picture



Levels of Requirements

- **Business** Requirements
  - ▶ High-level objectives of the organization or customer
- **User** Requirements
  - ▶ Tasks that users must be able to perform using the new product
- **Operational** Requirements
  - ▶ Tasks that operations staff must be able to perform using the new product
- **Environmental** Requirements
  - ▶ Aspects of the technology available and to be applied as well as the project's ecosystem

User/System Requirements

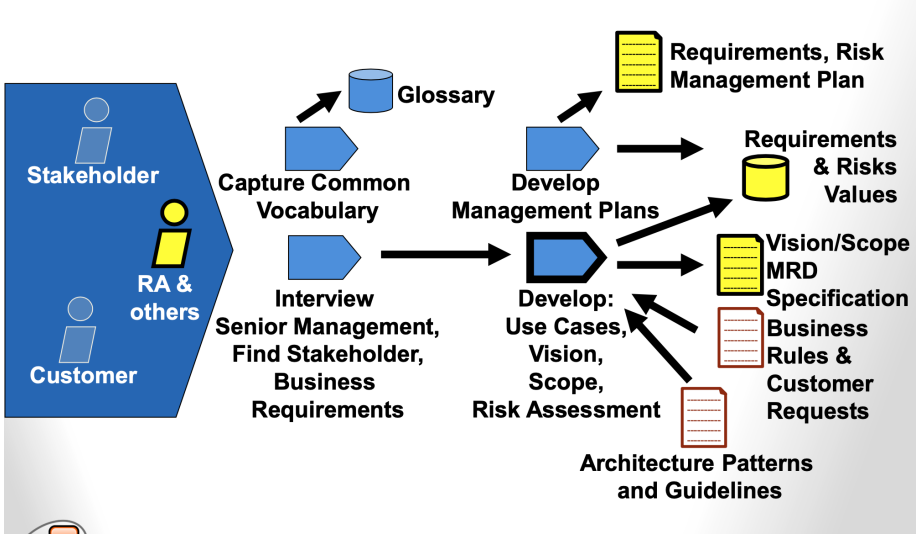
User requirements

- derived from user needs and capabilities
  - ▶ specify the extent to which user needs and capabilities are to be met when using the system
  - ▶ comprise *functional and non-functional* user requirements
  - ▶ high-level abstractions for client/contractor managers and system architects

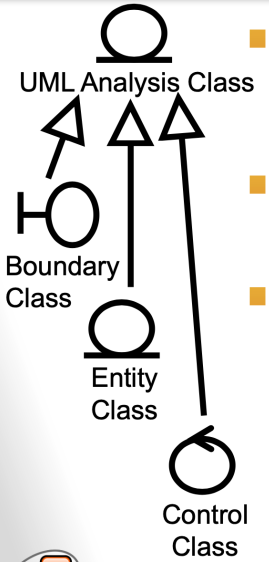
System requirements

- derived from user requirements and business requirements (context, budget etc.)
- detailed descriptions of a system's functions, services and operational contraints for system architects, developers
- comprise functional and non-functional system requirements
- basis for design and implementation
- typically captured in structured form (e.g. tabular with defined criteria)

Initiate Phase – Summary



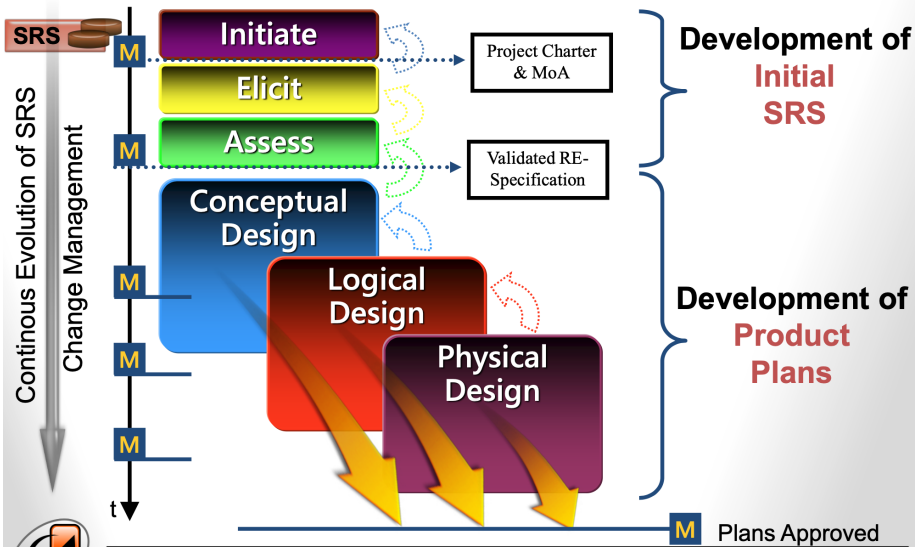
Conceptual Model



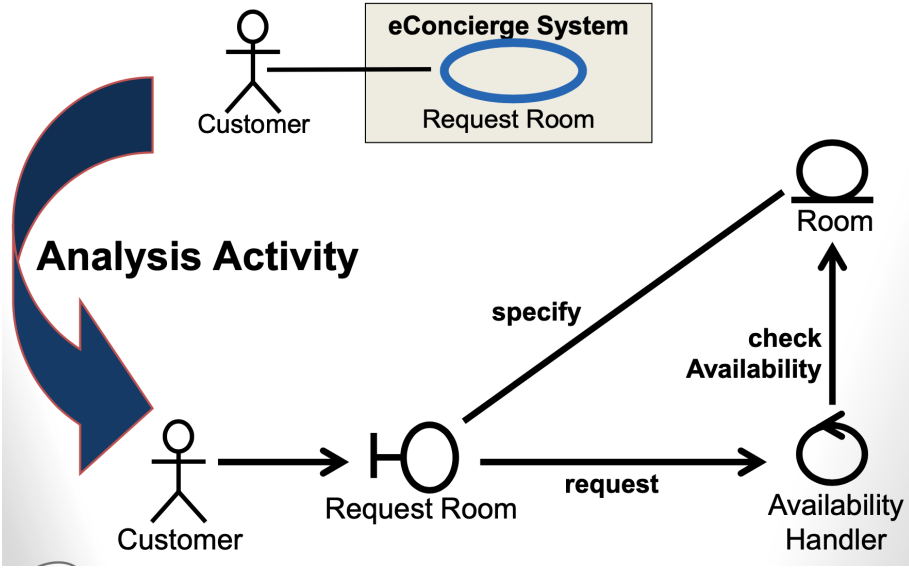
- **«boundary»**
  Models Interaction between Product and Actors (e.g. User Interface or Communication Interface)
- **«entity»**
  Models information and associated behavior (e.g. real-life object, event)
- **«control»**
  Models coordination, sequencing, transactions, and control of other objects (e.g. Business Logic)
  - Do not encapsulate interactions with actors!

SRS and Planning

Traceability

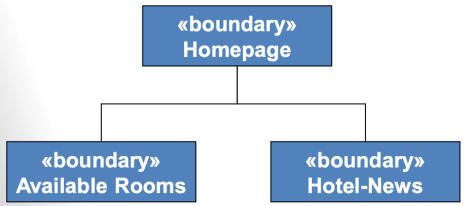


Transforming Example



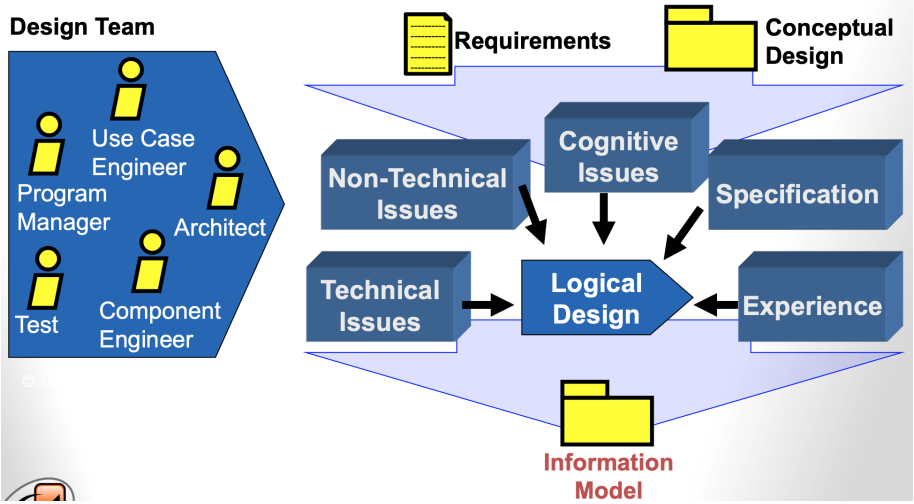
Model Navigation

- Analyze Relations
  - Entity to entity relations based on RNA results
  - Set-based relations
- Possible Navigation Patterns to apply



- Types of Relations
  - Direct access to entity
  - Unidirectional / bidirectional
  - Access by type, e.g. to retrieve additional information about an entity
  - Transactional
  - Context-oriented relations
- A lot of research done here, e.g. OOHDM, UWE, OOH, WebML etc.

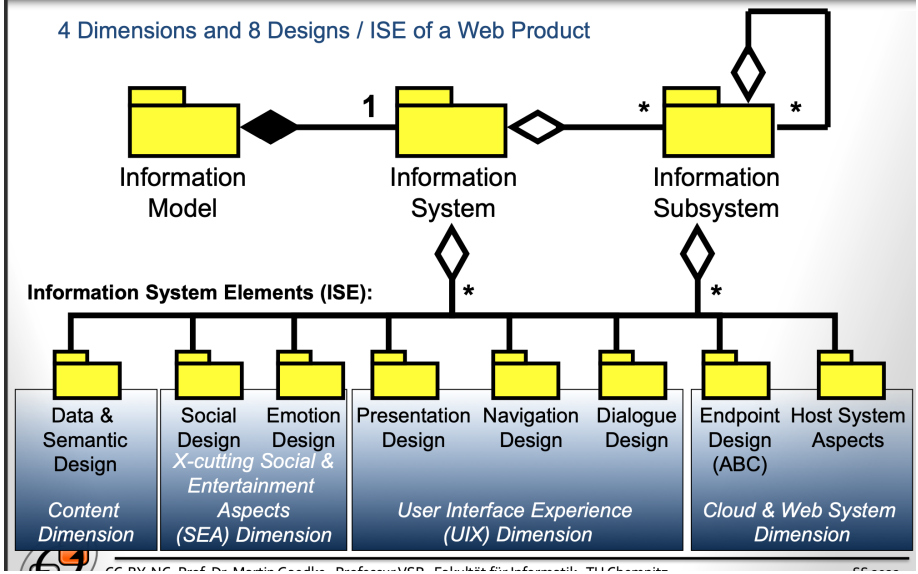
Design Team and Artifacts



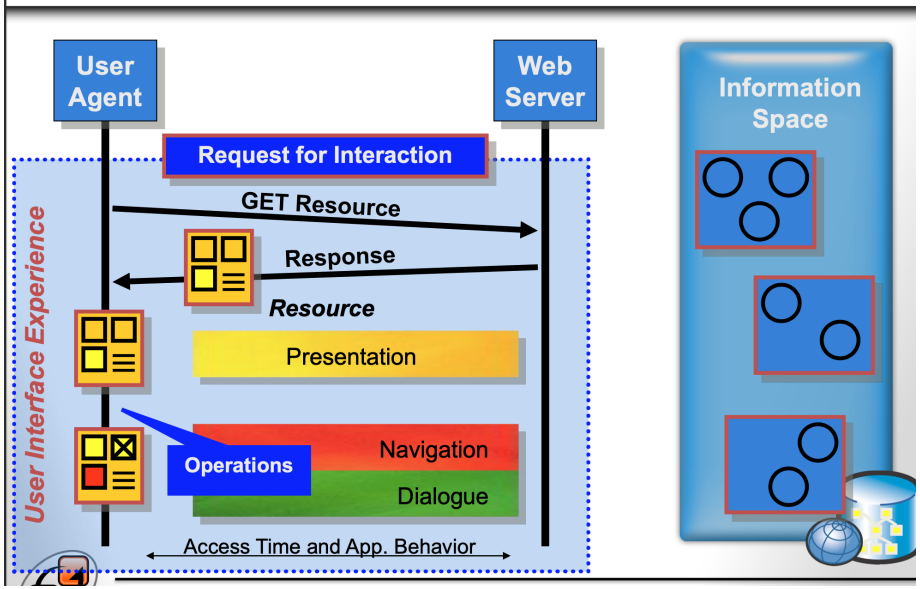
Summary

- Business Process
  - Spawns conceptual design of the distributed system
- Distributed System is represented by different views (layers)
  - Common approach: Process Layer, Service Layer, and Network Layer
  - Allows for layer of interest, e.g. Federation Layer for inter-organizational processes, hosting layer
  - Apply dedicated approaches (logical design and physical design) for each of these layers

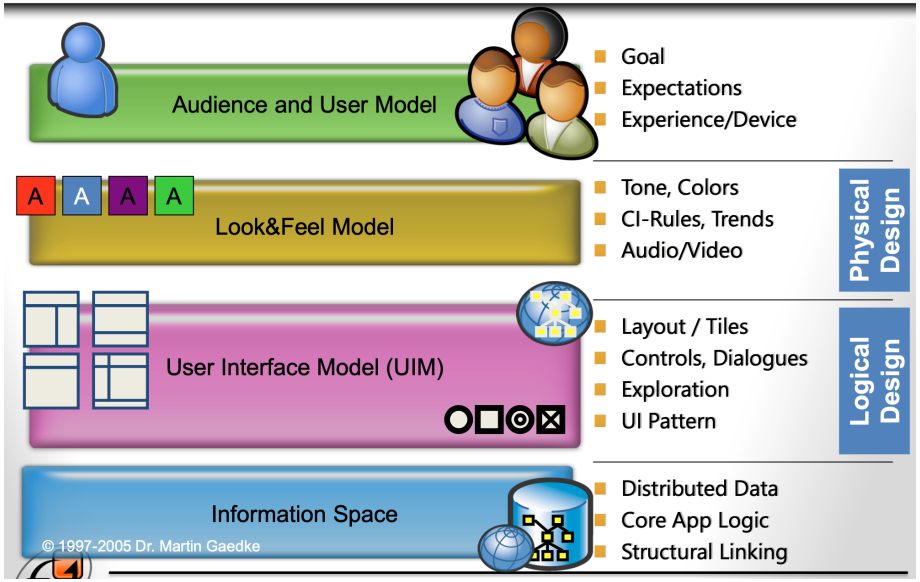
Information Model



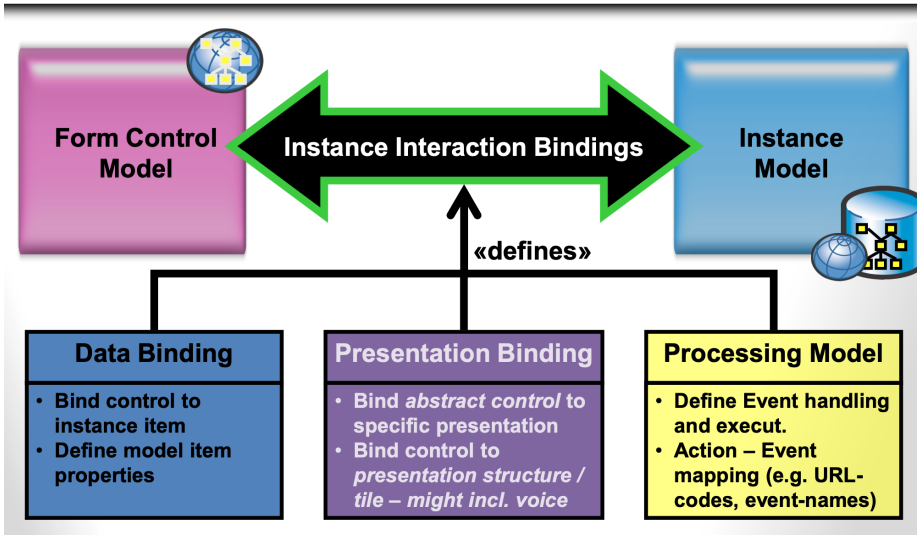
Web Interaction Model



Logical & Physical Aspects



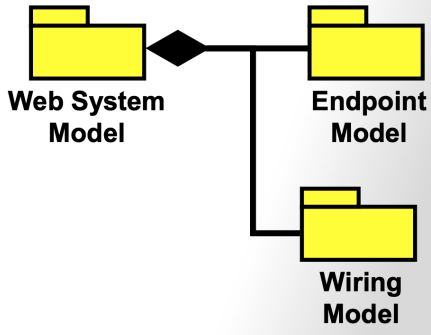
Shift to Physical Design



Web System

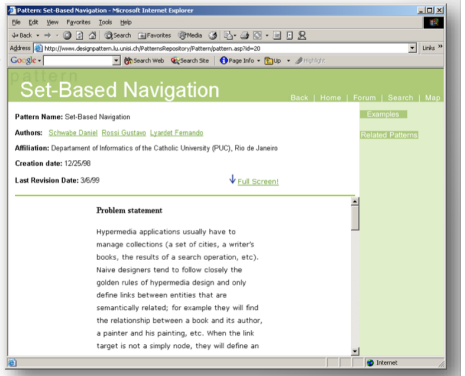
- **Web System Model** – A model that defines the overall product logic as a set of **Endpoints** and their **Wiring** (connections). It abstracts the implementation, e.g. SaaS / Public Cloud, Web on premise etc.

  - **Endpoint Model**
    - Endpoint interface and message definitions
    - Possible Service Level Access Requirements
  - **Wiring Model**
    - Defines wiring between endpoints
    - Depends on communication aspects



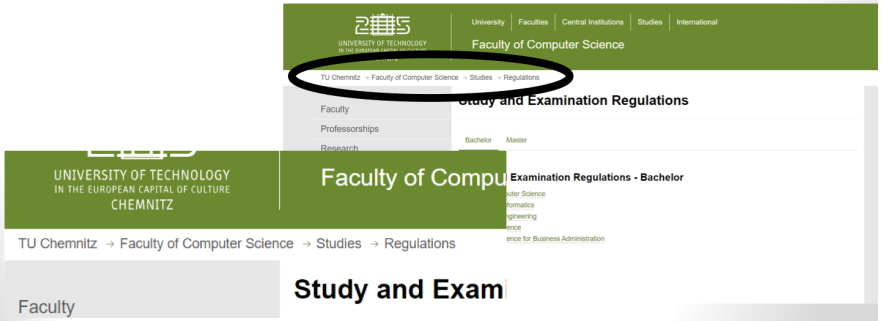
Pattern: Set-Based Navigation

- Name:
  - Set-Based Navigation
- Problem:
  - Naive Designers tend to follow closely the golden Rules of Hypermedia Design and only define Links between Entities that are semantically related…
  - The user will have to move from the index to a target



Pattern: Active Reference

- **Solution**:
  - Maintain an active and perceivable navigational object acting as an index to other nodes



Pattern: Active Reference

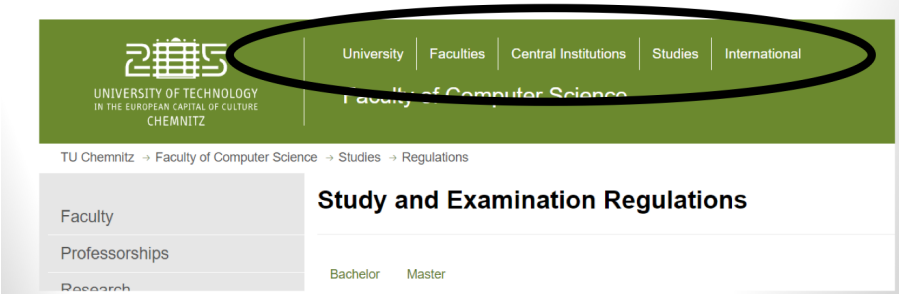
- **Name**:
  - Active Reference
  - Also known as: Breadcrumb Navigation
- **Problem**:
  - We need a way to help the user understand where she/he is and where to go next. Indexes or other access structures provide only partial solutions

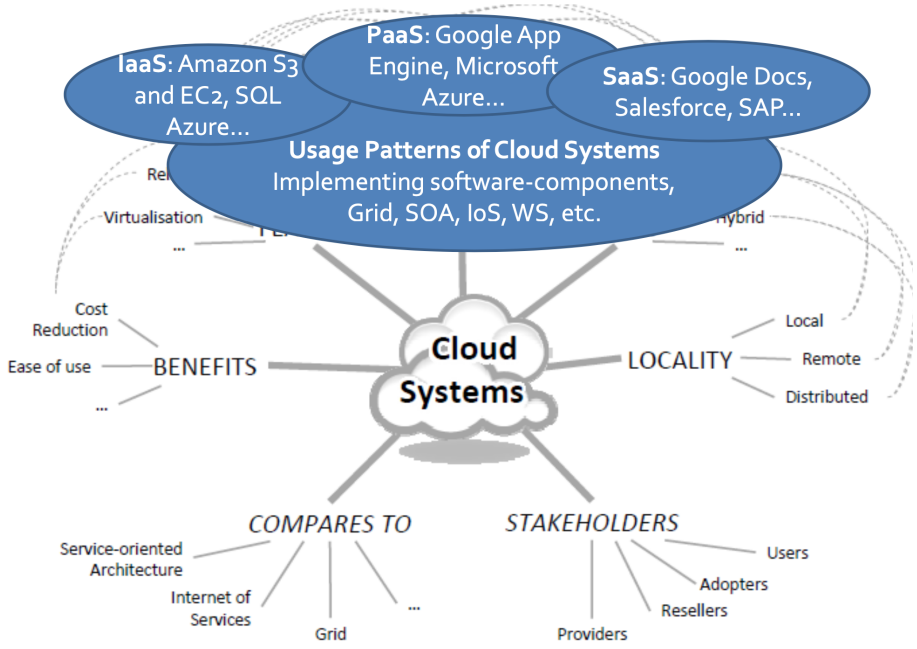
Pattern: Landmark

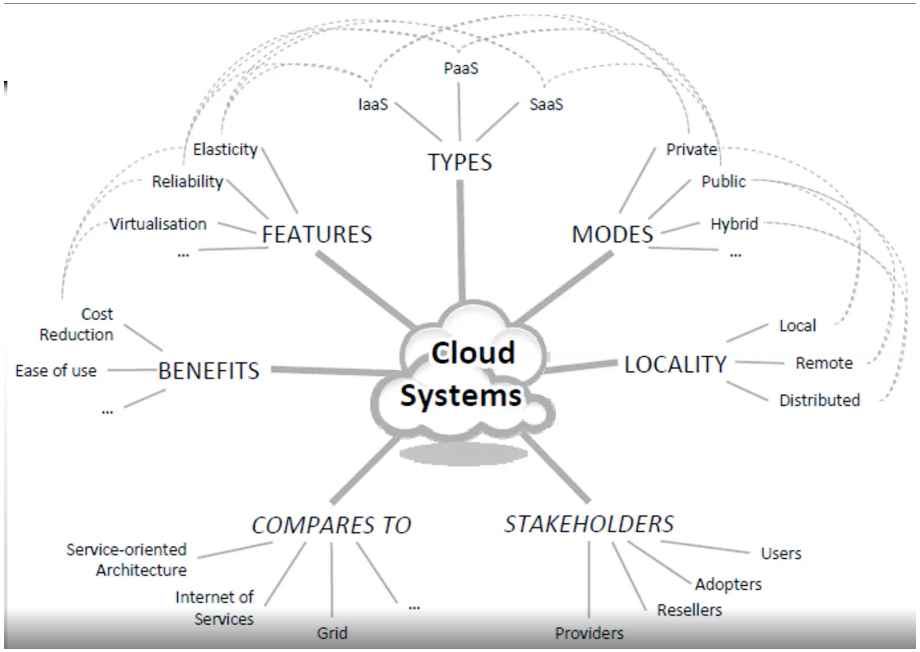
- **Name**:
  - Landmark
- **Problem**:
  - How to give easy access to different unrelated sub-systems? Web Applications usually contain many interesting entry-points; links to those points do not reflect conceptual relationships, and those links may yield a spaghetti-like structure

Pattern: Landmark

- **Solution**:
  - ▶ Define a set of Landmarks and make them accessible from every node in the application







Prototyping Model II

- Prototype – *only* responsible for defining system requirement
- Suitable if system requirements can not described completely in the beginning
- Applicable for reuse approaches
- Open Process Model – use of any process model
- Further Issues
  - ▶ Good for motivation of team
  - ▶ Increases trust of customer

What Is Availability?

- How much downtime can my organization afford without loosing productivity, profits, sales, etc.?
- The solution to High Availability is a combination of people, process, AND technology
  - Beware of 99.99% myth - The nines model does not take timing into account

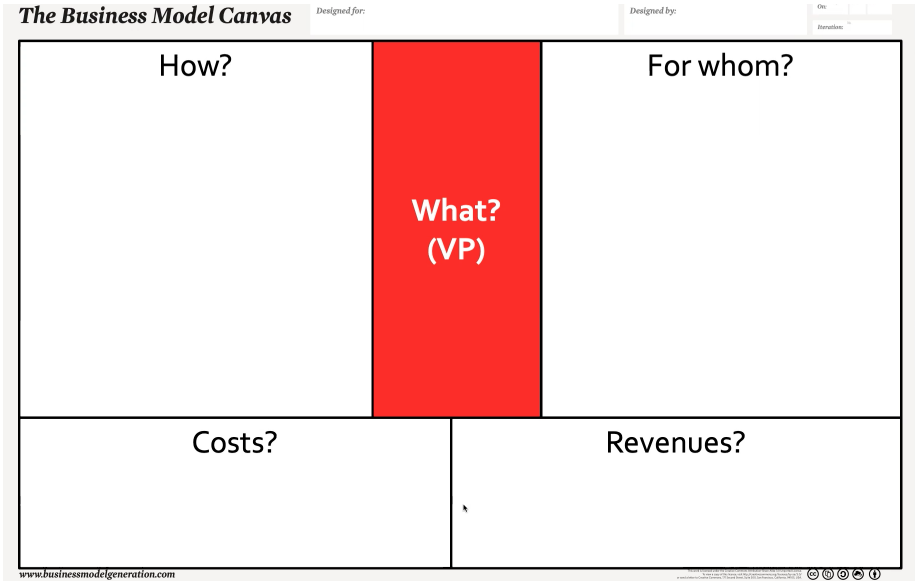
Availability:

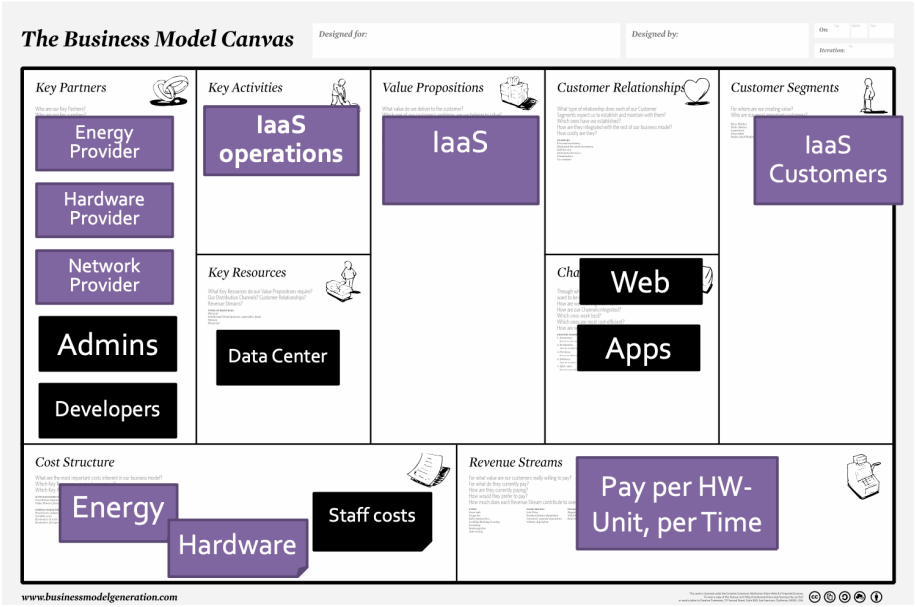
A = (MTBF / (MTBF + MTTR)) * 100

- Mean Time Between Failure (**MTBF**) – average time a system is actually operational **[hours / failure count]**
- Mean Time To Recovery (**MTTR**) – average time needed to repair and restore service after failure **[Repair hours / failure count]**

- Simple Example:
  - 24/7 Web Site with two failure a week and each requires 1 hour
- On a year's time:

$$\frac{(52*7*24 / 52*2)}{(52*7*24 / 2) + 1/2} * 100 = 99.41\%$$





Cloud Platform Characteristics

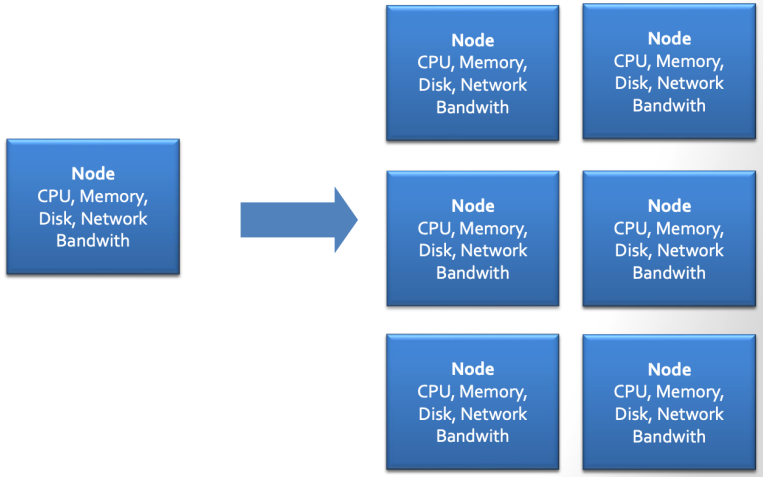
- Enabled by „infinite" resources, limited by maximum capacity of individual VMs, **cloud scaling is horizontal**
- Enabled by short-term resource rental model, **cloud scaling releases resources as easily as they are added**
- Enabled by a metered pay-for-use model, **cloud applications only pay for currently allocated resources**
- Enabled by self-service, on-demand, programmatic provisioning/releasing of resources, **cloud scaling is automatable**
- Enabled & Constrained by multitenant services, cloud applications are optimized for cost rather than reliability: **failure is routine, but downtime is rare**
- Enabled by a **rich ecosystem of managed platform services** such as for VMs, data storage, messaging and networking, cloud application development is simplified

Cloud-Native Application

- **Leverages cloud-platform services** for reliable, scalable infrastructure („*Let the platform do the hard stuff*.“)
- Uses **non-blocking asynchronous communication** in a **loosely coupled architecture**
- **Scales horizontally**, adding resources as demand increases and releasing resources as demand decreases
- Cost-optimizes to run efficiently, not wasting resources
- **Scales automatically** using proactive and reactive actions
- Handles scaling events & transient failures without user experience degradation
- Handles node & transient failures without downtime

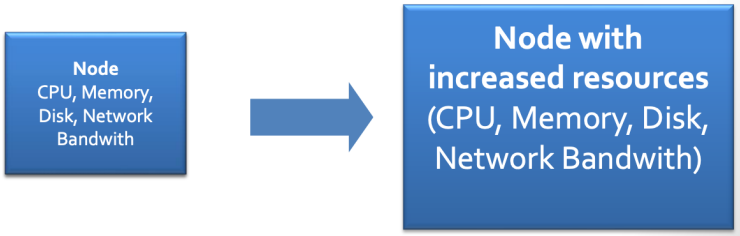
Vertical vs. Horizontal

- Scale out



Vertical vs. Horizontal

- Scale up



Horizontally Scaling Compute Pattern

- Cloud Scaling is **reversible**
- **Elasticity**
  - ▶ Add additional compute nodes when required
  - ▶ Release compute nodes when no longer required
- **Cost-oriented**
  - ▶ Down-scaling saves money
- **Virtualization** as enabling technology
  - ▶ Horizontal scaling is achieved adding/removing Virtual Machines with identical configuration (image) as compute nodes

Session state without stateful nodes

- For efficient horizontal scaling, session state cannot be stored locally on the nodes
- Session state has to be stored externally
  - ▶ use Cookies & LocalStorage if size allows
  - ▶ else use NoSQL data stores, cloud storage, distribute caches, store session identifiers in Cookies
- Allows individual nodes to stay autonomous
- Scalability issue is shifted to the storage mechanism

Queue-Centric Workflow Pattern

- UI (Web) Tier adds messages to the queue
- Requires *reliable queue* (usually as service provided by cloud platform)
- Receiver (at-least-once processing)
  1. Get next available message from the queue and mark as hidden
  2. Process message
  3. Delete message from queue
- Invisibility Window
  - ▶ Message is hidden for limited period of time
  - ▶ Automatic reappearance of messages if processing time exceeds invisibility window and no refresh → at-least-once

Queue-Centric Workflow Pattern

- Dequeue count
  - ▶ Increased for each processing attempt of a message
- Additional step indicators for multi-step processes, if failure continue from last completed step
- Poison messages
  - ▶ Erroneous messages, cannot be processed successfully
  - ▶ Detect: dequeue count > N, depending on processing time
  - ▶ Handle: remove from queue, important messages: add to dead letter queue for later inspection
- Beware of possible money leak due to queue service
  - ▶ Queue service calls (e.g. dequeue) cause costs
  - ▶ Avoid fast polling

When to apply Auto-Scaling

- Cost-efficient scaling of computational resources
- Continous monitoring of fluctuating resources is needed to maximize cost savings and avoid delays
- Frequent scaling requirements involve cloud resources such as compute nodes, data storage, queues, or other elastic components

Auto-Scaling Pattern

- Schedule for known **events** (e.g. Champoins league, business hours)
- Create **rules** to react to environmental **signals**
  - Fixed dates/times
  - Average response time & Queue length
  - Memory & CPU utilization
  - Node failures
- Rules
  - Increase/Decrease nodes (cf. scale units)
  - *N+1 rule* deploy N+1 nodes if N nodes are required to buffer sudden spikes & cope with node failure
- Use upper/lower bounds to constrain auto-scaling rules

Eventual Consistency

- CAP Theorem: A distributed system cannot satisfy all of the following three guarantees
  - Consistency (all queries yield the same data at the same time)
  - Availability (all requests served either successfully or with failure notification)
  - Partition tolerance (correct operation, even if some nodes fail)
- Eventual Consistency
  - trades Consistency for Availability and Partition tolerance

When to apply MapReduce

- Application processes large volumes of data (structured, semi-structured, unstructured) stored in the cloud
- Data analysis requirements change frequently or are ad hoc
- Application requires reports beyond capabilities of traditional database reporting tools because input data is too large or no in compatible structure

MapReduce

- Simple programming model for processing highly **parallelizable** data sets
- Batch processing approach
- Two phases: Map & Reduce
  - **Map** function applied to each input element (key-value pairs) produces intermediate results (key-value pairs)
  - Wait for all Map functions to complete, apply **Reduce** function to each intermediate result producing final results
- All Map functions are independent → can be run in parallel
- All Reduce functions are independent → can be run in parallel
- MapReduce as Service offered by Cloud Providers
  - Hadoop (as a Service)

MapReduce Example Word Count

```
def map(documentName, documentContent):
    for line in documentContent:
        words = line.split(" ")
        for word in words:
            EmitIntermediate(word, 1)

def reduce(word, counts):
    wordCount = 0
    for count in counts:
        wordCount += count
    Emit(word, wordCount)
```

MapReduce Example Word Count