Architectures

1 Intro

- Architectural Refinement
  - Domain > uses > Technical > uses > Platform
  - Architecture > structures > Components
- architecture specifies "load carrying" components, not details
- makes complexity manageable

2 Requirements

- should be
  - correct (users, stakeholders)
  - implementable
  - unambiguously defined
  - testable
- levels
  - Oranization Level
  - System Level
  - Component Level
- Functional
  - Use Cases
- Non-Functional (Design Time, Runtime, Org Env)

3 Principles

- (Loose) Coupling among
  - dependencies among components
  - keep complexity of structures low
  - increase changeability
- (High) Cohesion within
  - dependencies within a component
  - component can be understood or changedwithout looking at other components
- Design for Change
  - plan for foreseeable changes
  - consider requirements likely to arise
  - Risks
    - more development time
    - higher costs, memory, performance
- Separation of Concerns
  - Modularization vs "God Class"
  - Requirements
  - Complex Architecture Model into Views
  - Organizational Responsibilities
  - Aspect-Orientation...
- Information Hiding
  - hide implementation details behind an interface
  - present only really necessary parts > Facade Pattern
- More

    - Abstraction - Explicit Interfaces, Segregation of Interface and Implementation
    - Traceability, Self Documentation, Incremental Evolution

4 Styles

- describe fundamental structures of a software system and its properties
- Component types +Topology + Connectors + Semantic constraints
- Examples
    - Dataflow Systems
        - Batch Sequential, Pipes and Filters
    - Call-and-Return Systems
        - Main Prog, OO, Hierarchical Layers
    - Independent Components
        - Communication, Event Systems, Implicit Invocation
    - Virtual Machines
        - Interpreters, Rule-based Systems
    - Data-Centered-Systems (Repositories)
        - DBs, Hypertext Systems, Blackboards

5 Patterns

- Gang-of-Four Book Design Patterns
- reusable design expterise, time-proved solutions to commonly occuring problems
- also used for documentation and communication
- resolves/balances a system of forces (Memory, Costs, Performance, ...)
- describes consequences / positive negative
- Layers Pattern
    - problem
        - high- & low-level components decoupling
    - solution
        - layer has same level of abstraction
        - interface between layers
    - Indirection Layer Pattern: wrap access
- Batch Sequential Pattern
    - problem
        - complex task, can be sub-divided
    - solution
        - sub-steps separated
        - call next sequential step
    - processes batch of data per step
- Pipes and Filters Pattern
    - problem
        - flexibly compose individual sub-tasks
    - solution
        - Filters
            - independent components
        - Pipes
            - connectors, data buffers
- Shared Repository Pattern
    - problem

        - share data, sequential architectures
        - common way: invocation parameters
        - inefficient for large data sets
        - information varies from invocation to invocation
        - long-term persistence
    - solution
        - central data store
        - scaleable, consistent, transactions...
    - Active Repository
        - registry of clients
        - notification mechanisms
    - Blackboard
        - heuristic computation by sub-tasks
        - step-wise improvement
        - control unit

6 Patterns by Views

- Data Flow and Tranformation View
    - Pipes and Filters
    - Batch Sequential
- Layered Decomposition View
    - Layers
    - Indirection Layer
- Data Repository View
    - Shared Respository
    - Active Respository
    - Blackboard
- Adaptation Infrastructure View
    - Microkernel
        - system family, different versions
        - services - internal / external servers
    - Reflection
        - unanticipated changes
        - meta level, meta-objects
    - Plugin
        - extend, runtime environments without re-compile
        - central (re-)config at runtime
    - Interceptor
        - extend reusable services
        - register for dispatchers > interceptor events
- Language Infrastructure View
    - Interpreter
        - parse/interpret language syntax / grammar at runtime
        - scripts, portable to interpreter platform implementations
        - simple eg: class per grammar rule
    - Virtual Machine
        - portability, code optimization, no runtime interpretation req
        - intermediary byte-code > VM implementation layer
    - Rule-Based-System
        - logical problems hard to express, nested if-statements
        - facts (data), rules (knowledge) + engine (application)

- Interaction Decoupling View
  - MVC
    - model, encapsulates data
    - views, display data to the user
    - controller, receives user input, transmits to the model
  - Presentation-Abstraction-Control
  - C2
  - Explicit Invocation
    - couple client-supplier
      - performance
      - direct communication
      - block for result
      - fixed topology
    - direct invocation by service name and parameters
    - synchronous, asynchronous
      - fire and forget (no response)
      - sync with server (ack, no result)
      - poll object (result)
      - result callback (result notification)
  - Implicit Invocation
    - de-couple client-supplier
      - unknown supplier
      - results not needed yet
      - dynamic client add/removal
    - indirect invocation through
      - publish-subscribe
      - message queuing
      - broadcast
- Component Interaction View
  - Client-Server
  - n-Tier Architectures
  - Publish-Subscribe
  - Peer-toPeer
- Distributed Communication View
  - Broker
    - hides communication complexity
    - requestor, invoker, marshaller, request handler
  - RPC
  - Message Queuing

7 Modeling

- Meta-models describe models > Modeling levels
- Modeling languages
  - meta-model = abstract syntax
  - notation = concrete syntax
  - informal description = semantics
- ADL
  - Architecture Description Languages
  - readable, formal respresentation of architecture
  - Components
    - functional + non-functional aspects via interfaces

    - Connectors
        - component communication
    - Configuration
        - of components and connectors
    - eg: Acme, AADL, C2, Darwin, Wright
- UML
    - MOF Meta Object Facility
        - meta-meta model
    - OMG levels
        - M3 MOF - EBNF-Formalism
        - M2 UML - Java Grammar
        - M1 UML Model A - Java Program P
        - M0 instance A - Execution of Program P
    - instrastructure + meta-model superstructure
    - hard extension - meta-model, new UML language
    - soft extension - profile (steoreotypes, tags, constraints)

8 MDSD

- Model-Driven Software Development
- models as central artifacts
- domain-related for efficiency
- DSL
    - Domain Specific Languages
    - tailor-made language for a specific problem domain
    - specialized, expressive and easy to use
    - embedded
        - extends General Purpose Language (GPL)
        - often interpreted instead of generation
    - external
        - different format than target language
        - any syntax, not bound to host language or platform
        - transformations map to target platform
- Code Generation
    - eg using templates
    - alternative: interpreter
- Options
    - informal box-and-line diagrams
        - often not precise enough
    - use existing ADL
        - often no ADL fits exactly
    - use UML and extend UML meta-classes
        - tedious work
    - custom architecture meta-model
        - glossary of architecture concepts
        - clarifying

9 Views

- make complex systems understandable
- different
    - stakeholders

    - tasks
    - abstraction levels
- 4+1 View Model
    - Use Case view
        - centric
        - validates the others
    - Logical view
        - describes system in terms of abstractions
        - eg: class, sequence, collaboration diagrams
    - Development view
        - structure of modules, files, packages
        - eg: package diagram
    - Process view
        - process communication
        - eg: activity diagrams, process modeling languages
    - Physical view
        - installation, exection environment
        - eg: deployment diagrams

10 Decisions

- strategic (long-term and significant), not operational
- form architecture
- steps
    - prepare > make > communication > realize > evaluate
- problems
    - decisions get lost in models
- Decision Modeling
    - template or meta-model
    - capture key design issues, rationale
    - conscious design decisions
    - consider impact on NFRs and quality factors

11 Organisational

- organisation culture defines values and norms
- sub-organisations, internationalization, outsourcing
- Organizational Patterns
- Agile Methods
- Individuals
    - influence collaboration
    - require Social competences
- group
    - composition, heterogeneous
    - experienced, cooperative moderator
    - skills appointment
    - one or two creative team members
- TOGAF
    - Generic Skills
    - Business Skills and Methods
    - Enterprise Architecture Skills
    - Program or Project Management Skills

    - IT General Knowledge Skills
    - Technical IT Skills
    - Legal Environment

12 Prüfungsfragen

- Architecture decision modelling
- 4+1 View Modell
- Model Driven Software Development
    - Domain
    - Domain Driven Design
    - Domain Specific Languages
- MOF und UML
    - 4-Schichtige Modellierungsarchitektur des OMG
- Arten von Anforderungen bei Architekturen
- Architecture Modelling Options
- DSL
    - embedded vs external
- Patterns- problem- solution- comparison
    - Layer
    - Batch Sequential
    - Pipel and Filters