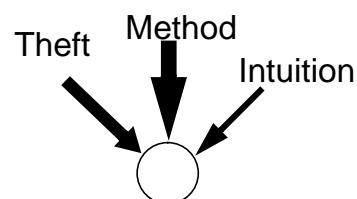
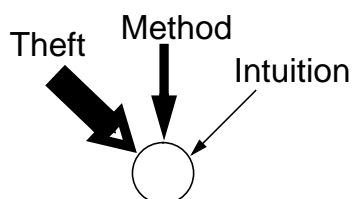


Review — Scope of Architectures

- Every system has an architecture
- Details of the architecture are a reflection of
 - system requirements
 - trade-offs that made to satisfy the requirements
- Possible decision factors
 - performance
 - compatibility with legacy software
 - planning for reuse
 - distribution profile
 - safety, security, fault tolerance
 - evolvability
- Critical question:
 - *HOW DOES ONE ARRIVE AT THE ARCHITECTURE THAT SATISFIES THE REQUIREMENTS?*

Sources of Architecture (1)

- Architecture comes from “black magic, people having ‘architectural visions’”
- Three main sources of architecture
 - theft
 - method
 - intuition
- Their ratio varies according to
 - architects’ experience
 - system’s novelty



Sources of Architecture (2)

- Theft
 - from previous similar systems
 - from literature
 - Method
 - systematic and conscious
 - possibly documented
 - architecture is derived from requirements via transformations and heuristics
 - Intuition
 - “the ability to conceive without conscious reasoning”
 - increased reliance on intuition increases the risk
- Routine design vs. innovative design

Routine Design

- Method is critical
 - an architecture built with 50% theft and 50% intuition is doomed to fail
- Standardized methods
- Similarity to previous solutions
- Theft
- Cheaper but not optimal
- Can be done by good designers
- Potential pitfall
 - over-reusing

Innovative Design

- Raw invention
- Intuition
- Derivation from abstract principles
- More optimal
- More expensive
- Must be done by great designers
- Potential pitfall
 - reinventing the wheel

Software “Architecting”

- The “architecting” problem lies in
 - *decomposition* of a system into constituent elements
 - *composition* of (existing) elements into a system
- Two idealized approaches
 - top-down
 - decompose the large problem into sub-problems
 - implement or reuse components that solve the sub-problems
 - bottom-up
 - implement new or reuse existing stand-alone components
 - compose (a subset of) the components into a system
- A realistic approach will require both

Issues in Decomposition (1)

- How do we arrive at
 - components
 - connectors
 - their configuration
- What is the adequate component granularity level?
- What constraints on components are imposed by
 - functional requirements
 - non-functional requirements
 - envisioned evolution patterns
 - system scale
 - computing environment
 - customers/users
- What assumptions can components make about one another?

Issues in Decomposition (2)

- How do components interact?
- What are the connectors in the system?
- What is the role of connectors?
 - mediation
 - coordination
 - communication
- What is the nature of connectors?
 - type of interaction
 - degree of concurrency
 - degree of information exchange

Issues in Composition

- These subsume and extend the problems of reuse in general
- Where does one locate existing
 - components
 - connectors
 - configurations
- How do we determine which elements are needed?
 - both at development-time and at reuse-time
- What is the adequate element granularity level?
- How do we ensure effective composition of heterogeneous elements?
- How do we know that we have the needed system?

Software “Architecting” Approaches

- Two general categories
 - process-driven
 - checklist-driven
- Process-driven approaches
 - broad scope
 - tailorable
 - dynamic
 - generality vs. practical usefulness
- Checklist-driven approaches
 - domain-specific
 - static
 - practical

Process-Driven Approach — OO

- Guided by analysis of the *problem* domain rather than a *solution*
- Problem domain is a set of interacting entities — *objects*
- Objects are members of families of objects — *classes*
- Classes may share traits — *subclassing, inheritance*
 - shared traits may be abstracted away by generalizing
 - new traits may be introduced by specializing
- Class instances are combined to form *applications*

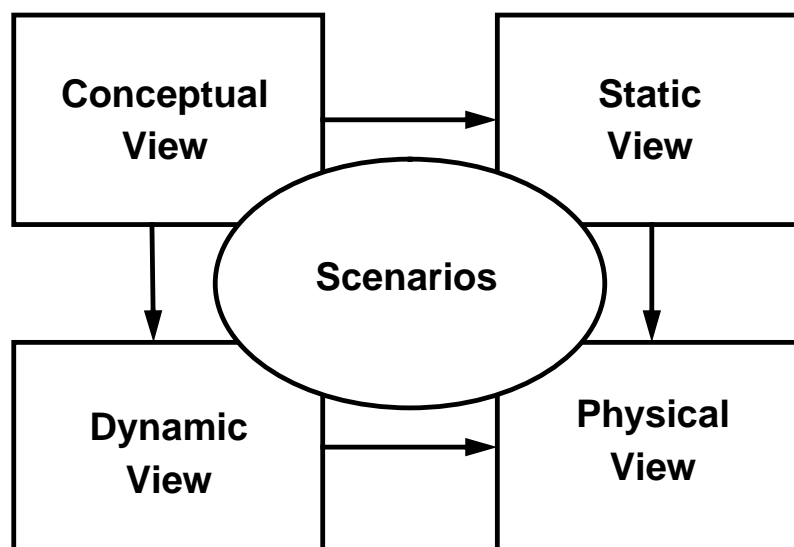
From Problem Description to OO Design

- Identify the objects in the problem domain
- Specify each object's state, behavior, and interface
- Identify commonalities among the objects
- Locate existing or form new classes to which the objects belong
- Compose the objects into an application
 - the composition is suggested by object relationships in the problem domain


Process-Driven Approach — 4+1 View Model

- 5 views of architectures
 - conceptual
 - “the object model of the design”
 - dynamic
 - concurrency and synchronization aspects
 - physical
 - mapping of software onto hardware
 - static
 - organization of software in the development environment
 - scenarios

The “4+1” View Model



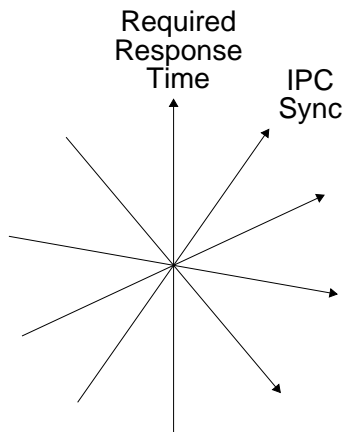
Scenario-Driven Iterative “Architecting” Approach

- Prototype, test, measure, analyze, and refine the architecture in subsequent iterations
 - Summary of the Approach:
 - choose scenarios and identify major abstractions from it
 - map the abstractions to the 4 blueprints
 - implement, test, measure, and analyze the architecture
 - capture design guidelines and lessons learned
 - select additional scenarios and reassess the risks
 - fit new scenarios into the original architecture and update blueprints
 - measure under load, in real target environment
 - review the blueprints to detect simplification/reuse potential
 - update rationale
- 

Checklist-Driven Approaches

- Enumerate the design space
 - multi-dimensional
 - each dimension is a variation
 - in a single system characteristic
 - in a single design choice
- Design space dimensions are correlated
 - indicate appropriate and inappropriate combinations of design choices
 - example negative correlation
 - a distributed system cannot be implemented with a monolithic program structure

Design Spaces



- A design is a single point in the multi-D space
- Dimensions can be functional and structural
- Example dimensions
 - communication
 - events
 - state
 - state deltas
 - state and events
 - control flow
 - single thread
 - multi thread
 - multi process

Design Rules

- Used to arrive at designs within design spaces
 - e.g., distributed system organization favors event-based communication
- +/- weight associations of design alternatives and their combinations from two or more dimensions
- Allows calculation of a “design value”
 - sum of all weights of all applicable rules
 - best designs have the highest value
- Two categories of design rules
 - relationship between functional and structural dimensions
 - system requirements drive structural decomposition
 - relationship between structural and structural dimensions
 - internal design consistency
- 600+ rules for UI systems

Case Study Analysis

- Large system consisting of six subsystems
 - C4
 - NOSS
 - Corporate databases
 - Billing
 - Downstream systems
 - Quick service
- Major C4 functions
 - service negotiation
 - account management
 - trouble call management

Functional Requirements

- Handle customer transactions
- Support long and/or interrupted transactions
- Interact with other subsystems
- Ensure → ANALYSIS COMPONENTS
 - service availability
 - data integrity
 - configuration validity
 - information completeness
- Resolve conflicting events → CONFLICT RESOLUTION COMPONENT
- Advise customers of available services → SALES COMPONENT
- Support for future changes → CLOCK COMPONENT

