Architecture

• An architecture (of a system) is
  – “The fundamental organization of a system embodied by its components [jil: building blocks], their relationships to each other [jil: connectors and interfaces, dependencies] and to the environment and the principles guiding its design [jil: rationales for choices, rules & constraints for building blocks and connectors] and evolution”

  (IEEE Standard P1471 Recommended Practice for Architectural Description of Software-Intensive Systems)

• An architecture description is
  – a collection of models organized into views that examine a system from a certain viewpoint defined by the concern of a stakeholder
    – for understanding, analysis, communication, construction, documentation
    – ….for answering questions

• Views include structure and behavior (scenarios)
Kruchten’s views and stakeholders

- **Article**: Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture, IEEE Software 12 (6), Nov. 1995, pp42-50
  - the paper does not describe it this explicit, but is interesting as an historic background

- **Stakeholders**: users, programmers, system engineers, system integrators

- **Views**:
  - **User**: *Logical view* – concerns: using the system, associated qualities
    - externally visible structure, typically modeled as an object diagram
  
  - **Programmer**: *Development view* – concerns: implementing and modifying the system
    - decomposition into subsystems, organization into files, components and modules
  
  - **System integrator**: *Process view* – concerns: performance aspects
    - units of deployment (‘programs’, ‘components’) and concurrency (‘threads’, ‘processes’)
  
  - **System engineer**: *Physical view, Deployment view* – concerns: putting systems together
    - computers, networks, infrastructure, distribution, mapping of software to hardware (deployment), protocols
  
  - (+1) Scenario’s (structured use cases) - sets of interactions with the system integrating the (models in the) views and providing behavioral models inside the views
Architectural styles (patterns)

• Remember: An architecture is “the fundamental organization of a system embodied by
  – its components [jl: building blocks],
  – their relationships to each other [jl: connectors and interfaces, dependencies] and to the environment
  – and the principles guiding its design [jl: rules & constraints for building blocks and connectors] and evolution”

• An architectural style is a coherent set of design decisions concerning the architecture
  – a combination of a typical (de)composition
  – and typical choices for connectors, components (building blocks) and behavior
  ….a generic solution for a class of problems

• We also have interaction styles, concerning just the interaction between building blocks
  – the nature of the connectors, and their organization
Definition: protocol

• **Protocol**: A formal set of rules that dictates how information exchange as well as interaction between entities (can be objects, devices, execution threads, etc.) should take place.

*The rules specify*

– the format of the messages exchanged;
– a number of different protocol states and what messages are allowed to be sent in each state; these states determine, among others, the order of the messages.
– timing constraints and other quality properties, if any.

• **Note**:

– ‘message’ includes function calling
– a protocol defines a service (the semantics of the message exchanges) to its users
– one may specify a protocol without being explicit about the overall service it realizes
Definition: service

- **Service**: a contractually specified overall functionality (semantics) of an entity.

- **Service quality**: extra-functional properties of a service (e.g. speed, reliability, ...).

- **Service interface (API)**: actions (“primitives”) and responses that make the service available; these responses can be autonomous (“events”, “call-backs”). In addition, a specification that
  - describes their effect on state variables and parameters, as well as their results;
  - describes rules as how and in what sequence to call them;
  - describes the functional and non-functional properties of sequences of calls.

  *(i.e., the interaction or access protocol)*
Definition: carrier, binding

• A protocol can be realized directly using basic services (e.g. hardware transport, machine instructions). If not, it relies on another service and corresponding protocol, called the carrier.

• The functionality and properties that a protocol provides is called the provided service. The provided service is often specified by an API for the protocol.

• A protocol requires services from its carrier and any carrier providing these services can be used. The set of rules that specify how a protocol is mapped onto a carrier is called a binding.

• In general, binding refers to establishing a relation between a reference and a referred object.
  – this interpretation of the term is slightly different than protocol binding
  – usually, the meaning can be derived from the context
Life cycle

• The life cycle of a product or system is the series of stages it goes through from inception to decline

• A typical life cycle for a software system is given to the right

• More detail is obtained by adding information regarding the activities in the stages

• Notes:
  – also system parts have life cycles, affecting the overall system
  – the life cycle should also address evolution, redesign
Programming model

• A Programming Model is set of concepts, primitives and combinators to express a computation or system behavior

• Examples:
  – the language C: variables, functions, assignments, expressions, function calls, repetition, selection, sequential composition
  – C++ adds objects, classes, inheritance as concepts, and primitives to use them
  – An RPC programming model has concepts and primitives to manage and express an RPC-based system
  – languages like Haskell, Perl, Prolog
  – Google’s MapReduce: functions Map and Reduce
(Programming) framework

- A framework consists of
  - a ‘static’ part
    - programming model, data model
      - libraries
    - life cycle model
    - methods or tooling for development
  - a ‘dynamic’ part
    - a run-time system, or platform
      - entirely separate entity or a library
    - a set of services
      - provided by the platform
      - e.g. binding, installation
    - a process model

- A framework has views, e.g.,
  - logical view for the framework user (which is an application developer)
    - programming model
    - visible services
  - development view for the framework developer, programmer
    - the logical organization of the framework tools and platform
    - the services structure
    - the code
  - process view for developer and framework installer
    - the processes in the framework, the connection to the OS, the protocols
  - deployment view
Privacy, Safety, and Security

- **Privacy**: control over personal information
- **Safety**: freedom from danger or risk on injury resulting from recognized but potentially hazardous events
- **Security**: regulating access to (electronic) assets according to some policy
  - *policy*: allowed and disallowed actions
  - *security mechanisms*: can be regarded as enforcing the policy
- Privacy and safety restrictions result in *security policies*
  - *security for privacy and security for safety*
- In addition: security for protection of the business case
  - *security for the business case*, quite often an *availability concern*