Software Architecture

Jim Fawcett

Software Modeling

Definitions

- “An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces … together with their behavior as specified in the collaborations among those elements, …”[1]
- “… abstract away some information from the system … and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction.”[2]
- “…designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.”[3]

References

1. Booch, Rumbaugh, and Jacobson. *The UML Modeling Language User Guide*. Boston: Addison-Wesley, 1999.
2. Bass, Clements, and Kazman. *Software Architecture in Practice*. Boston: Addison-Wesley 1997.
3. Shaw, Mary, and David Garlan. *An Introduction to Software Architecture*. In *Advances in Software Engineering and Knowledge Engineering*, volume I, ed. V. Ambriola and G. Tortora. World Scientific Publishing Company, 1993.

Definitions of Software Parts

- **Class:**
  - C# language construct that groups methods and data to satisfy a single responsibility
- **Package:**
  - Single C# file with prologue comments, class definitions, and test stub
- **Module a.k.a. subsystem:**
  - A collection of packages that focuses on a closely related set of operations
- **Program:**
  - A set of packages or subsystems that builds to create an executable
- **System:**
  - A set of cooperating programs that collaborate to provide some useful facility

Architecture: Our Definition

- Software architecture
  - An abstraction for a software part (system, program, package, class) that focuses on uses, structure, issues, and risks
  - **Uses**: How users (people and other software) interact with a part and how the part responds
  - **Structure**: The collection of parts and their interactions and dependencies
  - **Issues**: Things that developers are concerned about, like complexity for a large system, ease of use, robustness
  - **Risks**: Potential for unwanted results, often related to performance, safety, and financial and security threats

Intent

- When we develop software, we want our software to have an architecture developed explicitly, not accidentally.
- Its purpose is to allow us to think critically about a product we are developing before committing to code.
- For large systems an architecture may be represented by a, possibly large, document.
- For smaller systems and programs it may be presented on a web page or small collection of diagrams and notes, bound together in some form of accessible container.

Architecture Level

- Systems
  - We usually think of an architecture as describing some large, distributed system.
- Packages
  - But packages also have architectures: uses, users, structure, and issues.
  - Package structure relates to the package's classes and how they interact.
- Classes
  - Even a class has an architecture defined by its methods, data structures, and how they interact.

What Is Software Architecture?

- The architecture of a software system captures major features and design ideas for a software development project.
  - Describes relationship of users with the system
  - Describes structure and organizing principles of the system
    - Major partitions within the system and their interfaces
    - Responsibilities of, and resources needed by, each partition
    - Design concepts: data structures, algorithms, data flows that help developers understand and implement their piece of the system
  - Identifies major threads of execution
    - A thread is the sequence of activities that result from some system event. Examples are system startup, response to operator requests, and processing of errors.
  - Identifies critical timelines and risk areas
    - A timeline is a time-based budget for critical threads.
    - A risk area identifies objectives and requirements that will be difficult to meet under the current architectural and design concept or susceptibility to threats.

Architectural Concerns

- Software architecture is concerned with:
  - **Goals:**
    - Main objectives of the system
  - **Uses:**
    - How people and other software will interact with the system
  - **Tasks:**
    - Activities for a system and its major partitions
  - **Partitions:**
    - Subsystems, packages, and classes that make up the system
    - responsibilities
  - **Interactions:**
    - The relationships and data flows between partitions, and assumptions that partitions have about each other
  - **Events:**
    - Any occurrence that affects system activities
  - **Views:**
    - Appearance of the system to users and its designers
  - **Performance:**
    - Efficient use of computer resources—processor cycles, network bandwidth, memory

End of Asynchronous Presentation

Please read the remaining slides, where each of the architecture concerns are discussed, before joining the first synchronous session.

Uses

- Uses describe the way users and other software components interact with the system.
  - What is the user trying to accomplish?
  - What are the required inputs that the user supplies?
  - What are the system outputs that the user expects?
  - What controls will the user want to affect system operation?
- Uses are often developed as scenarios, called use cases.
  - Each scenario describes one or more of the following:
    - User roles, e.g., developer, manager, quality assurance.
    - Mode of operation, e.g., data collection, data analysis, data presentation.
    - Responses to specific important events, e.g., initialization, user inputs, computational errors, system output.
- Are there effective uses that go beyond the system specification but would be relatively easy to implement?
  - Can we select a structure that will be easy to extend to these new applications without significant impact on meeting the current requirements?
  - This could result in efficient development of new products and services.
- Impact on design
  - How will the identified uses affect the structure of the system and its design?

Tasks

- Tasks are a high-level list of the activities that the system will need to carry out.
  - First developed for the system as a whole.
  - Later, allocated to the major system partitions.
- Tasks are usually presented as lists and in activity diagrams.
  - Activity diagrams are like flow charts but at a higher level.
  - They describe activities that are important for the system or its major partitions.
  - Activity diagrams show required sequencing and synchronization of tasks.
  - When software is implemented, tasks allocated to each package are described in the package's manual page.

Partitions

- Partitions represent the grouping of system activities into logical and physical entities.
  - Package and module diagrams show the physical packaging of system processing into files.
  - Activity and data flow diagrams represent the partitioning of system activities into logical processes, showing the flow of information between the external environment and each process.
  - Classes show the logical partitioning of system data and processing into low-level program constructs.
- Partitions are the second most important part of the architecture concept, after the definition of its tasks.
  - Sequence of concept development is often: (1) uses, (2) tasks, (3) partitions, (4) interactions, (5) events, and (6) views.

Interactions

- Interactions describe the relationships between system partitions. They are described by:
  - **Data flow diagrams:**
    Used in the early phases of architecture and requirements development
  - **Package diagrams:**
    Describe static relationships between the system's physical partitions
  - **Class diagrams:**
    Describe the static relationships between the system's logical components
  - **Event trace diagrams:**
    Show the dynamic relationships between system components
  - **Structure charts:**
    Describe the relationships between the system's functions
  - **State charts:**
    Describe the dynamic relationships between the system's computations

Events

- Events describe specific occurrences to which the system must respond, or that affect its modes of operation.
  - Events are critically important for real-time systems, e.g., systems that must respond to asynchronous events from the outside environment.
  - For these systems, architecture development may revolve around the definition of critical threads.
    - A thread, as defined by the architecture concept, is all the processing that results from a specific event, e.g., a radar detection, user input, power on, computational error.
    - Many threads are defined, then sorted by importance, relative to the system requirements. The architecture isn't complete until processing that will support system requirements for each of the critical threads is defined.
  - Threads are usually described by activity and/or event trace diagrams.
  - In some non-real-time systems events play only a minor role in developing the system architecture.

Views

- Views are used in two ways:
  - Views describe the user interface as it appears to the user.
    - Layouts of controls and screens.
    - Screen shots showing what the user will see when entering data.
    - Screen shots showing what the user will see when observing operation.
    - Each of these views is accompanied with text describing how the user interacts with the controls and screens.
  - Views also describe the most important data structures and algorithms as they appear to the developer:
    - Data structure diagrams are ad hoc diagrams that show how data elements relate to each other.

Performance

- Level of communication affects performance by orders of magnitude:
  - Within a process
  - Between local processes on a single machine
  - Between machines in a network
  - Between networks, e.g., across the Internet
- Lazy communication:
  - Send information only when needed
  - Send only the specific information needed
- Data caching:
  - Store information locally so that it need not be requested repeatedly
- Minimize remote connectivity:
  - Connections consume threads, CPU cycles, memory
  - Make connection time least necessary to complete request, then disconnect.

Analysis

- Analysis of the architecture of a software system entails:
  - Analysis of scale
    - How many users, files, storage size, working set size?
  - Analysis of load
    - Number of concurrent users, open files, open connections
    - Peak and average data flows between process, machines, networks
  - Analysis of timelines
    - How long to initialize and perform key operations?
  - Analysis of function
    - What tasks and operations are essential?
    - How should they be organized?
      - Logical organization is easier to understand, develop, and maintain
      - Data flow often dominates performance
  - Analysis of risk
    - High-risk tasks and operations
  - Means of risk abatement
    - What steps can we take to minimize the impact of risk areas?

ENGINEERING@SYRACUSE