Architectural Analysis and Design how-to.

The starting point for the architectural or high level system design are the User Requirements & Use Cases together with accompanying documents. A design presents a model which is an abstraction of the system to be designed. The step from a real world system to abstraction is analysis for which we refer to the literature [1,2,3] or courses [4].
The procedure for producing an architectural design outlined below is taken from the Rational Unified Process (RUP) [5]. This is a well known process, suitable for large projects. It provides good documentation, templates and support tools for which CERN has acquired a license (the process itself including the use of templates, concepts and ideas is free).

Architectural Analysis and Design

Definition of Architecture: The highest level concept of a system in its environment [IEEE]. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successfully smaller components and interfaces.
Decompose larger systems into a hierarchy of smaller subsystems (the UML notation for a (sub)system is a package with stereotype <<(sub)system>>): The distinction between system and subsystem is obviously relative and before you start you will have to decide where you wish to stand and what your scope is. What follows is applicable to each (sub)system in the hierarchy.
To model a system of subsystems, proceed in three steps (iteratively, as they are related):

1. Identify the subsystems. This obviously determines their content.
2. For each subsystem, specify its context:. This determines dependencies, interactions and interfaces that must be designed to collaborate. The actors are the external world and its neighboring subsystems.
3. Model the architecture of each subsystem as for the whole system.

Identification of Subsystems (Decomposition)

Boundaries are often determined by technical, political, organizational or legacy reasons. Hints may be provided by the requirements document (e.g. grouping). Subsystems should be logically and physically loosely coupled to minimize the complexity of the interfaces and allow independent development, deployment and management. Separate software from hardware dominated components.

Context

Decomposition results in a set of logical packages corresponding to (sub)systems whose interactions are identified by the analysis of the use cases of the requirements document. Ultimately these packages will contain class diagrams whose methods implement interactions. Some of the classes will define the interfaces. Interfaces with the external world are determined by dependencies in the requirements document.

Modeling

The RUP suggests to present the following six views of the model [5]. The first two are mandatory, the remaining four are recommended.

1. Specify a use case view of the system. Analyse all use cases that describe the behavior of the system including testability. Model static aspects with use case diagrams and dynamic aspects with interaction (collaboration and sequence), statechart and activity diagrams.

2. Specify a design view of the system. This is a logical view. Identify classes, interfaces and collaborations. Model static aspects with class and object diagrams; dynamic aspects with statechart and activity diagrams.
3. Specify a process view of the system.This is a logical view. Identify threads and processes; analyse synchronisation and inter-process communication. Use diagrams similar to those of the design view.
4. Specify an implementation view of the system. This is a physical view. Identify the components that are used to assemble and release the system. Model static aspects with component diagrams; dynamic aspects with activity diagrams.
5. Specify a deployment view of the system. This is a physical view. Identify the hardware (nodes, interconnects, displays) on which the system runs. Model static aspects with deployment diagrams; dynamic aspects with interaction, statechart and activity diagrams.
6. Specify a data view of the system. Describe the architecturally significant persistent elements in the data model: overview, organization, views, relations and mapping onto databases.

Precede these views by a section “References”, “Architectural Goals and Constraints” and follow at the end by “Size and Performances” and “Quality”.

Template

A template following the suggested layout is under preparation. Many more details, suggestions and examples are available from [5].

Tools

Tools to produce UML diagrams and more generally, to assist the design process are available from [5] but also from other sources. They are mostly licensed. The Connect Forum is evaluating some of these and recommendations will follow.

References

1. M.Fowler, UML Distilled. Available from http://consult.cern.ch/books/
2. G.Booch, J.Rumbaugh, I.Jacobson, The Unified Modeling Language User Guide. Available from http://consult.cern.ch/books
3. T.Quatrani, Visual Modeling with Rational Rose, Addison-Wesley
4. J.Deacon, Object Oriented Analysis and Design, CERN Technical Training. Details from http://humanresources.web.cern.ch/HumanResources/internal/general/HN-training/default.asp
5. http://sdt.cern.ch/RUP