# Experience with Architecture-Centered Software Project Planning

Daniel J. Paulish
Robert L. Nord
Dilip Soni
Siemens Corporate Research, Inc.
Princeton, NJ 08540 USA

**Abstract**

This position paper describes our experiences with architecture-centered project planning for a software development project. A software architecture design document was the primary input to the top-down and bottom-up planning processes, and a software development plan was the primary output. The software development plan was used by each member of the development team to generate their personal schedules.

**Keywords:** Software architecture, estimation, project planning, high-level design.

## 1 Introduction

In 1995, staff members from Siemens Corporate Research (SCR) were asked to take responsibility for a Siemens software development project. Management responsible for the project wanted us to design a software architecture for the product and determine how much time and resources were required to complete the software development and to bring the product to market.

It is well known that effort and schedule estimates given in the very early stages of development can be very inaccurate [1]. We believe that schedule and resource estimates produced in the absence of a high-level architecture design have minimal value. Milestone dates planned in the absence of a design would likely be missed. That is why we decided to first complete the design of the architecture. When the design was complete, we created a project plan, schedule, and resource projections, all of which were dependent upon the software architecture of the product.

## 2 Approach

Architecture-Centered Software Project Planning (ACSPP) is applied early in the software development process, after the system requirements design is complete. Management often desires early estimates of the time and effort required to develop a new software product. In some cases these estimates may be required to determine whether or not the development project should be undertaken. Business and product planning are often based upon very early estimates, that are often very inaccurate. For example, according to [1], actual effort expended can be 1.5 times the cost estimates given after requirements specifications are complete. Actual effort expended can be 4 times the estimates given at the beginning of the project, before any feasibility analysis is done.

The approach for ACSPP is illustrated in Figure 1. A small design team initiated the high-level design of the software architecture. In parallel, we initiated top-down schedule planning using preliminary lines of code estimates as inputs to the Cocomo Model [1].


SIGSOFT 96 Workshop, San Francisco CA USA
© 1996 ACM 0-89791-867-3/96/10 ..$3.50

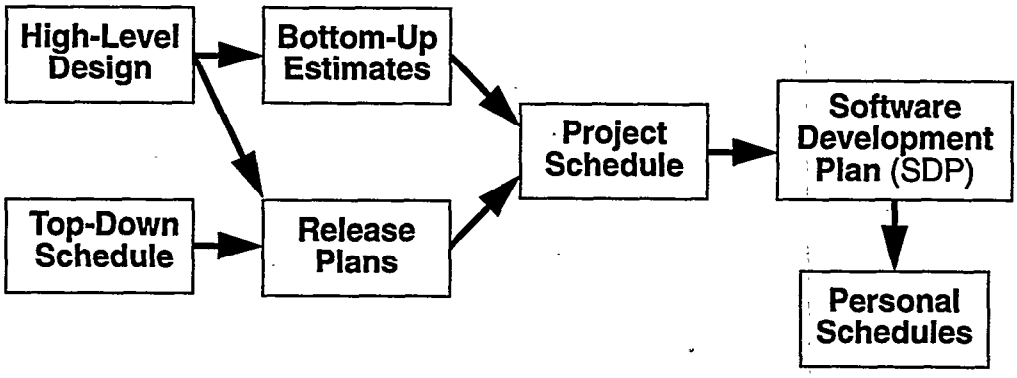


Figure 1. Architecture-Centered Software Project Planning (ACSPP).

The high-level design and the top-down schedule were used as inputs for planning software releases. The high-level design was also an input to the bottom-up estimation method where the team members estimated the size and effort for the software components. We reviewed and compared these estimates with the estimates used in the top-down schedule, and used them to generate a project schedule. The schedule, along with the resource assignments and organization, became part of the Software Development Plan (SDP). Based on the SDP, each team member developed their personal schedules.

## 2.1 High-Level Design

We designed and documented the software architecture for the product within a High-Level Design Document (HLDD). The software architecture was documented using description techniques developed at SCR [2]. These techniques document the architecture design using four architecture perspectives (Figure 2).

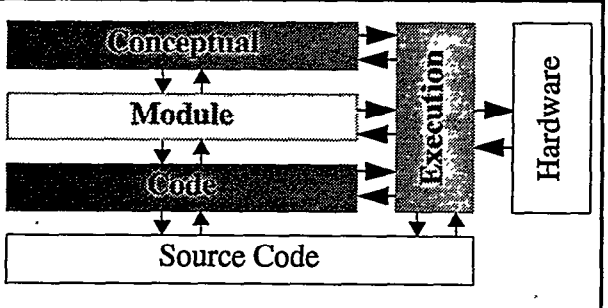


Figure 2. Four Perspectives of SW Architecture.

We explained the software architecture to the project team members through in-depth discussions. This was an iterative process that helped team members understand the high-level design and helped us to improve it.

A one page layer diagram was generated as part of the software architecture. An example layer diagram is illustrated in Figure 3 [2]. Modules within a layer can communicate with each other. Modules may only communicate with modules in the same or adjacent layers.

The architecture layer diagram provided stability and a point of reference for many aspects of the project, including technical coordination, assessment of alternative implementations, and project planning and scheduling.

## 2.2 Top-Down Schedule

In parallel with the architecture design, we initiated some investigations into top-down schedule and effort estimation using expected lines of code estimates for each subsystem. We validated the lines of code estimates by comparing the expected total lines of code for this project with other similar (competitive) products. These estimates were used as inputs to Cocomo Model calculations, which provided outputs of effort, schedule duration for major phases of development, and resource profile loading for various types of development skills.

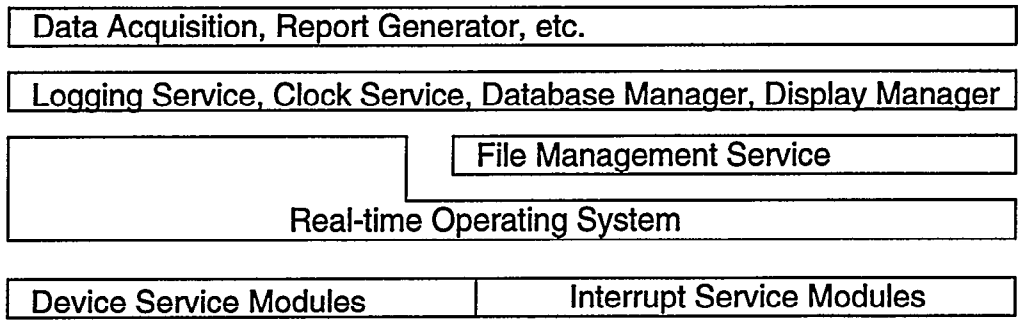


Figure 3. Architecture Layer Diagram.

### 2.3 Bottom-Up Estimates

When all of the roughly 130 components were defined, each team member did a "paper design" for their assigned components, documenting the subcomponents and dependencies. The time allocated to the paper design was limited to four hours per component. Each team member used the paper design to estimate each component's size, complexity, and coding effort. We reviewed these estimates, compared them with the top-down Cocomo Model estimates, and used them as inputs to create a project schedule.

This process helped team members understand the high-level design at a greater level of detail and provide more accurate estimates about the size and complexity of the components.

### 2.4 Release Plans

The top-down schedule specified three engineering releases before the final product release. We described the product features incorporated in each of these releases in a Feature Release Specification (FRS) document, which was completed with consultation from Marketing and Service. In a separate Component Release Specification (CRS) document, we described the component functionality necessary to implement required features in each of the engineering releases. In some cases, a component had multiple engineering releases, each supporting partial functionality. Responsibilities for these component releases were assigned to team members who later used these documents to create their own personal schedules.

We identified a software integration strategy for the components and the features that they implemented by partitioning them into three internal engineering releases. The component effort estimates were then reviewed along with the desired availability of the features within an engineering release. From that, we assigned the component developments to a time schedule and identified resources to design, code, and unit test each component. We also mapped the architecture into a development organization plan corresponding to the subsystems and components, and assigned at least one person to be responsible for each software component identified in the architecture.

### 2.5 Project Schedule

We used the top-down schedule, the bottom-up estimates, FRS, and CRS to develop a schedule skeleton for the project such that each of the internal engineering releases could be designed, coded, unit tested, integrated, and system tested. We divided each component's development into subtasks according to development phases (detailed design, coding, unit testing, and bug fixing) for each of its releases. We distributed each component's development within the schedule skeleton depending on the total estimated effort, the resources available, and the FRS and CRS. For example, the design and coding tasks for a large component would start early even though its features were not needed until later. It was sometimes necessary to modify the FRS and CRS in order to fit the component development subtasks within the schedule skeleton using the available resources.

The subtasks for each component, and their integration dependencies were incorporated in the project schedule. We had "high-confidence" in the resulting project schedule, in that the actual release date would be within 15-20% of the time estimated.

### 2.6 Software Development Plan

The software development plan (SDP) was a short document which included the schedules, engineering release definitions, staffing requirements, subcontractor utilization, project organization, cost estimates, development tools and procedures, task assignments, and hardware platform. It referred to the software development process, high-level design document (HLDD), feature release specification (FRS), and component release specification (CRS). The SDP summarized when, how, and with whom the software product would be developed. The SDP contained a description of the organizational structure for the project and a description of the roles of the team members.

### 2.7 Personal Schedules

The completed SDP, including the project schedule, was received by each team member to create their own detailed personal schedules. We used the inputs from the personal schedules to provide more detail to and update the project schedule which was then frozen as a baseline schedule. Since the tasks identified in the project schedule were consistent with the detailed activities identified by the team members, they had good ownership of the schedule by this time. The personal schedules were monitored weekly and the project schedule was updated every two weeks.

## 3 Conclusions and Lessons Learned

We have described our limited experience with planning a software development project using the software architecture.

We found that a well defined architecture was essential for the entire project planning effort. It was equally essential for the architecture to be well understood by the team members in order to get a much better grasp of what was necessary to implement the software product. We spent a lot of time and effort in communicating and reviewing the high-level design with all the team members, and improving the architecture description.

We found the architecture layer diagram to be a very valuable tool for the development team. Such a summary picture of all the major software components and their relationships provided stability and a point of reference for many aspects of the project. It gave the team and management an overview of what needed to be developed in order to implement the product. It was helpful for deciding if components from other products could be reused to implement the new product, and for planning and visualizing the internal releases and integration steps.

The four-hour paper design served two objectives: a more detailed understanding of the architecture and more accurate estimates. These then became the basis for the project work packages leading to a bottom-up creation of the project schedule.

Using both top-down and bottom-up approaches to estimation helped increase team members' confidence in and ownership of the resulting project schedule.

## References


1 Boehm, B. W. (1981), *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.

2 Soni, D., Nord, R. L., and Hofmeister, C. (1995), "Software Architecture in Industrial Applications", *Proc. of the 17th International Conference on Software Engineering*, Seattle, WA.