CALIFORNIA STATE UNIVERSITY
FULLERTON™

COLLEGE OF ENGINEERING
AND COMPUTER SCIENCE

Advanced Software Process

Part II: The Repeatable Process

6. The Project Plan

Dave Garcia-Gomez
Faculty / Lecturer
Department of Computer Science

Course Roadmap

Part I: Software Process Maturity
- 1. A Software Maturity Framework
- 2. The Principles of Software Process Change
- 3. Software Process Assessment
- 4. The Initial Process

Part II: The Repeatable Process
- 5. Managing Software Organizations
- 6. The Project Plan
- 7. Software Configuration Management (Part I)
- 8. Software Quality Assurance

Part III: Defined Process
- 9. Software Standards
- 10. Software Inspections
- 11. Software Testing
- 12. Software Configuration Management (continued)
- 13. Defining the Software Process
- 14. The Software Engineering Process Group

Part IV: The Managed Process
- 15. Data Gathering and Analysis
- 16. Managing Software Quality

Part V: The Optimizing Process
- 17. Defect Prevention
- 18. Automating the Software Process
- *19. Contracting for Software*
- *20. Conclusion*

The Project Plan

- Project Planning Principles
- Project Plan Contents
- Size Measures
- Estimating
- Productivity Factors
- Scheduling
- Project Tracking
- The Development Plan
- Planning Models
- Final Considerations

The Project Plan

The project plan defines the work and how it will be done.

- A definition of each major task
- An estimate of the time and resources required
- A framework for management review and control

When a project plan is properly documented, it is a benchmark to compare with actual performance.

Project Planning Principles

1. While requirements are initially vague and incomplete, a quality program can only be built from an accurate and precise understanding of the users' needs.

   The project plan thus starts by mapping the route from vague and incorrect requirements to accurate and precise ones.

Project Planning Principles

2. A conceptual design is then developed as a basis for planning.

   This initial structure must be produced with care since it generally defines the breakdown of the product into units, the allocation of functions to these units, and the relationships among them.

   Since this provides the organizational framework for planning and implementing the work, it is almost impossible to recover from a poor conceptual design.

Project Planning Principles

3. At each subsequent requirements refinement, resource projections, size estimates, and schedules are also refined
4. When the requirements are sufficiently clear, a detailed design and implementation strategy is developed and incorporated in the plan.

Project Planning Principles

5. As various parts of the project become sufficiently well understood, implementation details are established and documented in further plan refinements.
6. Throughout this cycle, the plan provides the framework for negotiating the time and resources to do the job.

Planning Considerations

- With rare exceptions, initial resource estimates and schedules are unacceptable.
- It is best to reach early agreement on the essential functions and to defer the rest until later.
- Quality products are an artistic blend of needs and solutions, requiring harmonious teamwork between the users and the software engineers.

The Planning Cycle

Iterative plan negotiation process [Figure 6.1]

1. The cycle starts with the initial requirements.

2. The response to every demand for a commitment must be “I understand your requirement and will produce a plan with that objective, but without a plan, I cannot make a commitment.”

The Planning Cycle

3. The plan is produced by first breaking the work into key elements, called a Work Breakdown Structure (WBS).
   This implies that a conceptual design has been developed.
4. The size of each product element is estimated.
5. The resource needs are projected.
6. The schedule is produced.

The Planning Cycle

The Software Development Planning Cycle [Figure 6.1]

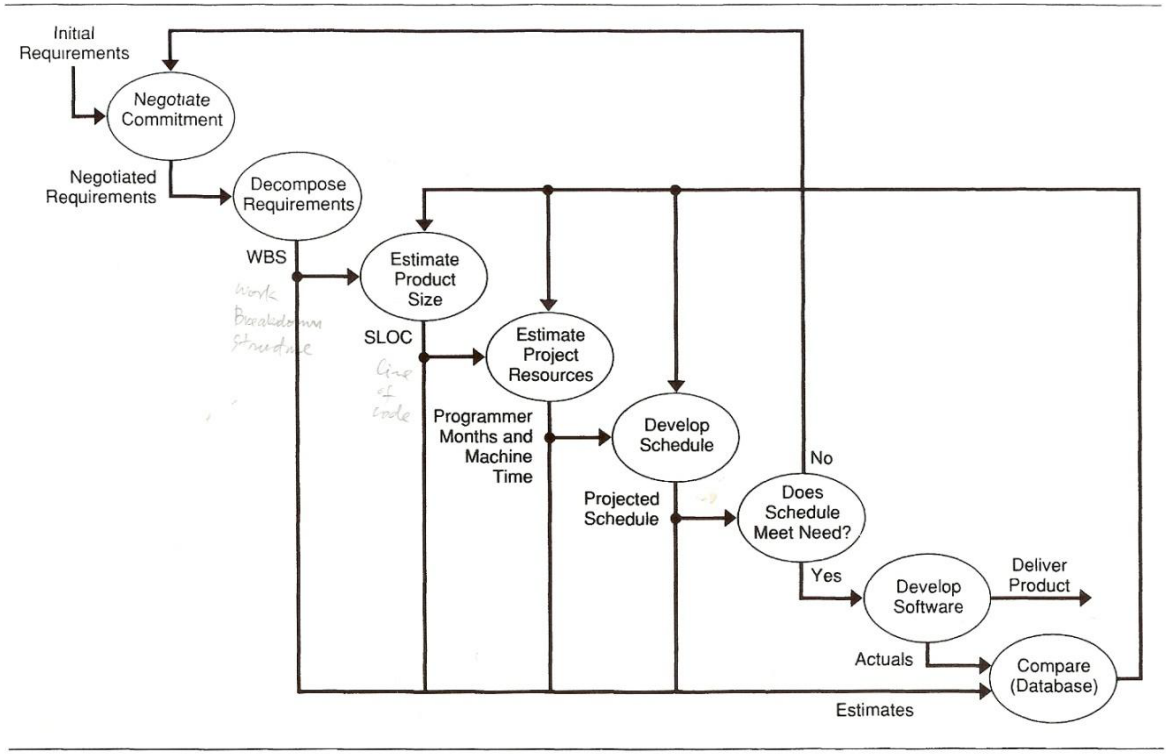

FIGURE 6.1
The Software Development Planning Cycle

Project Plan Contents

- Goals and objectives
- Work Breakdown Structure (WBS)
- Product size estimates
- Resource estimates
- Project schedule

Goals and Objectives

- The project's goals and objectives are established during the requirements phase.
- This is also a negotiation period between the software engineers and the users on what is to be done, how success will be measured, and how much time and resources are needed.

The Work Breakdown Structure (WBS)

- Project planning starts with an estimate of the size of the product to be produced.
- This estimation begins with a detailed and documented breakdown of the product into work elements.
- The breakdown generally has two parts:
  - The product structure
  - The software process
- The WBS then provides the framework for relating them.

Size Measures

- The measure used in program size estimation should be reasonably easy to use early in the project and readily measurable once the work is completed.
- Subsequent comparisons of the early estimates to the actual measured product size then provide feedback to the estimators on how to make more accurate estimates.
- Function points start from the user's perspective and estimate the size of the application [Albrecht 1979].

Line-of-Code Size Estimates

- Line-of-code (LOC) estimates typically count all source instructions and exclude comments and blanks [Jones 1978].
- Automated tools, for example, are often designed to count semicolons.
- Even though it is difficult to estimate lines of code from high-level requirements statements, this measure does facilitate a learning process that leads to improved estimating.

Line-of-Code Size Estimates

- The finer the product structural detail, the more likely relevant examples are available, and the more accurate the subsequent estimates.
- Perhaps the most important advantage of the LOC is that it directly relates to the product to be built.

Function Points

- With function points (FP), initial application requirements statements are examined to determine the number and complexity of the various inputs, outputs, calculations, and databases required.
- By using values that Albrecht (1979) has established, points are assigned to each of these counts.

Function Points

- These points are then summed to produce an overall function point rating for the product.
- Based on prior experience, this final function point figure can be converted into a reasonably good estimate of the development resources required.

Estimating

- Once a size measure has been established, an estimating procedure is needed.
- Probably the most accurate one is the Wideband Delphi Technique originated by the Rand Corporation.
  1. A group of experts is each given the program's specification and an estimation form.
  2. They meet to discuss the product and any estimation issues.
  3. They then each anonymously complete the estimation forms.

Estimating

4. The estimates are given to the estimate coordinator, who tabulates the results and returns them to the experts.
5. Only each expert's personal estimate is identified; all others are anonymous.
6. The experts meet to discuss the results, revising their estimates as they feel appropriate.
7. The cycle continues until the estimates converge to an acceptable range.

Estimating Inaccuracies

- Regardless of the method used, software estimates will be inaccurate.
- The inaccuracies must be compensated for, however, because a poor size estimate results in an inappropriate resource plan and a generally understaffed project.
- To date, at least, software size estimates have almost invariably erred in being too low.
- These low estimates typically result in a last-minute crisis, late delivery, and poor quality.

Estimating Inaccuracies

- There are two types of estimating inaccuracies, and they can both be compensated for, at least to some degree.
- Normal statistical variations can be handled by using multiple estimators.
- The other source of error is estimating bias caused by the project stage at which the estimate is made.
  - The earlier the estimate, the less is known about the product and the greater the estimating inaccuracy.

Estimating Contingencies

- When programmers estimate the code required to implement a function, their estimates are invariably low.
- While there are many potential explanations, the important point is that their optimism is a relatively predictable function of project status.
  - Code may grow at the completion of each project phase.
  - These amount should be added as an estimating contingency at the end of the indicated project phase.

Estimating Contingencies

- Every software size estimate should include a contingency, and the only debate should be over the amount to use.
- Because contingencies inflate the resource estimates, add to the schedule, and increase costs, they are energetically resisted by almost everyone on the project.
- Experience shows, however, that until enough is known to partition the product into modules of a few hundred LOC each, substantial contingencies are required.

A Size Estimating Example

Example Program Size Estimate [Table 6.2]

- Hypothetical satellite ground station control program estimated during the requirements phase

TABLE 6.2
EXAMPLE PROGRAM SIZE ESTIMATE—SATELLITE GROUND STATION CONTROL PROGRAM

Date: 5/25/87 Estimator: WSH Program: Satellite

Component	Base KLOC*	Contingency %	Contingency KLOC*	Total KLOC*
A Executive	9	100	9	18
B Function Calculation	15	100	15	30
C Control/Display	12	100	12	24
D Network Control	12	200	24	36
E Time Base Calculation	18	200	36	54
Total	66	145	96	162

*KLOC = thousands of lines of source code

Productivity Factors

- Once we have an estimate of the amount of code to be implemented, the amount can be converted into the number of programmer months and time required.
- While some programmers can intuitively estimate their work, most of us need a more orderly procedure, particularly when the projects are large and many people are involved.

Productivity Factors

- This involves productivity factors, or standard factors for the number of lines of code that, on average, can be produced per programmer month.
- There are many kinds of potentially useful software productivity factors, and each organization should select those most relevant to their situation.

Productivity Factors

- Productivity data
  - Organization productivity data
  - Developing productivity data
  - Using productivity data
  - Resource estimating
- Organizations can gather their own data and compare it with some of these factors.

Organization Productivity Data

- In productivity calculations, it is important to use factors that relate to the specific organization doing the work and not to use more factors than the available data warrants.
- Since all productivity factors are at best averaged data from several projects done by different people with various levels of ability, it is not wise to try to account for more than a few of the most significant variables.

Organization Productivity Data

- Because so many factors affect productivity and because there is no way to adjust standardized productivity numbers to account for all these variations, each organization should gather its own data to use as a baseline for productivity calculations.
- Adjustment can then be made to account for those variations that are considered most significant.

Organization Productivity Data

- Organizations can develop their own productivity factors by examining a number of recently produced programs, counting their lines of code in a consistent and defined manner, and calculating the programmer months (PM) required to do the job.

Developing Productivity Data

- Since software organizations generally have some records of the work they have produced, the first step in developing productivity data is to see what is available.
- With this information, the following approach should produce the needed data:

Developing Productivity Data

1. Identify a number of recently completed programs that are as similar to the new program as possible in size, language used, application type, team experience, and so forth.
2. Get data on the size, in LOC, of each project. Try to use the same counting scheme for each program.

Developing Productivity Data

3. For modified programs, note the percent of code modified and count only the number of new or changed LOC in the productivity calculations.
   Special management emphasis on the advantages of reuse is needed to counteract the bias toward new code that this measure might cause.

Developing Productivity Data

4. Obtain a count of the programmer months expended for each project, but be sure to include or exclude the same labor categories for each program.
   Generally the categories to include are direct design and implementation programmers, test, and documentation.
   To minimize the number of variables, it is often wise to exclude SQA, managers, clerical, requirements, and computing operations.
   The requirements effort in particular should often be excluded because it is highly dependent on customer relationships and application knowledge.

Using Productivity Data

- With the basic data in hand, it is now possible to derive base productivity numbers.
- Since the data gathered will be for various sizes and classes of programs, some adjustments may be needed.
- With some local data, it is thus reasonable to examine some of the published productivity figures to see which ones might apply to your organization.

Resource Estimating Example

- The actual process of making a resource estimate is quite simple once the size estimate and productivity factors are available.
- Program Resource Calculation Example [Table 6.6]

Scheduling

- Once the total resource needs have been calculated, the project schedule can be developed by spreading these resources over the planned product development phases.
- This is best done by using data on the organization's historical experience with similar projects.
- When such data is not available, publicly available factors can be used for guidance.

Scheduling

- Once the project resource distribution is known, an overall project schedule can be produced as follows:
  - Based on the overall project schedule objective, a staffing plan is developed.
  - A preliminary schedule for each phase is next established by comparing the cumulative resource needs with those expected to be available. An initial schedule is then made.
  - This preliminary plan is then reviewed to ensure that reasonable staffing assignments can be made consistent with this schedule and resource profile. Adjustments are generally required

Project Tracking

- One requirement for sound project management is the ability to determine project status.
- The planning process should thus produce a schedule and enough check-points to permit periodic tracking.
- One way to do this is with earned-value project scheduling [Snyder 1976].
- Schedule Tracking Example [Table 6.9]
- Module Checkpoint Plan [Figure 6.5]

The Development Plan

- After completing the estimates and schedule, the full development plan is assembled in a complete package and submitted to management for review and approval.

The Development Plan

Software development plan [Figure 6.6]

- Project purpose and scope
- Project goals and objectives
- Organization and responsibilities
- Management and technical controls
- Work definition and flow
- Development environment
- Software development methodology
- Configuration management
- Verification and validation
- Quality assurance provisions

Planning Models

- Software cost models can be helpful in making product plan, but they should be used with care.
  - COCOMO (Constructive Cost Model) [Boehm 1981]

    http://sunset.usc.edu/Research_Group/barry.html

    http://cost.jsc.nasa.gov/COCOMO.html
  - SLIM (SW Life Cycle Mgmt) [Putnam 1978]

    http://www.qsm.com/tools/index.html

    http://yunus.hun.edu.tr/~sencer/cocomo.html

Planning Models

- Experience has shown that models, if not calibrated to the specific organization's experience, can be in error by 500 percent or more.
- Models should thus be used to augment the estimating process and not to replace it.

Planning Models

- No model can fully reflect the product's characteristics, the development environment, and the many relevant personnel considerations.
- Once the size and resource estimates have been made, it is necessary to spread these resources over the planned schedule.

Planning Models

- If the organization does not have a good historical basis, a number of modeling techniques can be of help.
- Models such as COCOMO and SLIM, for example, can be used effectively to check the estimates for errors or oversights and to help in assessing risks.
- Under no circumstances, however, should such models be relied on to produce the final estimates.

Final Considerations

- The most important single prerequisite to good software cost estimating is the establishment of an estimating group.
- The group ensures that the cost estimate and actual performance data are retained in a database, that the available data is analyzed to establish the productivity factors and contingencies, and that the development groups are competently assisted in preparing and documenting their plans.

Final Considerations

- When development teams are so supported, they are in the best position to capitalize on their planning experience and to progressively improve their estimating accuracy.
- While experience is the key, experienced SW groups rarely improve their estimating accuracy to this degree without the support of an orderly planning process.

References

Humphrey, Watts S., *Managing the Software Process*, The SEI Series in Software Engineering, Addison-Wesley, 1989. (29th Printing, May 2003) (ISBN 0-201-18095-2)