Project Planning

Software Project Planning

- Software project planning encompasses five major activities
  - Estimation, scheduling, risk analysis, quality management planning, and change management planning
- Estimation determines how much money, effort, resources, and time it will take to build a specific system or product
- The software team first estimates
  - The work to be done
  - The resources required
  - The time that will elapse from start to finish
- Then they establish a project schedule that
  - Defines tasks and milestones
  - Identifies who is responsible for conducting each task
  - Specifies the inter-task dependencies

Observations on Estimation

- Planning requires technical managers and the software team to make an <u>initial commitment</u>
- Process and project metrics can provide a <u>historical perspective</u> and valuable input for generation of quantitative estimates
- Past experience can aid greatly
- Estimation carries <u>inherent risk</u>, and this risk leads to uncertainty
- The availability of historical information has a <u>strong influence</u> on estimation risk

Observations on Estimation (continued)

- When software metrics are available from past projects
  - Estimates can be made with greater assurance
  - Schedules can be established to avoid past difficulties
  - Overall risk is reduced
- Estimation risk is measured by the degree of uncertainty in the quantitative estimates for cost, schedule, and resources
- Nevertheless, a project manager should not become obsessive about estimation
  - Plans should be iterative and allow adjustments as time passes and more is made certain

Task Set for Project Planning

1) Establish project scope
2) Determine feasibility
3) Analyze risks
4) Define required resources
   a) Determine human resources required
   b) Define reusable software resources
   c) Identify environmental resources
5) Estimate cost and effort
   a) Decompose the problem
   b) Develop two or more estimates using different approaches
   c) Reconcile the estimates
6) Develop a project schedule
   a) Establish a meaningful task set
   b) Define a task network
   c) Use scheduling tools to develop a timeline chart
   d) Define schedule tracking mechanisms

Scope and Feasibility

Software Scope

- Software scope describes
  - The functions and features that are to be delivered to end users
  - The data that are input to and output from the system
  - The "content" that is presented to users as a consequence of using the software
  - The performance, constraints, interfaces, and reliability that bound the system
- Scope can be define using two techniques
  - A narrative description of software scope is developed after communication with all stakeholders
  - A set of use cases is developed by end users

Software Scope (continued)

- After the scope has been identified, two questions are asked
  - Can we build software to meet this scope?
  - Is the project feasible?
- Software engineers too often rush (or are pushed) past these questions
- Later they become mired in a project that is doomed from the onset

Feasibility

- After the scope is resolved, feasibility is addressed
- Software feasibility has four dimensions
  - **Technology** – Is the project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?
  - **Finance** – Is is financially feasible? Can development be completed at a cost that the software organization, its client, or the market can afford?
  - **Time** – Will the project's time-to-market beat the competition?
  - **Resources** – Does the software organization have the resources needed to succeed in doing the project?

Project Resources

Resource Estimation

- Three major categories of software engineering resources
  - Human Resources
  - Reusable software Resources
    - Often neglected during planning but become a paramount concern during the construction phase of the software process
  - Environmental Resources
- Each resource is specified with
  - A <u>description</u> of the resource
  - A statement of <u>availability</u>
  - The <u>time</u> when the resource will be required
  - The <u>duration</u> of time that the resource will be applied

Categories of Resources

People
- Number required
- Skills required
- Geographical location

Development Environment
- Software tools
- Computer hardware
- Network resources

The Project

Reusable Software Components
- Off-the-shelf components
- Full-experience components
- Partial-experience components
- New components

Human Resources

- Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Small projects of a few person-months may only need one individual
- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

Development Environment Resources

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products
- Most software organizations have many projects that require access to the SEE provided by the organization
- Planners must identify the time window required for hardware and software and verify that these resources will be available

Reusable Software Resources

- Off-the-shelf components
  - Components are from a third party or were developed for a previous project
  - Ready to use; fully validated and documented; virtually no risk
- Full-experience components
  - Components are similar to the software that needs to be built
  - Software team has full experience in the application area of these components
  - Modification of components will incur relatively low risk
- Partial-experience components
  - Components are related somehow to the software that needs to be built but will require substantial modification
  - Software team has only limited experience in the application area of these components
  - Modifications that are required have a fair degree of risk
- New components
  - Components must be built from scratch by the software team specifically for the needs of the current project
  - Software team has no practical experience in the application area
  - Software development of components has a high degree of risk

Estimation of Project Cost and Effort

Factors Affecting Project Estimation

- The accuracy of a software project estimate is predicated on
  - The degree to which the planner has properly estimated the size (e.g., KLOC) of the product to be built
  - The ability to translate the size estimate into human effort, calendar time, and money
  - The degree to which the project plan reflects the abilities of the software team
  - The stability of both the product requirements and the environment that supports the software engineering effort

Project Estimation Options

- Options for achieving reliable cost and effort estimates
  1) Delay estimation until late in the project (we should be able to achieve 100% accurate estimates after the project is complete)
  2) Base estimates on similar projects that have already been completed
  3) Use relatively simple decomposition techniques to generate project cost and effort estimates
  4) Use one or more empirical estimation models for software cost and effort estimation
- Option #1 is not practical, but results in good numbers
- Option #2 can work reasonably well, but it also relies on other project influences being roughly equivalent
- Options #3 and #4 can be done in tandem to cross check each other

Project Estimation Approaches

- **Decomposition techniques**
  - These take a "divide and conquer" approach
  - Cost and effort estimation are performed in a stepwise fashion by breaking down a project into major functions and related software engineering activities

- **Empirical estimation models**
  - Offer a potentially valuable estimation approach if the historical data used to seed the estimate is good

Decomposition Techniques

Introduction

- Before an estimate can be made and decomposition techniques applied, the planner must
  - Understand the scope of the software to be built
  - Generate an estimate of the software's size
- Then one of two approaches are used
  - Problem-based estimation
    - Based on either source lines of code or function point estimates
  - Process-based estimation
    - Based on the effort required to accomplish each task

Software Sizing

- Function point sizing
  - Develop estimates of the information domain characteristics (Product Metrics for Software)
- Standard component sizings
  - Estimate the number of occurrences of each standard component
  - Use historical project data to determine the delivered LOC size per standard component
- Change sizing
  - Used when changes are being made to existing software
  - Estimate the number and type of modifications that must be accomplished
  - Types of modifications include reuse, adding code, changing code, and deleting code
  - An effort ratio is then used to estimate each type of change and the size of the change

The results of these estimates are used to compute an optimistic (low), a most likely,and a pessimistic (high) value for software size

Problem-Based Estimation

1) Start with a bounded statement of scope
2) Decompose the software into problem functions that can each be estimated individually
3) Compute an LOC or FP value for each function
4) Derive cost or effort estimates by applying the LOC or FP values to your baseline productivity metrics (e.g., LOC/person-month or FP/person-month)
5) Combine function estimates to produce an overall estimate for the entire project

Problem-Based Estimation (continued)

- In general, the LOC/pm and FP/pm metrics should be computed by project domain
  – Important factors are team size, application area, and complexity
- LOC and FP estimation differ in the level of detail required for decomposition with each value
  – For LOC, decomposition of functions is essential and should go into considerable detail (the more detail, the more accurate the estimate)
  – For FP, decomposition occurs for the five information domain characteristics and the 14 adjustment factors
    - External inputs, external outputs, external inquiries, internal logical files, external interface files

pm = person-month

Problem-Based Estimation (continued)

- For both approaches, the planner uses lessons learned to estimate an optimistic, most likely, and pessimistic size value for each function or count (for each information domain value)
- Then the expected size value S is computed as follows:

$$S = (S_{opt} + 4S_m + S_{pess})/6$$

- Historical LOC or FP data is then compared to S in order to cross-check it

Process-Based Estimation

1) Identify the set of functions that the software needs to perform as obtained from the project scope
2) Identify the series of framework activities that need to be performed for each function
3) Estimate the effort (in person months) that will be required to accomplish each software process activity for each function

Process-Based Estimation (continued)

4) Apply average labor rates (i.e., cost/unit effort) to the effort estimated for each process activity
5) Compute the total cost and effort for each function and each framework activity (See table in Pressman, p. 655)
6) Compare the resulting values to those obtained by way of the LOC and FP estimates
   - If both sets of estimates agree, then your numbers are highly reliable
   - Otherwise, conduct further investigation and analysis concerning the function and activity breakdown

This is the most commonly used of the two estimation techniques (problem and process)

Reconciling Estimates

- The results gathered from the various estimation techniques must be reconciled to produce a single estimate of effort, project duration, and cost
- If widely divergent estimates occur, investigate the following causes
  - The scope of the project is not adequately understood or has been misinterpreted by the planner
  - Productivity data used for problem-based estimation techniques is inappropriate for the application, obsolete (i.e., outdated for the current organization), or has been misapplied
- The planner must determine the cause of divergence and then reconcile the estimates

Empirical Estimation Models

Introduction

- Estimation models for computer software use empirically derived formulas to predict effort as a function of LOC or FP
- Resultant values computed for LOC or FP are entered into an estimation model
- The empirical data for these models are derived from a limited sample of projects
  - Consequently, the models should be calibrated to reflect local software development conditions

COCOMO Model

- Stands for COnstructive COst MOdel
- Introduced by Barry Boehm in 1981 in his book "Software Engineering Economics"
- Became one of the well-known and widely-used estimation models in the industry
- It has evolved into a more comprehensive estimation model called COCOMO II
- COCOMO II is actually a hierarchy of three estimation models

COCOMO-II Model

- **Application composition model** - Used during the early stages of software engineering when the following are important
  - Prototyping of user interfaces
  - Consideration of software and system interaction
  - Assessment of performance
  - Evaluation of technology maturity
- **Early design stage model** – Used once requirements have been stabilized and basic software architecture has been established
- **Post-architecture stage model** – Used during the construction of the software

COCOMO-II Model

- The COCOMO II models require sizing information.
- Three different sizing options are : object points, function points, and lines of source code.
- The **object point** is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application.
- Each object instance is classified into one of three complexity levels (i.e.,simple, medium, or difficult).
- Once complexity is determined, the number of screens, reports, and components are weighted according to following table

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

COCOMO-II Model

- When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

NOP(new object points)**= (object points) * [(100 - %reuse)/100]**

- To derive an estimate of effort based on the computed NOP value, a **"productivity rate"** is derived.

PROD =NOP/person-month

- Using productivity rate, an **estimate of project effort** is computed,

Estimated effort=NOP/PROD

The Software Equation

- The software equation is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project.
- The model has been derived from productivity data collected for over 4000 contemporary software projects.
- Based on these data, derived estimation model is

$$E = \frac{\text{LOC} \times B^{0.333}}{P^3} \times \frac{1}{t^4}$$

- Where,E =effort in person-months or person-years

  t = project duration in months or years

  B = “special skills factor”

  P = “productivity parameter”

Here, (1)an estimate of size (in LOC) and (2) an indication of project duration in calendar months or years. are independent parameter.

The Software Equation

- To simplify the estimation process and use a more common form for their estimation model,Minimum development time is defined as

$$t_{min} = 8.14 \frac{LOC}{P^{0.43}} \text{ in months for } t_{min} > 6 \text{ months}$$

- E =180 Bt^3 in person-months for E>=20 person-months

- Like the COCOMO model,the software equation continues to evolve.

Make/Buy Decision

- It is often more cost effective to acquire rather than develop software.
- Managers have many acquisition options
  - Software may be purchased (or licensed) off the shelf
  - "Full-experience" or "partial-experience" software components may be acquired and integrated to meet specific needs
  - Software may be custom built by an outside contractor to meet the purchaser's specifications
- The make/buy decision can be made based on the following conditions
  - Will the software product be available sooner than internally developed software?
  - Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
  - Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?