Unit III

Software Development

Software Size and Reuse Estimating

”Predicting the size of a software system becomes progressively easier as the project advances”

”At no other time are the estimates so important than at the beginning of a project”

Product Development Life Cycle

- Estimating size and effort will occur many times during the life cycle
- After requirements, after analysis, after design, and so on...

Learning Objectives

- Explain why the sizing of software is an important estimating technique
- Describe how Work Breakdown Structure can be used to estimate software size
- List, explain and describe several models used to estimate size
- Summarize the advantages and disadvantages of the models
- Explain the impact of reused software components upon the size estimate

Problems with Estimating

- Problem is not well understood
- Little or no historical data
- No standards
- Management uses estimates as performance goals
- Developers are optimistic
- Customer demands quick estimates
- etc...

Risks of Estimating

- If incomplete or incorrect estimate:
  - disappointing the customer
  - possibly losing future business
  - too optimistic fixed-price contract result in the contractor losing money and losing face

How to tackle size-related risks

- Produce a WBS, decomposed to the lowest level possible → smaller is easier to estimate
- Review assumptions with all stakeholders
- Research past organizational experiences and historical data
- Stay in close communication with other developers, common language
- Update estimates
- Use many size estimating methods
- Educate staff in estimation methods

Getting Started

- Sizing is the prediction of product deliverables needed to fulfill requirements
- Estimation is the prediction of effort needed to produce the deliverables
- WBS is a description of the work to be performed, broken down into key elements
- WBS is our TOC, a hierarchical list of the work activities required to complete a project
- Choose a method and stick with it
- Compare actual data to planned estimates
- Everything should be counted, any observable physical piece of software

Size measures

- Lines of Code (LOC)
- Function Points
- Feature Points
- Object Points
- Model Blitz
- Wideband Delphi

Lines of Code

- Very difficult to know how many LOC will be produced before they are written or even designed
- LOC measure has become infamous
- Still the most used metric
- Average programmer productivity rate remains despite of new languages
- Functionality and quality are the real concerns, not the LOC

Lines of Code 2

- WBS should be decomposed to the lowest level possible
- Estimating using expert opinions, asking experts who have developed similar systems
- Estimating using Bottom-Up summing, asking developers to estimate the size of each decomposed level
- Ask for an optimistic (200), pessimistic (400) and realistic size (250) estimate → (200+400+(4*250)) / 6 = 266 LOC
- What exactly is a line of code? Standards and rules needed

Lines of Code 3

- Translate the number of LOC to assembly language line in order to make comparisons between programming languages
- e.g.
  convert 50,000 LOC system written in C to Java
  Assembler level for C = 2.5, Java = 6
  50,000 * 2.5 = 125,000 if written in assembler
  125,000 / 6 = 20,833 LOC if written in Java

Advantages of LOC

- Widely used and accepted
- Allows for comparison between diverse development groups
- Directly relates to the end product
- Easily measured upon project completion
- Measure from the developer’s point of view
- Continuous improvement

Disadvantages of LOC

- Difficult to estimate early in the life cycle
- Source instructions variate with language, design, programmer etc.
- No industry standards for counting LOC
- Fixed costs are not included with coding
- Programmers may be rewarded for large LOC counts
- Distinguish between generated code and hand-crafted code
- Can not be used for normalizing if languages are different
- Only existing products and expert opinions can be used to predict a LOC count