CSC 408F/CSC2105F Lecture Notes

These lecture notes are provided for the personal use of students taking CSC 408H/CSC 2105H in the Fall term 2004/2005 at the University of Toronto.

Software Project Estimation

- Both before and during a project we need to be able to estimate the resources (people and time) that the project will consume in the *future*
- These estimates are used to allocate resources and budget and to plan the schedule for the project.
- Estimates are typically based on historical data (previous projects) and on various rules and heuristics to determine the size and scope of a project from very incomplete information.
- An organization *must* expend the effort to record accurate cost and time information about its projects if it is going to do serious estimation for future projects.

Pre Project Estimation

- Estimate resource requirements, cost and schedule for project during project planning.
- These estimates will, in part, determine whether the project
  - Will be undertaken at all
  - Will finish on, under or over schedule.
  - Will finish on, under or over budget.
  - Will have sufficient resources.
- Estimation at this stage involves a lot of guesswork.
  Use historical information (i.e. similar projects) if available
- Estimate conservatively to avoid unpleasant overruns later
  Estimate liberally to sell the project to management
- Serious misestimation is a source of subsequent disasters when a project runs over budget or schedule

Intra Project Estimation

- Estimation resource requirements *during* project.
  - Track original estimates
  - Respond to changes in requirements.
  - Respond to management-imposed budget or schedule changes.
  - Update project plan resource/budget requests.
  - Anticipate disasters

Estimation Techniques

- Estimation Techniques

  1. *Decomposition* - divide project into more manageable parts and estimate each part, then sum
  2. *Empirical Models* - use historical data from similar projects to estimate project based on a few simple parameters like lines of code or number of modules

  Possibly acquire automated tool to assist in developing estimate

- Assume a simple (e.g. linear) relationship between effort and cost.
  For example estimate number of person-months or number of lines of code.
  Multiply by a fixed unit cost ($/month or $/line) to get total cost.

- The better estimation techniques are based on a statistical analysis of previous projects and a soundness argument about the validity of the results

- Need to be very careful about applying estimation techniques to projects very different than the history they are based on.

- Need to standardize measurement techniques to make estimates repeatable.
  e.g. define *line of code* and *person-month*
- Estimate total people costs up to first use of software.
  Maintenance costs are not usually estimated.

Key Estimation Factors

- *Complexity of the Software*
  - Relative to the software developers experience and expertise
  - Historical data shows that complex software (e.g. performance constrained real time systems) requires much more development effort than simpler software
- *Project Size*
  - Communication and coordination costs increase more than linearly as more personnel are added to a project.
  - Interdependencies among software components grows with project size
  - Problem decomposition requires more effort as project size increases
- *Project Structure*
  - degree to which the software can be partitioned into separable, independent components.

- *Uncertainty in Requirements*
  - High uncertainty makes all estimates less precise
  - Historically uncertainty in requirements requires greater effort (e.g. Rapid Prototyping) to achieve a satisfactory system specification
- *Ill-defined Project Scope*
  - Projects with ill-defined scope tend to become open-ended and perpetual
  - Great risk to software developer of never achieving completion
  - Great risk to customer of never achieving useful software
- *Lack of familiarity with application, project language or project hardware*
  - A developer (or customer) who is unfamiliar with the area of application of a project is undertaking a much greater risk of an unsatisfactory conclusion
  - A project is much riskier if the project team has to learn the project implementation language or the hardware during the project.

- Project Team Risks
  - Capabilities of the individual team members
  - Experience with the application area
  - Experience with the implementation language(s)
  - Experience with the target environment and hardware
  - Experience with the development environment and hardware
- Requirements for integration with existing systems.
  - Availability of interface documentation
  - Stability of the interfaces
- Availability of tools, documentation and standards.

Factors Determining Project Magnitude

- Upper and lower bounds on quantitative values that affect the scope of the system (e.g. maximum number of users, maximum database size, etc.)
- Degree to which the project involves research (exploring the unknown) as opposed to using well understood technology
- Performance requirements and constraints
- Functionality required in the software
- Environment in which the software will be used
  Constraints imposed by the environment
- Reliability and robustness requirements
- Intended users of the software (e.g. novice vs. expert)

Why Estimates Fail

- Pre-project Estimates
  - Frequent changes in requirements
  - Activities missing from estimate.
  - Users lack of understanding of the requirements.
  - Insufficient analysis during estimation
  - Inadequate or inappropriate estimation method.
- Intra-project Estimates
  - Poor coordination during development.
  - Team productivity lower than expected.
  - Problems breed further problems. Small errors can have large consequences.
  - 90% complete syndrome, real status of project never made clear.
- Use of a new technology (e.g. extreme programming) means we have no basis (history or statistics) to estimate the next project.

Pre-Project Estimation Tips

- Determine *scope* of the project.

  Look for potential problems, unknowns, risks.

- Decompose the requirements into reasonable parts.

  Try to get user sign off to freeze requirements.

- Decide on development processes (tools and methods)

- Investigate reuse opportunities.

- Estimate more than one way and compare as reality check.

- Establish change control procedures so estimates can track requirements changes.

Estimation Methods

- Comparison with historical projects
- Delphi - iterative consensus of experts
- Weighed average
- Intuition and experience
- Estimation models

Models for Estimating Effort

- Estimate the number of person months for a project as a function of
  1. cost drivers, i.e. factors influencing productivity
  2. lines of code (KLOC), usually source.
  3. syntactic units, e.g. operators and operands
  4. variables occurring in the program
  5. function points (FP), i.e. data structures
- Issues
  - How many hours in a person month?
  - Differences due to programming languages?
  - Are these equations based on circumstances sufficiently similar to the project we are estimating?
- Calculate average productivity and costs per KLOC or FP based on your organizations history.

Decomposition Estimation from KLOC or FP

- Do a top down recursive sub-division of the system into smaller subcomponents
- Decomposition depends on designers experience and may use previous projects for guidance
- Stop recurring as soon as a reasonable KLOC or FP estimate can be obtained for each subcomponent

  FP estimation usually requires less detailed decomposition than KLOC estimation
- Derive *expected value* for KLOC or FP, based on (optimistic, realistic and pessimistic) estimates

  $$Expected-Value = (optimistic + 4 \times realistic + pessimistic)/6$$
- Sum upward to derive estimated KLOC or FP for entire software project

Decomposition Estimation (cont'd.)

- Once KLOC or FP estimate has been derived, use it and historical data from *similar* project to estimate project attributes
- Total Effort - KLOC or FP divided by historical programmer productivity
  Beware of interaction effects.
  A 100 person-month project *can not* be done in 2 weeks by 200 programmers (or probably even in 2 months by 50 programmers)
- Total Cost - KLOC or FP multiplied by historical cost of developing KLOC or FP in *similar* projects
- **Estimation is not rocket science!**
  An estimation error of $\pm$ 10 .. 20 % is considered good
- Example: estimate new project as medium complexity 500 KLOC
  Find previous projects of medium complexity in the 400KLOC .. 600KLOC size range. Use time and cost from those projects to estimate new project.

Function Point Calculation

- Assign a weighting factor (Simple, Average, Complex) to each of the five characteristics and compute $CountTotal$ as a weighted sum.
- Determine the *complexity adjustment factors* F_i
  Each F_i is rated from 0 (no influence) to 5 (essential)
- Compute functions points as
  $FP = CountTotal \times [\, 0.65 + 0.01 \times \sum_{i=1}^{i=14} F_i \,]$
- Advantages of Function Points
  - Programming language independent
  - Easier to compute early in a project
- Disadvantages of Function Points
  - Subjective complexity factors
  - Function Points have no direct physical meaning
  - Function Points use *external* characteristics of the system.
    They don't evaluate *internal* complexity well.

Definitions for Estimation

E Effort in person-months.

$KLOC$ Thousands of lines of source code (to calculate E)

Sometimes thousands of delivered source code instructions.

FP Function points (used to calculate $KLOC$)

A measure of the complexity of a piece of software

Low FP implies simple, high FP implies complex.

ln/FP Number of instructions per function point.

Approximately 70 for Ada, 100 for Cobol, 65 for PL/I, 91 for Pascal,

128 for C, 53 for C++ & Java, 320 for assembler.

OP Object point, used to calculate E for 4th generation languages

Estimation Example
Use Unadjusted FP to Calculate $KLOC$

Characteristic	Example	Weight	Number	Unadj. FP
Number of inputs	record read from file	4	14	56
Number of outputs	addition or change to a file	5	20	100
Number of inquiries	1 inquiry case/transaction	4	30	120
Number of internal files		10	8	80
Number of interfaces	connection to another system	7	4	28

$$
\begin{aligned}
KLOC &= ln/FP * FP \\
&= (\text{instructions per FP} * \text{unadjusted FP})/1000 \\
&= (128\ (\text{ for C})\ * 384) \\
&= 49.15\ \text{thousand lines of C code}
\end{aligned}
$$

Refining the FP Estimate

- Map simple counts used in previous slide to more accurate count based on the complexity of the item.
- Example for input types

Number of	Number of Data Elements		
File types	1 – 4	5 – 15	> 15
0 or 1	simple	simple	average
2 – 3	simple	average	complex
> 3	average	complex	complex

- Refinements for other items are similar.

Use Refined Unadjusted FP to calculate $KLOC$

	simple		average		complex		function
Characteristic	Weight	No.	Weight	No.	Weight	No.	points
Number of inputs	3	4	4	5	6	5	62
Number of outputs	4	10	5	5	7	5	100
Number of inquiries	3	10	4	10	6	10	130
Number of files	7	2	10	4	15	2	84
Number of interfaces	5		7	4	10		28
						Total	404

$$
\begin{aligned}
KLOC &= ln/FP * FP \\
&= (\text{instructions per FP} * \text{unadjusted FP})/1000 \\
&= (128\ (\text{ for C}) * 404) \\
&= 51.71 \text{ thousand lines of C code}
\end{aligned}
$$

Complexity Adjustment Factors

	Application Characteristic	Example
1	Data Communication	3
2	Distributed functions	3
3	Performance	3
4	Heavily used configuration	2
5	Transaction rate	2
6	Online data entry	4
7	End-user efficiency	3
8	Online update	3
9	Complex processing	2
10	Reusability	4
11	Instalation ease	2
12	Operational ease	2
13	Multiple sites	5
14	Facilitate change	3
	Total (DI)	41

Use Adjusted FP to Calculate $KLOC$

$$
\begin{aligned}
AFP &= \text{adjusted function point} \\
FP &= \text{unadjusted function point} \\
TCF &= \text{technical complexity factor} \\
DI &= \text{degree of influence}
\end{aligned}
$$

$$
\begin{aligned}
AFP &= FP * TCF \\
&= FP * (0.65 + 0.01 * DI) \\
&= 404 * (0.65 + 0.01 * 41) \\
&= 404 * 1.06 \\
&= 428.24 \\
KLOC &= 428.24 * 128 \\
&= 54.81 \text{ thousand lines of C code}
\end{aligned}
$$

Boehm's COCOMO Estimation Model

- COCOMO[a] is a widely used empirical estimation technique.
  It was based on the software development processes that were current in the early 1980s.
- COCOMO 2.0 is an updated version of the original model that is based on more modern software development processes.
- COCOMO 2.0 contains three estimation models
  - *Application Composition Model* Used for prototyping, assumes heavy software reuse, prototyping tools.
  - *Early Design Model* Used during the architectural design stage (requirements .. design)
  - *Post-Architecture Model* used during development and testing.
    An extension of the original COCOMO model.

[a] B. Boehm, Software Engineering Economics, Prentice-Hall, 1981

- In COCOMO 2.0 the estimation basis changes as project proceeds.

Stage	Estimation Basis
Application Composition	Object Points
	(number of screens, reports, 3GL modules)
Early Design	unadjusted function points + cost drivers
Post-Architecture	lines of code + cost drivers

COCOMO 2.0 Basic Estimation

- All of the COCOMO estimation models follow the same general pattern.
  - Compute the appropriate estimation basis using the rules and weights appropriate to the model.
  - Use formulas for the model to estimate the required effort E.
- The Application Composition Model
  - Compute number of Object Points (OP) for the whole system[a]
  - Estimate a percentage of software reuse, and compute the number of *New* Object Points (NOP) required.
  - Determine a productivity rate $PROD$ which is the number of new object points per month that the project team can produce.
  - Estimate $E = NOP/PROD$

[a] See van Vliet Section 7.3.6

- Early Design Model
  - Calculate unadjusted function points (UFP)for the system.
  - Convert the UFP to thousands of lines of code ($KSLOC$) using an environment specific factor.
  - Use a set of cost drivers to modify the initial estimate
    1. product reliability and complexity
    2. required reuse
    3. platform difficulty
    4. experience of personnel
    5. capability of personnel
    6. available facilities
    7. schedule
  - Estimate E as $E = KSLOC\ x\ \prod_i\ cost\_driver_i$

- Post Architecture Model
  - The estimation formula for the model is

    $$E = a\ x\ A\ x\ KSLOC^{b}\ x\ \prod_i\ cost\_driver_i$$

  - The factor a is determined by the type of project ($a\ =\ 2.4\ or\ 3.0\ or\ 3.6$)
  - The factor A accounts for software reuse.
  - The exponent b is derived from a sum of five factors using the formula

    $$b = 1.01\ +\ 0.01\ x\ \sum_i\ W_i$$

    Where each of the factors W_i ranges from 5 (low) to 0 (high). The factors are

    1. Precedentedness (novelty of the project to the organization)
    2. Development flexibility (external constraints/interfaces)
    3. Architecture risk/resolution (risks resolved, interfaces specified)
    4. Team cohesion (all stake holders, consistent objectives)
    5. Process maturity (e.g. CMM rating)
  - The values of the cost drivers are estimated on a 7-point scale are are assigned values centered on 1.0 (neutral). See van Vliet Table 7.14.

Post Architecture Code Drivers

Product factors

- Reliability required
- Database size
- Product complexity
- Required reusability
- Documentation needs

Project Factors

- Use of software tools
- Multi-site development
- Required development schedule

Platform factors

- Execution time constraints
- Main storage constraints
- Platform volatility

Personnel factors

- Analyst capability
- Programmer capability
- Application experience
- Platform experience
- Language and tool experience
- Personnel continuity

The Software Reuse

- The factor A is used to account for software reuse and the effort required to adapt the reused software to the project.
- A is calculated as:

$$\begin{aligned} A &= 0.4\,x\,(fraction\,of\,the\,system\,requiring\,redesign) \\ &+ 0.3\,x\,(fraction\,of\,the\,system\,requiring\,recoding) \\ &+ 0.3\,x\,(fraction\,of\,the\,system\,requiring\,reintegration) \\ &+ SU + AA \end{aligned}$$

- SU is the *software understanding* increment. SU ranges from 10% for well-organized and documented modules to 50% for high-coupling, low-cohesion, poorly documented modules.
- AA accounts for the effort required to assess and assimilate the reuse software. AA ranges from 0% for no effort required, to 8% for extensive testing, evaluating and documenting.

Using Timing Estimates

- The estimating models calculate a value for E, total project time in months.
- We still need to plan how to use those months.
- Various estimating models can also be used to estimate total project time T.
  For example for COCOMO 2.0 $T = 3.0\, x\, E^{0.33+0.2x(b-1.01)}$
- The calculated value of T is the nominal time.
  There is *limited* flexibility to do a project in time less than T by increasing the resources available to the project.
- As the size of a project team grows, the productivity of the individual team members decreases.
  - Communication and consultation time increases.
  - New team members are less productive initially.
  - New team members require time from existing team members to learn the project.

- *Brooks's Law:* Adding people to a late project only makes it later.
- A study by Conte *et.al.* showed that the average productivity of individual team members varied as $\sqrt{team\ size}$

  Increasing the size of a team by a factor of 4 increases the amount of code produced per month by a factor of 2.

  Cost increases by a factor of 4 so each line of code costs twice as much to produce.
- Boehm conjectures that trying to do a project in less than $0.75\ x\ T$ is not feasible.
- Trying the develop a system *to fast*, ignoring project estimates is a recipe for disaster.

Course Project material HOME =

http://www.cs.toronto.edu/∼yijun/csc408h/

- Project Deliverables and Marking Schemes

  HOME/handouts/deliverables-PhaseA.txt

  HOME/handouts/deliverables-PhaseB.txt

- Project Help pages

  HOME/handouts/WebServices-HOWTO.html

- RSS syndication: [a]

  HOME/announcements.xml

  HOME/handouts/HOWTO.xml

[a] Live Bookmarks (firefox): http://www.mozilla.org/products/firefox