Chapter 2 – Evolution of Software Economics

2.1 Software Economics

- Five fundamental parameters that can be abstracted from software costing models:
  - Size
  - Process
  - Personnel
  - Environment
  - Required Quality
- Overviewed in Chapter 2
- Much more detail in Chapter 3.

Software Economics – Parameters (1 of 4)

- Size: Usually measured in SLOC or number of Function Points required to realize the desired capabilities.
  - Function Points – a better metric earlier in project
  - LOC (SLOC, KLOC…) a better metric later in project
  - These are not new metrics for measuring size, effort, personnel needs,…

- Process – used to guide all activities.
  - Workers (roles), artifacts, activities…
  - Support heading toward target and eliminate non-essential / less important activities
  - Process **critical** in determining software economics
    - Component-based development; application domain…iterative approach, use-case driven…
  - Movement toward '**lean**' … everything!

Software Economics – Parameters (2 of 4)

- <u>Personnel</u> – capabilities of the personnel in general and in the application domain in particular
  - Motherhood: get the right people; good people; Can't always do this.
  - Much specialization nowadays. Some are terribly expensive.
  - Emphasize 'team' and team responsibilities…Ability to work in a team;
    - Several newer light-weight methodologies are totally built around a team or very small group of individuals…

Software Economics – Parameters (3 of 4)

- Environment – the tools / techniques / automated procedures used to support the development effort.
  - Integrated tools; automated tools for modeling, testing, configuration, managing change, defect tracking, etc…

- Required Quality – the functionality provided; performance, reliability, maintainability, scalability, portability, user interface utility; usability…

Software Economics – Parameters (4 of 4)

Effort = (personnel)(environment)(quality)(size $_{\text{Process}}$)

(Note: effort is exponentially related to size….)

What this means is that a 10,000 line application will cost less per line than a 100,000 line application.

- These figures – surprising to the uninformed – are true.
- Fred Brooks – Mythical Man Month – cites over and over that the additional communications incurred when adding individuals to a project is very significant.
  - Tend to have more reviews, meetings, training, biases, getting people up to speed, personal issues…
- Let's look at some of the trends:

Notice the Process Trends….for three generations of software economics

- Conventional development (60s and 70s)
  - Application – custom; Size – 100% custom
  - Process – ad hoc …(discuss) – laissez faire;
  - 70s - SDLC; customization of process to domain / mission, structured analysis, structured design, code.
- Transition (80s and 90s)
  - Environmental/tools – some off the shelf.
    - Tools: separate, that is, often not integrated esp. in 70s…
  - Size: 30% component-based; 70% custom
  - ☐ Process: repeatable
- Modern Practices (2000 and later)
  - Environment/tools: off-the-shelf; integrated
  - Size: 70% component-based; 30% custom
  - Process: managed; measured (refer to CMM)

Notice Performance Trends….for three generations of software economics

- Conventional: Predictably bad: (60s/70s)
  - usually always over budget and schedule; missed requirements
    - All custom components; symbolic languages (assembler); some third generation languages (COBOL, Fortran, PL/1)
    - Performance, quality almost always less than great.
- Transition: Unpredictable (80s/90s)
  - Infrequently on budget or on schedule
  - Enter software engineering; 'repeatable process;' **project management**
  - Some commercial products available – databases, networking, GUIs; But with huge growth in complexity, (especially to distributed systems) existing languages and technologies not enough for desired business performance
- Modern Practices: Predictable (>2000s)
  - Usually on budget; on schedule. Managed, measured **process management**. Integrated environments; 70% off-the-shelf components. Component-based applications RAD; iterative development; stakeholder emphasis.

All Advances Interrelated…

- Improved ‘process’ requires ‘improved tools’ (environmental support…)
- Better ‘economies of scale’ because
  - ☐ Applications live for years;
  - Similarly-developed applications – common.
  - First efforts in common <u>architectures, processes</u>, <u>iterative processes</u>, etc., all have <u>**initial**</u> <u>high overhead;</u>
  - But follow-on efforts result in economies of scale…and much better ROI. (See p. 25)
  - “All simple systems have been developed!”

2.2 "Pragmatic" Software Cost Estimation

- Little available on estimating cost for projects using iterative development.
  - Difficult to hold all the controls constant
    - Application domain; project size; criticality; etc. Very 'artsy.'
    - Metrics (SLOC, function points, etc.) <u>NOT consistently applied</u> EVEN in the same application domain!
    - Definitions of SLOC and function points are not even <u>consistent</u>!
  - Much of this is due to the nature of development. There is no <u>magic date</u> when design is 'done;' or magic date when testing 'begins' …
  - Consider some of the issues:

Three Issues in Software Cost Estimation:

- 1. Which cost estimation model should be used?
- 2. Should software size be measured using SLOC or Function Points? (there are others too…)
- 3. What are the determinants of a good estimate? (How do we know our estimate is good??)

So very much is dependent upon estimates!!!!

Cost Estimation Models

- Many available.
- Many organization-specific models too based on their own histories, experiences…
  - Oftentimes, these are super if 'other' parameters held constant, such as process, tools, etc. etc.
- COCOMO, developed by Barry Boehm, is the most popular cost estimation model.
- Two primary approaches:
  - Source lines of code (SLOC) and
  - Function Points (FP)
- Let's look at this – overview.

Source Lines of Code (SLOC)

- Many feel comfortable with 'notion' of LOC
- SLOC has great value – especially where applications are <u>custom-built</u>.
  - Easy to measure & instrument – have tools.
  - <u>Nice when we have a history</u> of development with applications and their existing lines of code and associated costs.

- Today – with use of components, source-code generation tools, and objects have rendered SLOC somewhat ambiguous.
  - We often <u>don't know</u> the SLOC – but do we care? How do we factor this in? ☐

Source Lines of Code (SLOC)

- Generally more useful and precise basis than FPs
- Appendix D – an extensive case study.
  - Addresses how to count SLOC where we have reuse, different languages, etc.
  - Read this appendix (five pages)

- We will address LOC in much more detail later.

- Appendix provides <u>hint at the complexity</u> of using LOC for software sizing particularly with the new technologies using <u>automatic code generation</u>, <u>components</u>, <u>development of new code</u>, and more.

Function Points

- Use of Function Points - many proponents.
  - International Function Point User’s Group – 1984 – “is the dominant software measurement association in the industry.”
  - Check out their web site (www.IFPUG.com ??)
  - Tremendous amounts of information / references
  - Attempts to create industry standards….

- ☐ Major advantage: Measuring with function points is <u>independent of the technology</u> (programming language, tools …) used and is thus better for comparisons among projects. ☐

Function Points

- Function Points measure numbers of
  - external user inputs,
  - external outputs,
  - internal data groups,
  - external data interfaces,
  - external inquiries, etc.
- ☐ Major disadvantage: Difficult to measure these things.
  - Definitions are primitive and inconsistent
  - Metrics difficult to assess especially since normally done <u>earlier</u> in the development effort using more abstractions.
- Yet, no project will be started without estimates!!!!

But:

- Cost estimation is a real necessity!!! Necessary to 'fund' project!
- All projects require estimation in the beginning (inception) and adjustments…
  - These must stabilize; They are rechecked…
  - Must be <u>**reusable**</u> for additional cycles
  - Can <u>create organization's own methods</u> of measurement on how to 'count' these metrics...
- No project is arbitrarily started without cost / schedule / budget / manpower / resource <u>estimates</u> (among other things)
- <u>☐ SO</u> critical to budgets, resource allocation, and to a host of stakeholders

So, How Good are the Models?

- COCOMO is said to be ‘within 20%’ of actual costs ’70% of the time.’ (COCOMO has been revised over the years…)
- Cost estimating is still **disconcerting** when one realizes that there are already a plethora of missed dates, poor deliverables, and significant cost overruns that characterize traditional development.
- Yet, all non-trivial software development efforts require costing; It is a basic management activity.
- RFPs on contracts force contractors to estimate the project costs for their survival.
- So, let’s look at top down and bottom up estimating.

Top Down versus Bottom Up Substantiating the Cost…

- Most estimators perform bottom up costing - **substantiating** a target cost - rather than approaching it a top down, which would yield a ‘should cost.’
- Many project managers create a ‘target cost’ and then play with parameters and sizing until the target cost can be justified…
  - Work backwards!
  - Attempts to win proposals, convince people, …
- Any approach should force the project manager to assess risk and discuss things with stakeholders…

Top Down versus Bottom Up

- Bottom up … substantiating? Good?
  - <u>If well done</u>, it requires considerable analysis and expertise <u>based on much experience and knowledge</u>; Development of similar systems a great help; similar technologies…
  - <u>If not well done</u>, causes team members to go <u>***crazy***</u>! (This is not uncommon)
- Independent cost estimators (consultants…) not reliable.

Author suggests:

- Likely best cost estimate is undertaken by an **experienced project manager**, software architect, developers, and test managers – and this process can be quite iterative!
- **Previous experience is essential**. Risks identifiable, assessed, and factored in.
- When created, the **team must live with** the cost/schedule **estimate**.
- More later in course. But for now (Heuristics from our text:)

A Good Project Estimate:

- □Is conceived and supported by the project manager, architecture team, development team, and test team accountable for performing the work.
- □Is accepted by all stakeholders as ambitious but doable
- Is based on a well-defined software cost model with a credible basis
- □ Is based on a database of relevant project experience that includes similar processes, similar technologies, similar environments, similar quality requirements, and similar people, and
- □ Is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

A Good Project Estimate

- Quoting: "An 'ideal estimate' would be derived from a mature cost model with an experience base that reflects multiple similar projects done by the same team with the same mature processes and tools.

- "Although this situation rarely exists when a project team embarks on a new project, good estimates can be achieved in a straightforward manner in later life-cycle phases of a mature project using a mature process."