Software Estimation Further Techniques

CSSE3012

“You cannot control what you cannot measure”

(Famous management adage)

How can you estimate the size of something if you've never measured the size of anything?

Estimation

- Get initial feel for project cost
- Determine if project is still feasible
- Cull and re-prioritise features based on cost
- Estimates are very coarse-grained
  - this is a first pass
  - we inevitably revise these as more info arrives
- Should be a range rather than a single value
- Should be based on real data
- Done as a team

Estimation Guidelines

- Use a structured and defined process
- Apply throughout the project
- Collect and use historical data
- Adjust to suit new projects
- Apply statistical analysis where possible
- Look for automation opportunities
- Apply multiple techniques and compare results

Software Size Measures

- We need to be able to quantify software size in an objective manner
  - enables comparison
- Types of size measures
  - syntactic
    - e.g. Lines of Code (LOC)
  - semantic
    - e.g. Function Points

Lines of Code

- Widely used, even though there are obvious limitations
  - need a counting standard
  - language dependent
  - hard to visualise early in a project
  - hard for clients to understand
  - does not account for complexity or environmental factors

Function Points

- Developed by Albrecht (1979) at IBM
  - for data processing domain
  - subsequently refined and standardised
- Based on a user view of a system
  - external inputs – e.g. create, update, delete
  - external outputs – e.g. generate report
  - external enquiries – e.g. read data
  - internal logical files – e.g. database table
  - external interface files – e.g. database table shared between applications

Function Points – Calculating

- Basic Function Points =

  $4 \times EI + 5 \times EO + 4 \times EQ + 10 \times ILF + 7 \times EIF$

- Each elementary function type has its own relative weighting for complexity
  - low -25%
  - average
  - high +50%

Function Points – VAF

- Value Adjustment Factor (VAF)
  - determined by 14 general system characteristics
    - e.g. operational ease
    - transaction volume
    - distributed data processing
- VAF ranges from 0.65 to 1.35

Difficulties with Function Points

- Counting function points is subjective
  - even with standards in place
- Counting cannot be automated
  - even for finished systems (c.f. LOC)
- Factors are dated and do not account for newer types of software systems
  - e.g. real-time, GUI, web
- Many extensions to the original function points attempt to address new types of system

Sample Productivity (FP/hr)

	Mainframe (Cobol, 4GL)	Midrange (C++, Java)	PC (MS-Access)
P25	12.2	9.6	3.8
Median	21.0	14.0	6.3
P75	35.6	21.4	11.7

Based on International Software Benchmarking Standards Group (ISBSG) data, release 6 (April 2000)

Function Points

- Function point count modified by complexity of the project
- Can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC × number of function points
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL

Parametric Models

- Parametric cost models attempt to capture the relationship between size and effort
- Some sample models for LOC sizing
  - $E = 5.5 + 0.72 \times KLOC^{1.16}$ – Bailey-Basili
  - $E = 2.4 \times KLOC^{1.05}$ – COCOMO Basic
- Some sample models for FP sizing
  - $E = -12.39 + 0.0545 \times FP$ – Albrecht-Gaffney
  - $E = 60.62 + 7.728 \times 10^{-8} \times FP^3$ – Kemerer

Simple Estimation Technique

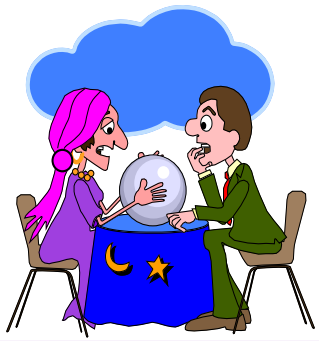
- Each entity class in a model will need
  - 1 user interface, 1 data access, and ½ a helper class
  - assumes a 3-tier architecture
- Estimation Formula
  - $\boldsymbol{E = 3.5 \times Model\ Classes \div Productivity}$
- Assumes you know developer productivity
  - *must* keep logs
  - typical industry ranges
    - 2 – 20 classes/month
      - average 4 – 8

Internet Banking Example

- 9 entity classes in business logic layer
- 3 developers at 6 classes / month
  - $(3.5 \times 9) \div (3 \times 6) = 31.5 \div 18 =$ 1.75 months
- 2 developers at 6 classes / month
  - $(3.5 \times 9) \div (2 \times 6) = 31.5 \div 12 = 2.625 \approx$ 2.66 months
- 3 developers at 5 classes / month
  - $(3.5 \times 9) \div (3 \times 5) = 31.5 \div 15 = 2.1 \approx$ 2.25 months
- 3 developers at 4 classes / month
  - $(3.5 \times 9) \div (3 \times 5) = 31.5 \div 12 \approx$ 2.66 months

COCOMO 2

- Algorithmic model for cost estimation
  - developed by Barry Boehm at USC
- Based on data from a large number of projects
  - many were large defence projects
- 3 sub-models
  - account for different development approaches
  - reflect information known at different stages of development



COCOMO 2 – Target Sectors

- Application Composition
  - assumes component reuse, scripting, or DB programming
  - uses application points for size estimates
- Application Development / System Integration / Infrastructure Development
  - normal development regime

COCOMO 2

Application Development, System Integration & Infrastructure Development

- Initial Prototyping
  - uses application composition model
- Early Design
  - applicable after requirements are understood
  - may use function points translated to LOC
- Post-Architecture
  - applicable after architectural design
  - uses LOC for estimates
  - takes into account component reuse and code generation
  - considers complexity factors

Object / Application Points

- Alternative function-related measure to FP
  - suited to DB programming or scripting languages
    - 4GLs
  - easier to estimate than FP
- Object points are *NOT* the same as objects in an OO sense
  - COCOMO 2 uses the term Application Points to avoid confusion
- Can be estimated early in the development process

Application Point Estimation

- Number of separate screens that are displayed
  - simple screens are 1 object point
  - average screens are 2
  - complex screens are 3
- Number of reports that are produced
  - simple reports are 2
  - average reports are 5
  - complex/difficult reports are 8
- Number of 3GL modules that must be developed to supplement the 4GL code
  - 10 object points per module (e.g. a class in Java)

Application Composition Model

- Early “order of magnitude” estimate
  - may be more accurate for 4GL intensive projects
- Based on standard estimates of developer productivity in application points / month
  - considers developer experience and CASE tool use
  - doesn’t consider system complexity or developer variability

Estimation Formula

- $$PM = \left(NAP \times \left(1 - \left(\%reuse \div 100\right)\right)\right) \div PROD$$
  - PM – effort in person-months
  - NAP – number of application points
  - PROD – productivity

Developer Experience and Capability	V Low	Low	Nominal	High	V High
CASE Maturity and Capability	V Low	Low	Nominal	High	V High
PROD (NAP / Month)	4	7	13	25	50

ATM Example

- Screens
  - Enter PIN, Wrong PIN, Keep Card, Terminal Unavailable, Select Transaction, Select Account, Enter Amount, Display Balance, Select Transfer Account, Collect Cash, Remove Card
  - all fairly simple: 11 application points
- 3GL Modules
  - Transaction, Deposit, Transfer, Withdrawal, Balance, ReceiptPrinter, CashDispenser, Money, Currency, Account, Savings, Chequing, Investment, CreditCard, PIN, Customer, ATM, BankSystem, CardReader
  - 19 classes = 190 application points
- PM = (NAP × (1 – (%reuse ÷ 100))) ÷ PROD
- PM = (201 × 1) ÷ 4 = 50.25 PM
- PM = (201 × 1) ÷ 13 = 15.46 ≈ 15.5 PM
- PM = (201 × 1) ÷ 50 = 4.02 ≈ 4 PM

ATM Example with Reuse

- What if we can reuse the Account, Transaction and Currency classes?
  - 10 classes
- PM = (NAP × (1 – (%reuse ÷ 100))) ÷ PROD
- PM = (201 × (1 – (10 ÷ 19))) ÷ 4
  = (201 × 0.4737) ÷ 4
  = 23.8 ≈ <u>24 PM</u>
- PM = (201 × (1 – (10 ÷ 19))) ÷ 13 = 7.33 ≈ <u>7.33 PM</u>
- PM = (201 × (1 – (10 ÷ 19))) ÷ 50 = 1.9 ≈ <u>2 PM</u>

Early Design Model

- Requirements have been agreed upon
- $\boldsymbol{PM = A \times Size^{B} \times M}$
  - *A* = 2.94 in initial calibration
  - *Size* in KLOC
  - *B* varies from 1.1 to 1.24
    - depending on novelty of project, development flexibility, risk management, and process maturity
  - *M* is a complexity multiplier

Multipliers

- Multipliers reflect complexity of project for this team
  - RCPX – product reliability and complexity
  - RUSE – reuse required
  - PDIF – platform difficulty
  - PREX – personnel experience
  - PERS – personnel capability
  - SCED – required schedule
  - FCIL – team support facilities

Effect of Complexity Multipliers

Multiplier	Complex Project	Simple Project
RCPX (reliability & complexity)	1.5 (v. high)	0.75 (low)
RUSE (reuse)	1.0 (none)	0.9 (good amount)
PDIF (platform difficulty)	1.25 (complex)	1.0 (common)
PREX (personnel experience)	1.25 (v. little)	0.9 (experienced)
PERS (personnel capability)	1.5 (v. inexperienced)	0.75 (v. capable)
SCED (schedule difficulties)	1.25 (tight)	1.0 (none)
FCIL (team support facilities)	1.25 (minimal)	0.9 (good toolset)

Effect of Complexity Multipliers

- Initial estimate
  - estimating that an ATM has about 4000 LOC
  - $2.94 \times 4^{1.17} = 14.9 \approx$ 15 PM
- Complex project
  - $14.9 \times 1.5 \times 1.0 \times 1.25 \times 1.25 \times 1.5 \times 1.25 \times 1.25 = 81.85 \approx$ 82 PM
- Simple project
  - $14.9 \times 0.75 \times 0.9 \times 1.0 \times 0.9 \times 0.75 \times 1.0 \times 0.8 = 5.43 \approx$ 5.5 PM

Post-Architecture Model

- Same formula as early design model
  - $\boldsymbol{PM = A \times Size^{B} \times M}$
    - with 17 complexity multipliers (not 7)
- Estimate of size is adjusted to take into account
  - requirements volatility
  - extent of possible reuse
    - reuse is non-linear and has associated costs so this is not a simple reduction in LOC
  - review of LOC estimate
  - risk
  - team and process maturity

The Exponent Term – B

- Depends on 5 scale factors
  - 1.01 + (sum of scale factors ÷ 100)
- Precedentedness
  - experience with this type of project
- Development Flexibility
  - ability to adapt and tailor the development process
- Architecture / Risk Resolution
  - extent of risk analysis
- Team Cohesion
  - how well the team works together
- Process Maturity
  - formal measure; can simply be (5 – CMMI Level)

Exponent Example

- Newly formed team, rated at CMMI level 2, takes on a project in a new domain. Client has fixed requirements, has not defined the process to be used, and has not allowed time for risk analysis.
  - Precedentedness – new project (4)
  - Development flexibility – fixed reqs - very low (5)
  - Architecture/risk resolution – no risk analysis - very low (5)
  - Team cohesion – new team - low (4)
  - Process maturity – some control - nominal (3)
- Exponent is: 1.01 + (4+5+5+4+3) ÷ 100 = <u>1.22</u>

Project Duration and Staffing

- Minimal calendar time formula
  - $\boldsymbol{TDEV = 3 \times PM^{0.33+0.2 \times (B-1.01)}}$
    - PM is the effort computation
    - B is the exponent calculated previously
      - B is 1 for the early prototyping model
- Assumes an unlimited number of people are available
  - too few people will increase duration
- TDEV = $3 \times 15^{(0.33+0.2 \times (1.22-1.01))} = 3 \times 15^{0.372} \approx$ 8.25 months
- TDEV = $3 \times 82^{0.372} \approx$ 15.5 months
- TDEV = $3 \times 5.5^{0.372} \approx$ 5.66 months

Staffing Requirements

- Staff numbers can't be computed by dividing development time by the schedule
- Number of people required depends on activities being performed
- The more people who work on the project, the more total effort is usually required
- Rapid build-up of people often correlates with schedule slippage

"Adding people to a late project makes it later."

– Fred Brooks. *The Mythical Man-Month*. Addison Wesley, 1982.

Reading

- Sommerville, Chapter 23
- Wiegers, Sections 19.1, 19.2
- Larman, Chapter 40

Next Steps

- Lecture next week
  - Software process improvement
  - Personal software process
  - Final exam hints
- Finalize use case modelling project
  - Due: 30 May, 4:00 pm AEST