Lecture 22: Software Measurement

- Basics of software measurement
  - metrics
  - predictive models
  - validity
- Some example models
  - COCOMO (for effort and time estimation)
  - Function Points (for estimating software size)
  - Reliability Models
  - Cyclomatic Complexity

Basics of Software Measurement

Source: Adapted from Pfleeger 1998, p465-470

- Definitions
  - Metric - a quantifiable characteristic of software
  - Measurement - the process of mapping from real world attributes to a mathematical representation
  - Model - a mathematical relationship between metrics
    - e.g. between quality factors and available metrics
  - Validity - Does the metric accurately measure what it purports to measure
  - Prediction system - a set of metrics and a model that can be used to predict some attribute of a future entity.
    - Deterministic predictions give the same result for the same inputs
    - Stochastic predictions provide a window of error around the actual value
- Difficulties with software measurement
  - We are not measuring repeatable, objective phenomena
  - Software development is so complex that all models are weak approximations
    - models that work for one project or team don't work for others
    - local contingency factors may be more important than the metrics in the model

Example model: COCOMO

Source: Adapted from van Vliet, 1999, section 7.3.2

- COnstructive COst Model (COCOMO)
  - Used to predict cost of a project from a measure of size (lines of code)
  - Basic model is: $E = aL^b$ (E = effort; a, b = project specific factors; L = lines of code)
- Modeling process
  - Establish type of project (organic, semidetached, embedded)
    - this gives sets of values for a and b
  - Identify the component modules, and estimate L for each module
  - Adjust L according to how much is reused
    - COCOMO has a model for adjusting according to how much design, code and integration data is reused
  - Compute effort for each module using $E = aL^b$
  - Adjust E according to difficulty of the project
    - COCOMO identifies 15 effort multipliers to take into account
    - Product attributes: eg required reliability, complexity, database size
    - Computer attributes: eg execution time constraints, storage constraints, etc.
    - Personnel attributes: eg capability & experience of analysts and programmers,
    - Project attributes: eg use of CASE tools, programming language, schedule
  - Compute time using $T = cE^d$
    - c and d provided for different project types like a and b were

Example model: Function Points

Source: Adapted from van Vliet, 1999, section 7.3.5

- Function Points
  - used to caculate size of software from a statement of the problem
  - tries to address variability in lines of code estimates used in models such as COCOMO
    - e.g. because SLOC varies with different languages
  - Originally for information systems, although other variants exist
  - Basic model is: $FP = a_1 I + a_2 O + a_3 E + a_4 L + a_5 F$ (I etc. = metric from problem statement; a_i = weighting factor for this metric)
- Example
  - Sets of weightings (a_i) provided for different types of project
  - Measure properties of the problem statement:
    - I = number of user inputs (data entry)
    - O = number of user outputs (reports, screens, error messages)
    - E = number of user queries
    - L = number of files
    - F = number of external interfaces (to other devices, systems)
  - Example calculation:
    - $FP = 4I + 5O + 4E + 10L + 7F$

Example model: Reliability growth

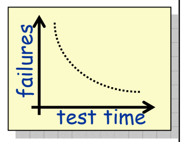
Source: Adapted from Pfleeger 1998, p359

➜ Motorola's Zero-failure testing model

- Predicts how much more testing is needed to establish a given reliability goal
- basic model:

$$\text{failures} = a e^{-b(t)}$$

(a, b: empirical constants; t: testing time)



➜ Reliability estimation process

- Inputs needed:
  - fd = target failure density (e.g. 0.03 failures per 1000 LOC)
  - tf = total test failures observed so far
  - th = total testing hours up to the last failure
- Calculate number of further test hours needed using:

$$\frac{\ln(fd/(0.5 + fd)) \times th}{\ln((0.5 + fd)/(tf + fd))}$$

- Result gives the number of further failure free hours of testing needed to establish the desired failure density
  - if a failure is detected in this time, you stop the clock and recalculate
- Note: this model ignores operational profiles!

Example model: Cyclomatic Complexity

Source: Adapted from van Vliet, 1999, pp308-311

➜ McCabes' complexity measure

- This is a measurement model, not a predictive model
- It measures complexity as a function of the number of paths through a program
- Basic model is:

$$CV = e - n + p + 2$$

(CV: "cyclomatic complexity"; e: number of edges; n: number of nodes; p: number of graphs (procedures))

➜ Application

- Draw each module as flowchart
- Convert each flowchart to a graph
  - nodes show statements, edges show control paths
  - branches (IF, WHILE, etc) have multiple edges coming out of them
- Count edges and nodes in each graph
- CV also corresponds to the number of linearly independent paths in the graph
- CV > 10 is usually taken as an indicator that a module is overly complex
- But the validity of this measure is hotly disputed!

But software measurement is hard

➜ Key problems for software measurement:

- Most attributes of interest cannot be measured directly
- Most metrics are very hard to validate
- Most models are at best vague approximations
  - The validity of each of the models described is disputed
  - Models usually have to be adapted to a particular organization
  - Need to collect data over a long period to validate and adapt the models
- The technology keeps changing
  - parameters for these models are derived from past projects which might be unlike future projects

➜ Predictive models can be self-fulfilling

- Predictive model is used to generate effort and time estimates
  - …which are used to generate a project plan
  - …which is used by managers to manage the project to
  - …so the project ends up having to conform to the estimate!

➜ But you cannot control it if you cannot measure it

- poor models may be better than no models at all
- predictions will need to be continuously revised as the project proceeds

References

van Vliet, H. "Software Engineering: Principles and Practice (2nd Edition)" Wiley, 1999.

Chapter 6 has some introductory comments about measurement of various different things in software engineering, especially with respect to any attempt to measure software quality. Various metrics are introduced throughout the book, at appropriate places. For example, cost estimation (COCOMO, Function Point Analysis, etc) is in chapter 7; measurement of design complexity (Halstead, McCabe, …) is in chapter 11; measurement of testing (test coverage, test adequacy criteria) is in chapter 13; and reliability estimation is in chapter 18. This is of course appropriate – measurement should be an integrated part of software engineering, not something you bolt on afterwards!

Pfleeger, S. L. "Software Engineering: Theory and Practice" Prentice Hall, 1998.

Pfleeger's research area is software measurement, so she gives it a very strong treatment throughout her book.