How to measure a system

Perdita Stevens

School of Informatics
University of Edinburgh

Measuring what?

A metric is simply a measure of something. Of what?

In this lecture, we consider two quite separate things that both often need to be measured and/or estimated: software size and project cost and reliability.

Reliability follows on from the last lecture on verification; size and cost looks forward to the future lectures on project management.

Measuring why?

- to get advance warning of problems on this project;
- to get information with which to judge how to do future projects better.

Classifying metrics

Let's focus for a moment on "measured and/or estimated" above.

A metric may be a:

- predictor – use one measurement (e.g. number of functions) to predict another (e.g. cost of project). The thing used is a predictor metric.
- result – if a measurement was predicted by a predictor metric, measure its actual value at the end of a project (or phase). Feed the discrepancy back into process improvement and future estimation.

Usually we measure predictor metrics to estimate result metrics in advance, then we measure the result metrics, and use the discrepancy to improve our estimation process.

Complications arise in that not all result metrics are easily measurable.

Useful metrics should ideally be...

- Measurable – e.g. not someone's opinion of how complex something was
- Independent – i.e. not something that can be altered without altering something we care about
- Accountable – i.e. managers should be able to justify both the use of this metric and the value given to it.
- Precise – i.e. the minimal accuracy of the number must be known.
- MEANINGFUL!! – there must be some reason to believe that numbers for the metric have something to do with something we care about!

Reliability

Reliability is a key non-functional requirement in many systems. But how does one specify reliability?

Several ways – most appropriate depends on the nature of the system.

POFOD

Probability of failure on demand is the probability that the system will fail when service is requested.

Mainly useful for systems that provide emergency or safety services. E.g. the emergency shutdown in a nuclear power plant will never be used – but if it is, it shouldn't fail. ('New' Sizewell B Primary Protection System specified 0.0001 – and achieved 0.001 in testing.)

How to evaluate? Repeated tests in simulation.

ROCOF

Rate of failure occurrence is the number per unit time of failures (unexpected behaviour). 'Time' may mean elapsed time, processing time, number of transactions, etc.

Mainly used for systems providing regular service, where failure is significant. E.g. banking systems. (VisaNet processes around 10^8 transactions/day. Failure rate is not published, but probably (much) less than 10^{-5} failures/transaction.)

MTTF

Mean time to failure is the average time between successive failures.

Mainly used where a single client uses the system for a long time. E.g. CAD systems – or indeed desktop PCs. Popular metric for hardware components.

MTTF

Mean time to failure is the average time between successive failures.

Mainly used where a single client uses the system for a long time. E.g. CAD systems – or indeed desktop PCs. Popular metric for hardware components.

Q: What's the difference between MTTF and ROCOF?

MTTF

Mean time to failure is the average time between successive failures.

Mainly used where a single client uses the system for a long time. E.g. CAD systems – or indeed desktop PCs. Popular metric for hardware components.

Q: What's the difference between MTTF and ROCOF?

Q: You buy a hard drive with an MTTF of 5 years. When will you replace it?

Availability

Availability is the proportion of the time that the system is 'available for use'. Often quoted as 'five nines', meaning 0.99999, 'four nines' etc.

Appropriate for systems offering a continuous service, where customers expect it to be there all the time. ('Five nines' is achieved by large data processing systems (e.g. Visanet) – running on IBM mainframes, not PCs!)

Availability

Availability is the proportion of the time that the system is 'available for use'. Often quoted as 'five nines', meaning 0.99999, 'four nines' etc.

Appropriate for systems offering a continuous service, where customers expect it to be there all the time. ('Five nines' is achieved by large data processing systems (e.g. Visanet) – running on IBM mainframes, not PCs!)

Q: What's the difference between availability and ROCOF?

MCQ from 05/06 exam

Aircraft that have to make an emergency landing usually dump ('jettison') fuel first, to reduce the risk of fire. You are writing the non-functional requirements for the part of the aircraft control system that controls the jettison pumps. Which reliability metric would you use?

1. mean time to failure
2. probability of failure on demand
3. rate of occurrence of failures
4. availability

Cost estimation

Before starting, or bidding for, any significant project, need to know how much it will cost.

Many things are factors in this estimation: but the main factors are software size and complexity, and engineer productivity.

We will here consider only software size and complexity. (Productivity is ratio of this to time required.)

LoC

A simple measure of software size is lines of code (LoC).

Not very meaningful by itself. What is a line of code?

How many lines of Haskell correspond to how many lines of Java correspond to how many lines of C? What about library routines?

Still widely used.

Function points

Can try to measure something a bit more objective about *functionality* of system. The number of function points in a program is a weighted sum of

- external inputs and outputs
- user interactions
- external interfaces
- internal system files

(typical weights: 4, 4, 10, 7), or similar.

Then further corrections are made to account for aspects of the system such as distribution, code re-use, etc.

Object points are a variation aimed at transaction processing software.

Estimation

Metrics are all very well, but how do you guess (sorry, estimate) them for software that doesn't exist yet? Some approaches:

- algorithmic cost modelling: develop (from past data) a model relating size/complexity to ultimate cost
- expert consensus: get a bunch of expert estimates. Compare, discuss, repeat until convergence.
- analogy: relate the cost to that of similar completed projects
- available effort: no comment
- what the customer will pay: or what you think you need to bid

COCOMO

COCOMO is a long-standing algorithmic model, publicly available, well supported, and widely used. It has a number of sub-models.

Required Reading: Sommerville §26.3.1. (Sommerville chapter 26 is available as PDF at `http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/SampleChapters/ch26.pdf`

Answer the following:

- how does COCOMO estimate the cost of automatically generated code?
- what is the purpose of the exponent in the early design model estimate?
- A client offers to pay your firm a £200k bonus if you deliver their software system six months ahead of schedule. Which COCOMO parameter will you change to in your cost estimation?

Why do projects almost always slip...

... relative to human intuitions of how long they should take?

(This is why we need something like COCOMO.)

Discussed in paper *The Rational Planning of (Software) Projects*, Mark C. Paulk, unfortunately no longer available online.

This paper discusses the effects of three features of human nature:

- people tend to be risk-averse when there is a potential of loss
- people are unduly optimistic in their plans and forecasts
- people prefer to use intuitive judgement rather than (quantitative) models

It goes on to discuss how a framework like the CMM can help. We'll discuss CMM in a later lecture.

Reading

Required: Sommerville chapter 26.3.1, see above

Suggested: The rest of Sommerville Ch26.