Software Measurement

We can't accurately measure software, yet we must have measures if we are to understand large-scale design.

- What are the issues?
- Which measures are appropriate to them?
- How do we identify and prioritise issues?
- What should be in a measurement plan?
- How severe are our limitations?
- How do we use indicators and estimators?

Why Measure?

In traditional, structured lifecycles we want to:

- Assess and manage risk.
- Trade design decisions against others.
- Track progress.
- Justify objectives.

but software resists measurement.

Issues for Measurement

Schedule : Is it on time?

Cost : Can we afford to finish?

Growth : Will it scale?

Quality : Is it well made?

Ability : How good are we at design?

Technology : Is the technology viable?

These interact (*e.g.* ability → cost → shedule → quality → growth).

Issue Categories (1): Schedule

Category	Measure
Milestone	Date of delivery
Work unit	Component status Requirement status Paths tested Problem report status Reviews completed Change request status

Issue Categories (2): Cost

Category	Measure
Personnel	Effort Staff experience Staff turnover
Financial performance	Earned value Cost
Environment availability	Availability dates Resource utilisation

Issue Categories (3): Growth

Category	Measure
Product size and stability	Lines of code Components Words of memory Database size
Functional size and stability	Requirements Function points Change request workload

Issue Categories (4): Quality

Category	Measure
Defects	Problem reports Defect density Failure interval
Rework	Rework size Rework effort

Issue Categories (5): Ability

Category	Measure
Process maturity	Capability maturity model level
Productivity	Product size/effort Functional size/effort

Issue Categories (6): Technology

Category	Measure
Performance	Cycle time
Resource utilisation	CPU utilisation I/O utilisation Memory utilisation Response time

Identifying Issues

- Risk assessments.
- Project constraints (*e.g.* budgets).
- Leveraging technologies (*e.g.* COTS).
- Product acceptance criteria.
- External requirements.
- Past projects.

Prioritising Issues

Issue	Probability of occurrence	Relative impact	Project exposure
Aggressive schedule	1.0	10	10
Unstable reqs	1.0	8	8
Staff experience	1.0	5	8
Reliability reqs	0.9	3	4
COTS performance	0.2	9	1

Making a Measurement Plan

- Issues and measures.
- Data sources.
- Levels of measurement.
- Aggregation structure.
- Frequency of collection.
- Method of access.
- Communication and interfaces.
- Frequency of reporting.

Limitations (1)

- Milestones don't measure effort, only give critical paths.
- Difficult to compare relative importance of measures.
- Incremental design requires measuring of incomplete functions.
- Important measures may be spread across components.
- Cost of design is not an indicator of performance.
- Current resource utilisation may not be best.
- Reliable historical data is hard to find.
- Some software statistics are time consuming to collect.
- Some measures only apply after coding has been done.

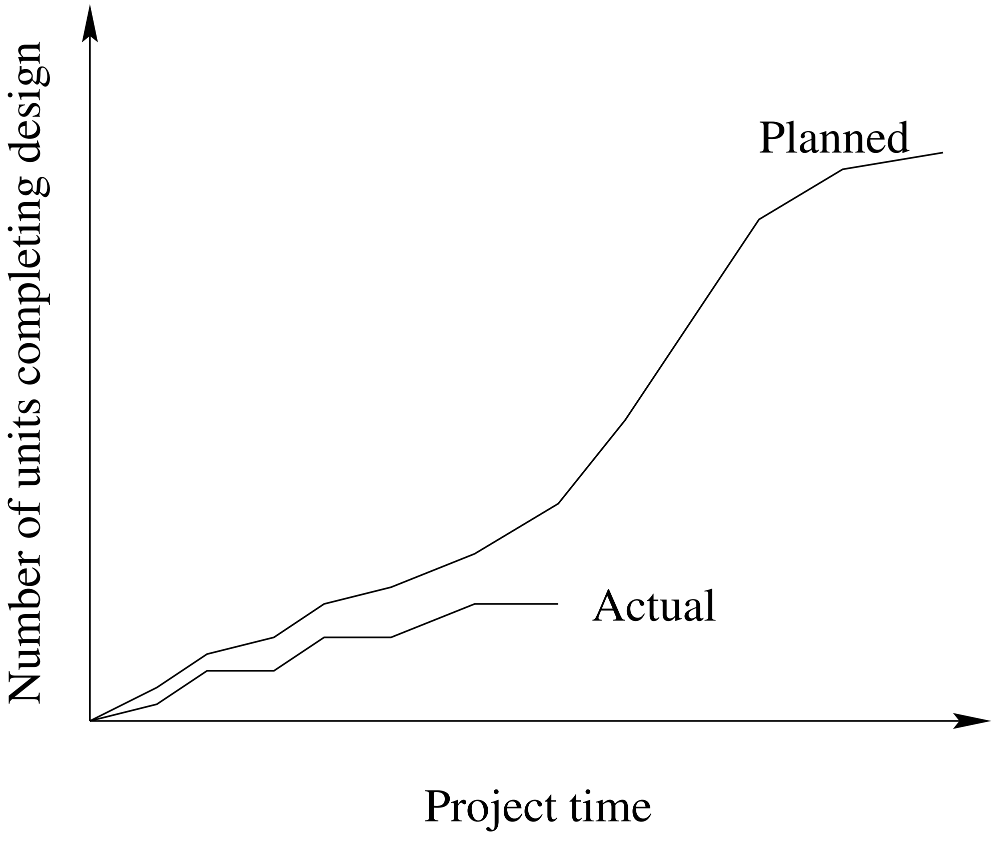
Limitations (2)

- Size doesn't map directly to functionality, complexity or quality.
- Time lag between problems and their appearance in reports.
- Changes suggested by one performance indicator may effect others.
- Often no distinction between work and re-work.
- Overall capability maturity level may not predict performance on a specific project.
- Technical performance measures often are not as precise as they may seem.
- Technical resource utilisation may only be known after integration and testing.

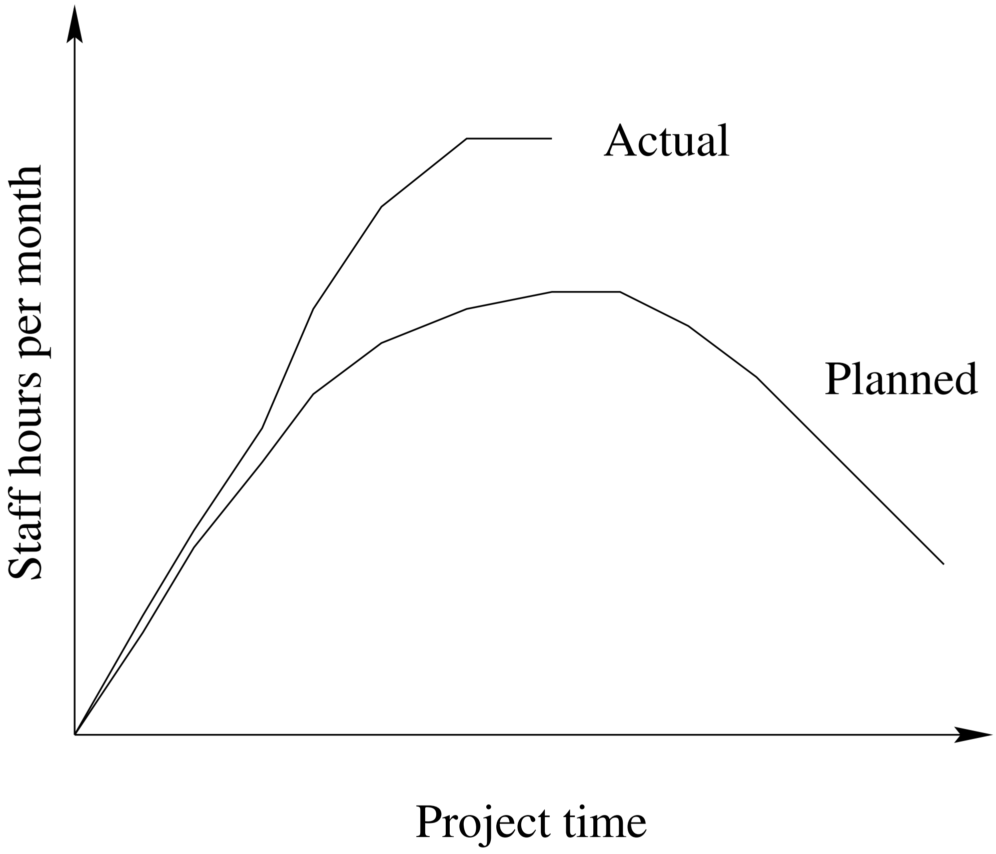
Checking Your Data

- Are units of measure comparable (*e.g.* lines of code in Ada versus Java)? Normalisation?
- What are acceptable ranges for data values?
- Can we tolerate gaps in data supplied?
- When does change to values amount to re-planning.

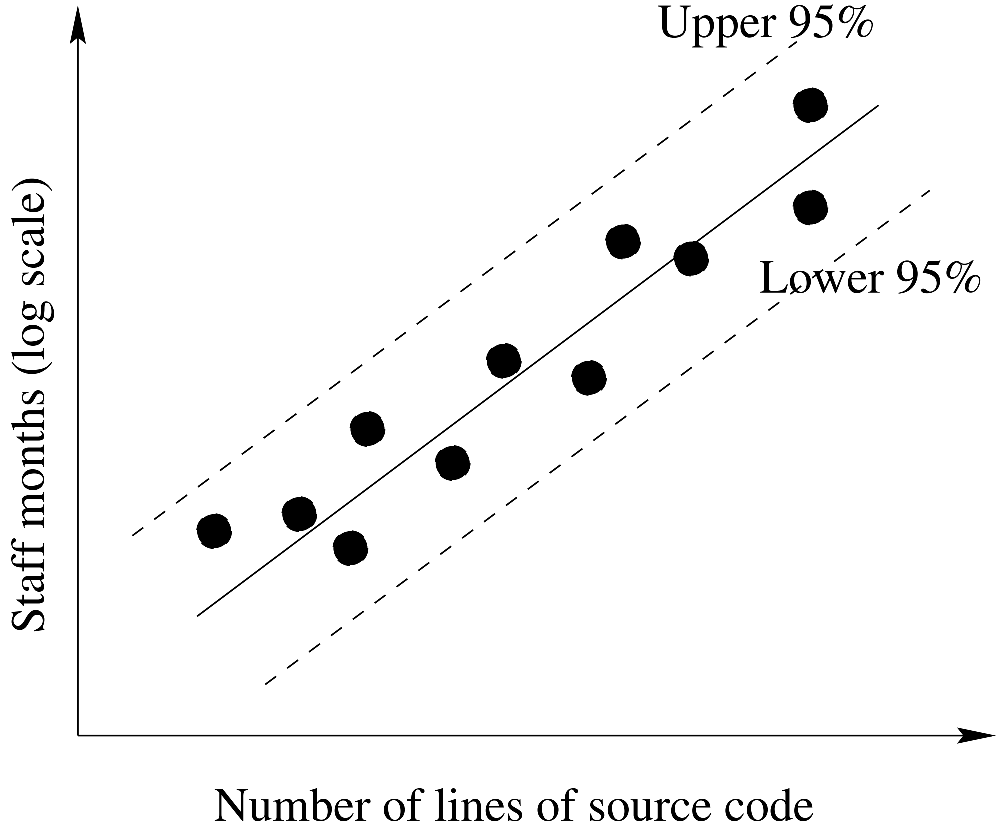
Indicator (1): Design Progress



Indicator (2): Effort



Estimator: Size-Effort


Upper 95%
Lower 95%
Staff months (log scale)
Number of lines of source code