Software Measurement

Dr. James A. Bednar

jbednar@inf.ed.ac.uk

http://homepages.inf.ed.ac.uk/jbednar

Dr. David Robertson

dr@inf.ed.ac.uk

http://www.inf.ed.ac.uk/ssp/members/dave.htm

Why Measure?

When planning and executing a project, we want to:

- Assess and manage risk
- Trade design decisions against others
- Track progress and reevaluate when necessary
- Verify that objectives have been met
- Predict how the project will go in the future

To do this well requires accurate, timely measurement, particularly crucial for waterfall-type methodologies.

How to Measure

Before measuring we should answer several questions:

- What are the issues we need to measure?
- Which measures are appropriate to them?
- How do we identify and prioritize issues?
- What should be in a measurement plan?
- How severe are our limitations?

Bear in mind that the limitations are often very strong, and obtaining useful data is difficult and sometimes impossible.

Issues to be Measured

1. **Schedule** : Can we expect it to be done on time?
2. **Cost** : Can we afford to finish this project, or will it end up costing more than it is worth?
3. **Growth** : Is the project stable, or expanding in size and scope?
4. **Quality** : Is the product being made well, with few bugs?
5. **Ability** : How talented is our team at design, coding?
6. **Technology** : Is the underlying technology viable?

Most of these interact strongly with the others.

Issues 1. Schedule

What you want to know:	What you can measure:
Is progress being made?	Dates of milestone delivery
Is work being done?	Components completed Requirements met Paths tested Problem reports resolved Reviews completed Change requests completed

Issues 2. Cost

What you want to know:	What you can measure:
How much is it demanding of our staff?	Total effort Number of staff involved Staff experience levels Staff turnover
Are we getting our money's worth?	Earned value Cost
Is project making good use of external resources?	Availability dates (too early, late?) Resource utilization

Issues 3. Growth

What you want to know:	What you can measure:
How large is this program?	Lines of code Number of components Words of memory Database size
How much does this program accomplish?	Requirements met Function points Change requests completed

Issues 4. Quality

What you want to know:	**What you can measure:**
Are there a lot of bugs?	Problem reports Defect density Failure interval
How hard was it to fix the bugs?	Rework size Rework effort

Issues 5. Ability

What you want to know:	What you can measure:
Is the development process well managed?	Capability maturity model level
How productive is this team?	Code size / effort Functional size / effort

Issues 6. Technology

What you want to know:	**What you can measure:**
Is the program fast enough?	Cycle time
Are the resources required by the program acceptable?	CPU utilization I/O utilization Memory utilization Response time

Identifying Issues

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

Prioritizing Issues Example

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

Limitations 2

- Reliable historical data is hard to find
- Some software statistics are time consuming to collect
- Some measures only apply after coding has been done
- Size doesn’t map directly to functionality, complexity or quality
- Time lag between problems and their appearance in reports

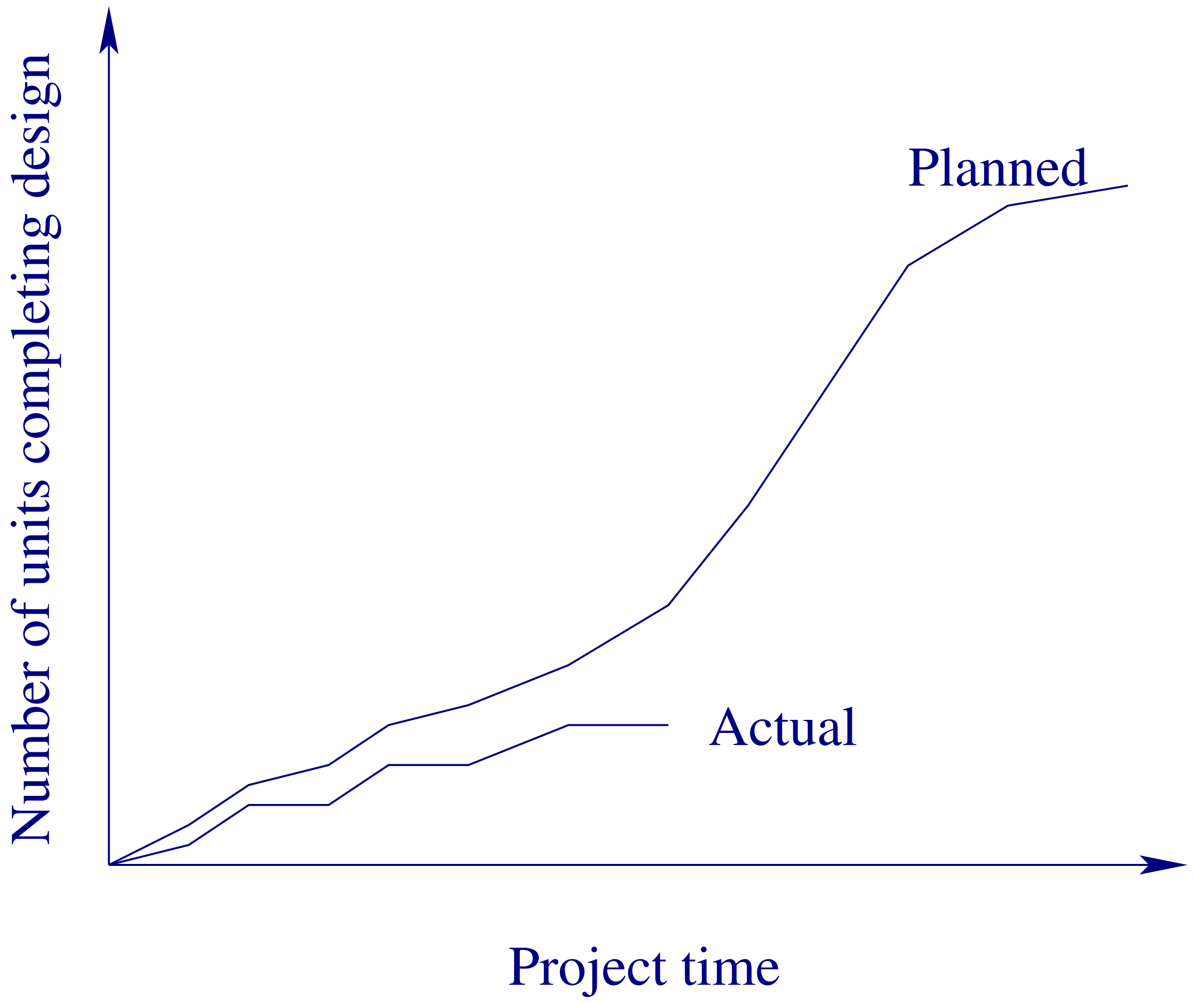
Limitations 3

- Changes suggested by one performance indicator may affect 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 misleadingly precise, yet not very accurate
- Technical resource utilization may only be known after integration and testing

Checking Your Data

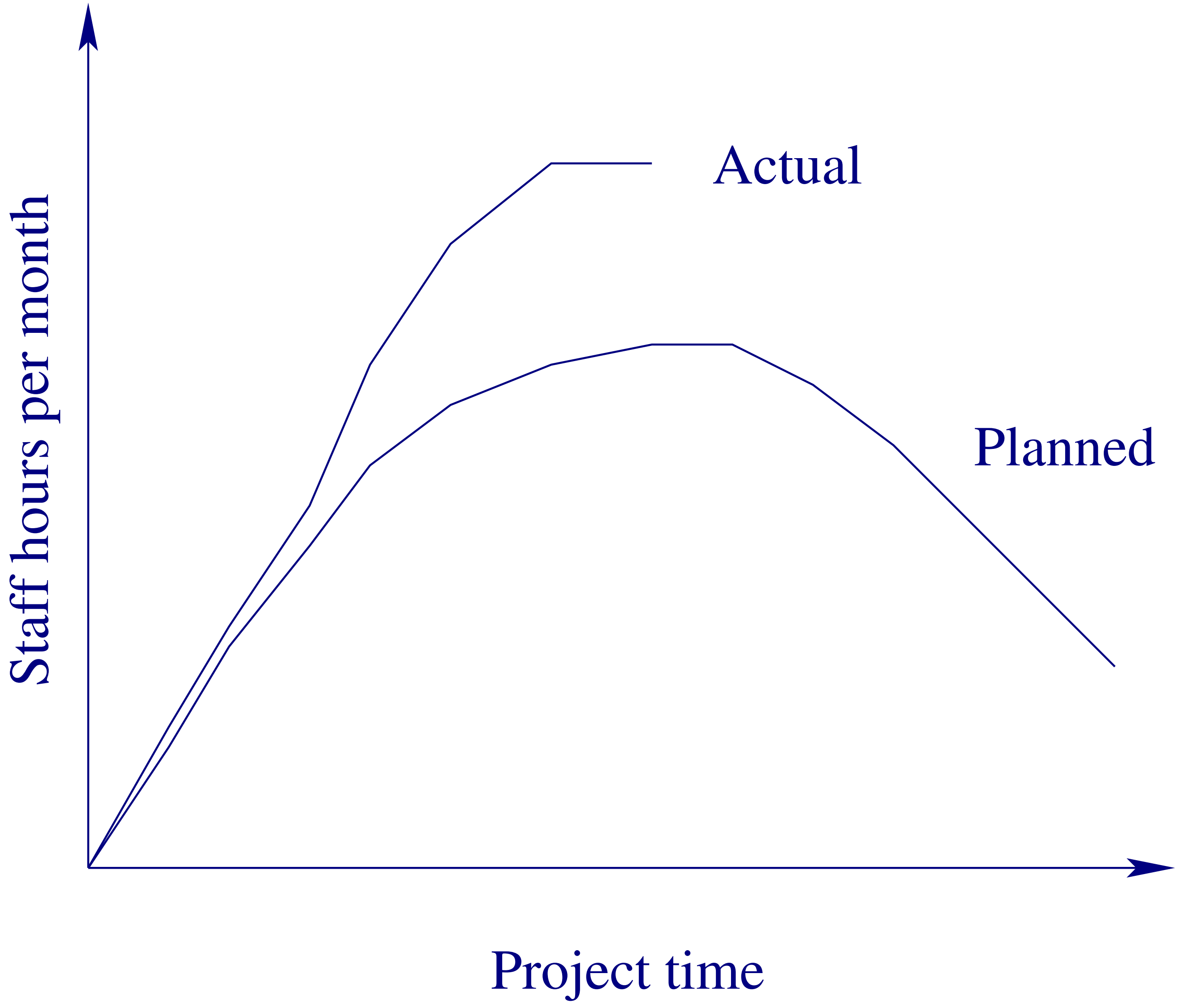
- Are units of measure comparable (e.g. lines of code in Ada versus Java)? Normalization?
- What are acceptable ranges for data values?
- Can we tolerate gaps in data supplied?
- When does change to values amount to re-planning?

Indicator 1. Design Progress

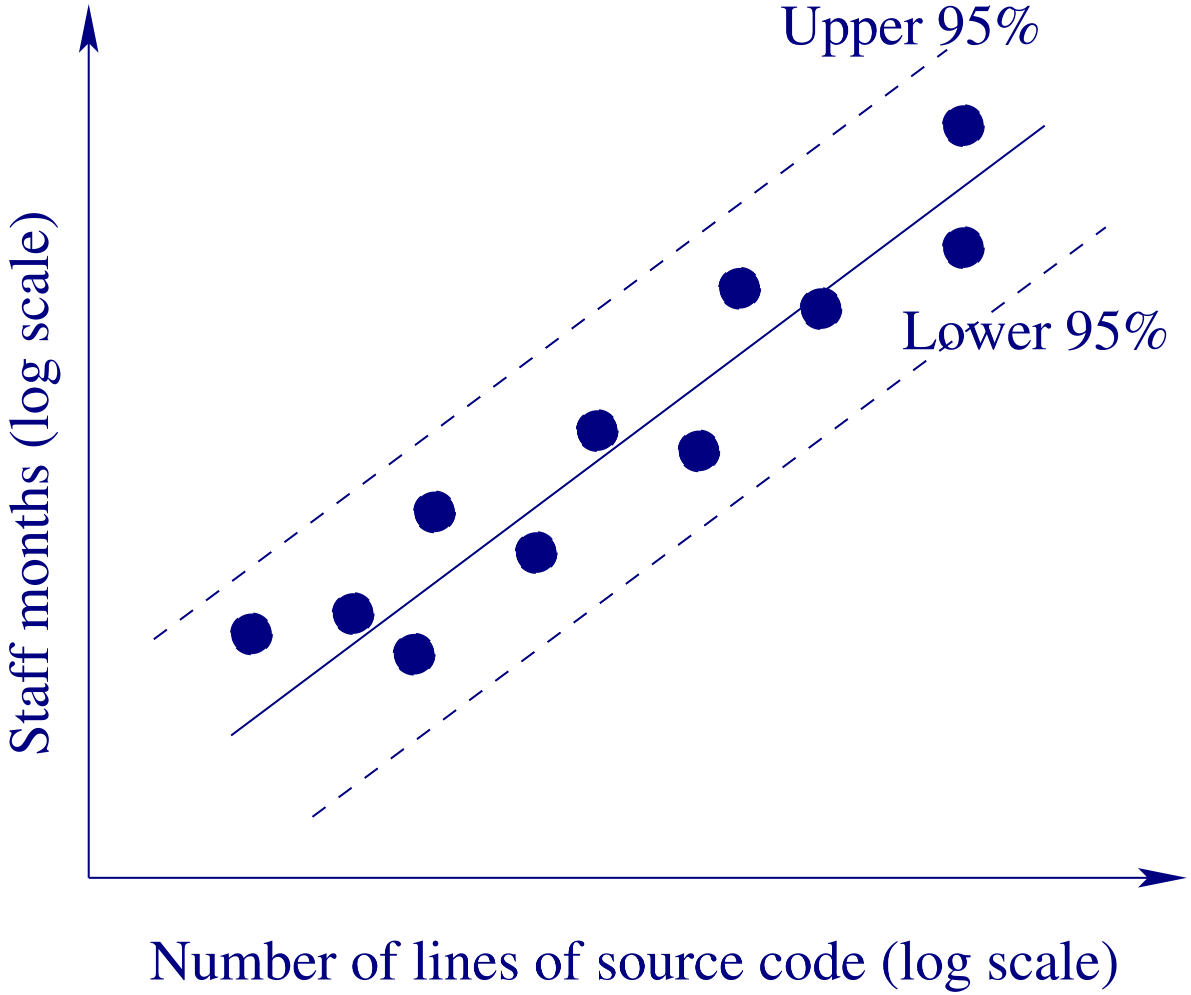


With an indicator and a plan, you can see if you are on track.

Indicator 2. Effort



Estimator: Size-Effort



With enough data, you can try to predict future performance.

Summary

- Measurement is time-consuming, difficult, and impossible to do perfectly
- You need to choose what you want to find out, and how to approach measuring that
- Always be aware of the limitations of the measurement and of how it relates to what you really want to know
- Be careful when trying to relate past performance to the future