

# Software Estimation

# Agenda

- ✍ The estimation problem
- ✍ Prerequisites for good estimation procedures
- ✍ Definition of an “estimate”
- ✍ Components of an estimate
- ✍ Using estimates to support business requirements
- ✍ Estimation Techniques

# Problem 1 - Software

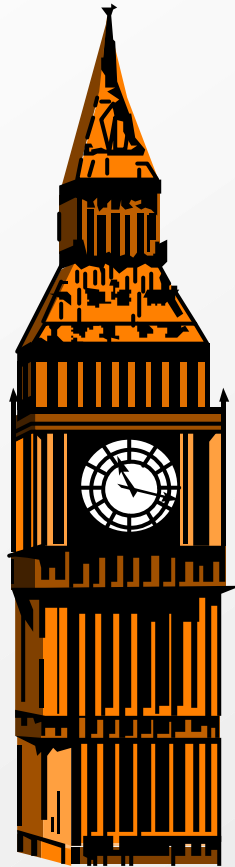
✍ Estimation is accurate for:

- Well-defined products
- Automated processes

✍ Software:

- Novel products
- Changing processes
- Intellectually-intensive tasks

# Problem 2 - Politics



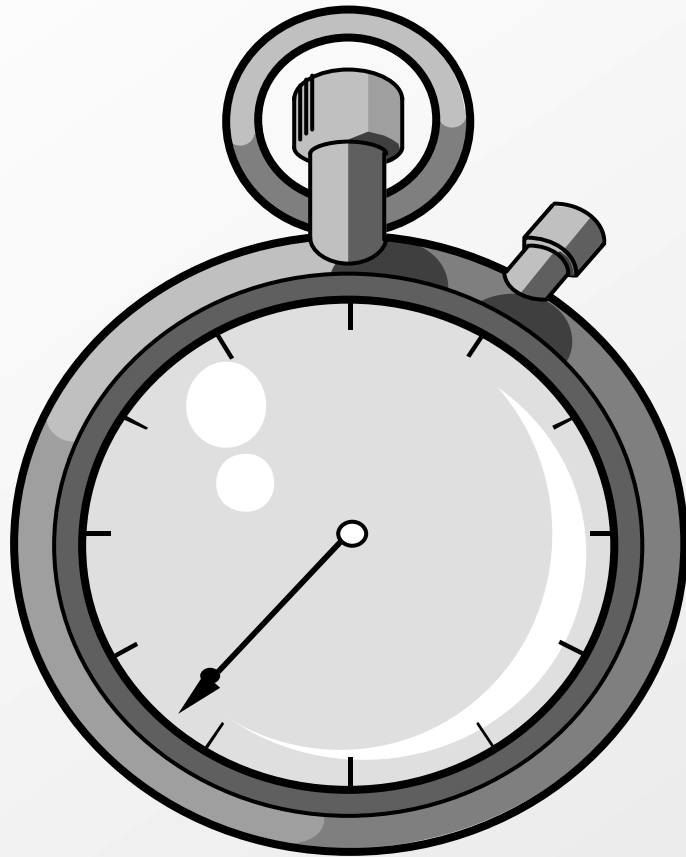
## Internal

- “The boss doesn’t want to know”

## External

- “The customer doesn’t want to know”
- “We’ll recover costs when they make changes”

# Poor Bid Management



✍ Estimates are required:

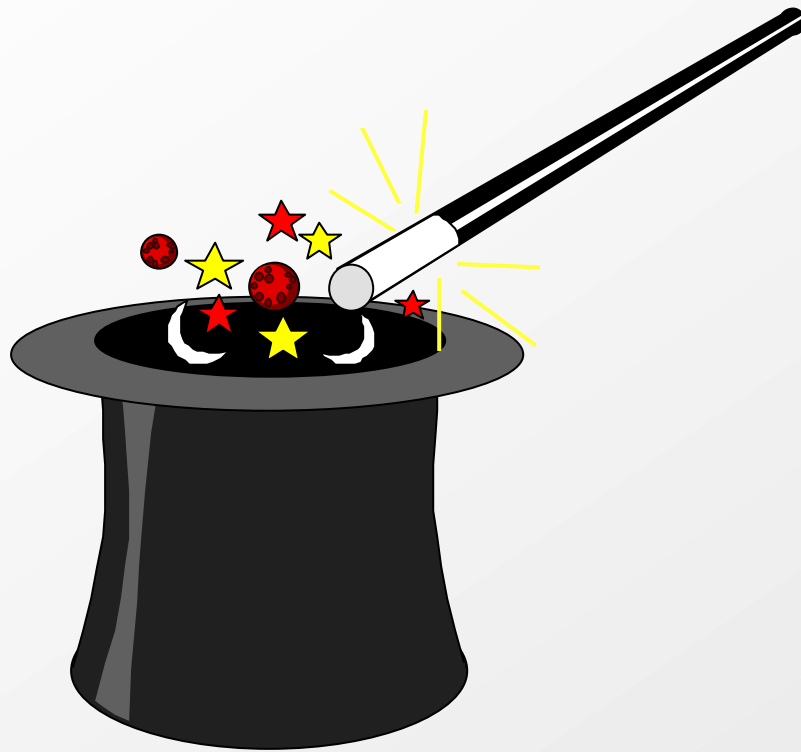
- In short timescales
- With little product information available

# Problem 3 - No procedures



- ✗ Little historic data
- ✗ Few defined methods
- ✗ Limited expertise

# Problem 4 - No magic solution



- ✍ No single “best” method
- ✍ No guarantee of accuracy
- ✍ Requires investment
  - Time & effort
  - Infrastructure

# Basis of good estimates

- ✍ Local estimation process
- ✍ Matching local development and business procedures
- ✍ 20 years of research results around the world confirm
- ✍ You can't rely on:
  - Other people's data
  - Off-the-shelf metrics and models

# Prerequisites - 1

- ✍ Estimates based on a past performance
- ✍ Within a well-defined business area
  - Producing similar applications
    - ✍ standard product types
  - Using common, agreed development methods
  - Using teams of similar capability
    - ✍ staff differences usually masked by team similarity

# Prerequisites - 2

- ✍ Organisations improvement requires quantitative records of past projects:
  - Effort per activity
  - Early size measures
  - Duration
  - Staffing levels
- ✍ Including estimates and constraints
- ✍ All properly defined
  - Repeatable and comparable

# Prerequisites - 3

- ✍ Estimation “standards”
- ✍ Packaged to cater for
  - “local conditions”
  - application type
  - business process

# Prerequisites - 4

## Feedback to estimation process:

- To assess estimate accuracy
- To improve estimation procedures
- To respond to development process changes

## Feedback to estimators

## Feedback to development processes

- Identify effort/schedule bottlenecks
- Focus development process improvements

# Software cost estimates


## Forecasts:

- Effort for production & maintenance
- Duration
- Staffing levels



# An estimate should be...

## An unbiased prediction:

- Most likely value not least possible value with non-zero probability
- Qualified by
  -  Upper and lower bounds







## Understandable within the context of:

- Assumptions
- Inherent risk
- Estimation method

## Transitory

## An input to other processes

# An estimate should not be...

-  A target or constraint
-  A planning value
-  A single fixed value
-  A price
-  Fixed for eternity
-  An end in itself

# Estimate scope

- ✍ What “metrics”
  - defined units (hours, days)
- ✍ Which activities/tasks
  - management?
  - validation?
  - debugging?

# Bounds

- ✍ Quantify the accuracy of an estimate
- ✍ But may not be accurate themselves!
- ✍ Depends on how they are calculated
  - state method used

# Bound calculation methods

- ✍ Best method (accuracy performance)
  - estimate accuracy achieved on past projects
  - at comparable stage in project
- ✍ 2nd best (statistical confidence limits)
  - 95% confidence limits from statistical estimation model
- ✍ 3rd best (main driver variation)
  - estimation model with best and worst input values
- ✍ Worst (best guess)
  - Gut-feel

# Assumptions

## Explicit assumptions:

- inputs to any estimation model used
- “similar projects” and specific adjustments if analogy is used

## Implicit assumptions:

- “Normal conditions”

 Must state any implicit assumptions that are conditional

# Risks

- ✍ Risks are derived from assumptions
- ✍ Assume an assumption is invalid:
  - identify alternative states of nature
  - assess probability that alternative state of nature will occur
  - recalculate estimate if alternative assumption is true ( $E_2$ )
  - calculate contingency
    - ✍  $(E_2 - E_1) \cdot \text{Prob}(E_2)$
  - calculate risk adjusted estimate
    - ✍  $E_1 + \text{contingency}$

# Risks v. Bounds

## In the absence of risk assessment

- estimate users must choose appropriate value from range
- expressing their risk preference!

## After risk assessment


- the risk adjusted estimate is best choice

# Use of software estimates

## Throughout lifecycle:

- Feasibility & bidding

  -  Input to bid managers

    -  Least accurate but most important

## Programme management

- Input to release definition

# Use of Estimates

## Project planning

 Input to project managers

 Usually increased accuracy but constrained by contract

## Project monitoring & control

 Input to project managers

 Feedback of actuals can refine estimate

# Summary - 1

*It is very difficult to make predictions  
especially about the future*

# Summary - 2

- ✍ Software estimation has special problems
- ✍ Improved by an estimation processes calibrated to local conditions
  - ✍ measures
  - ✍ models
  - ✍ procedures & standards
- ✍ Contingency calculations
  - assumption-based risk assessment

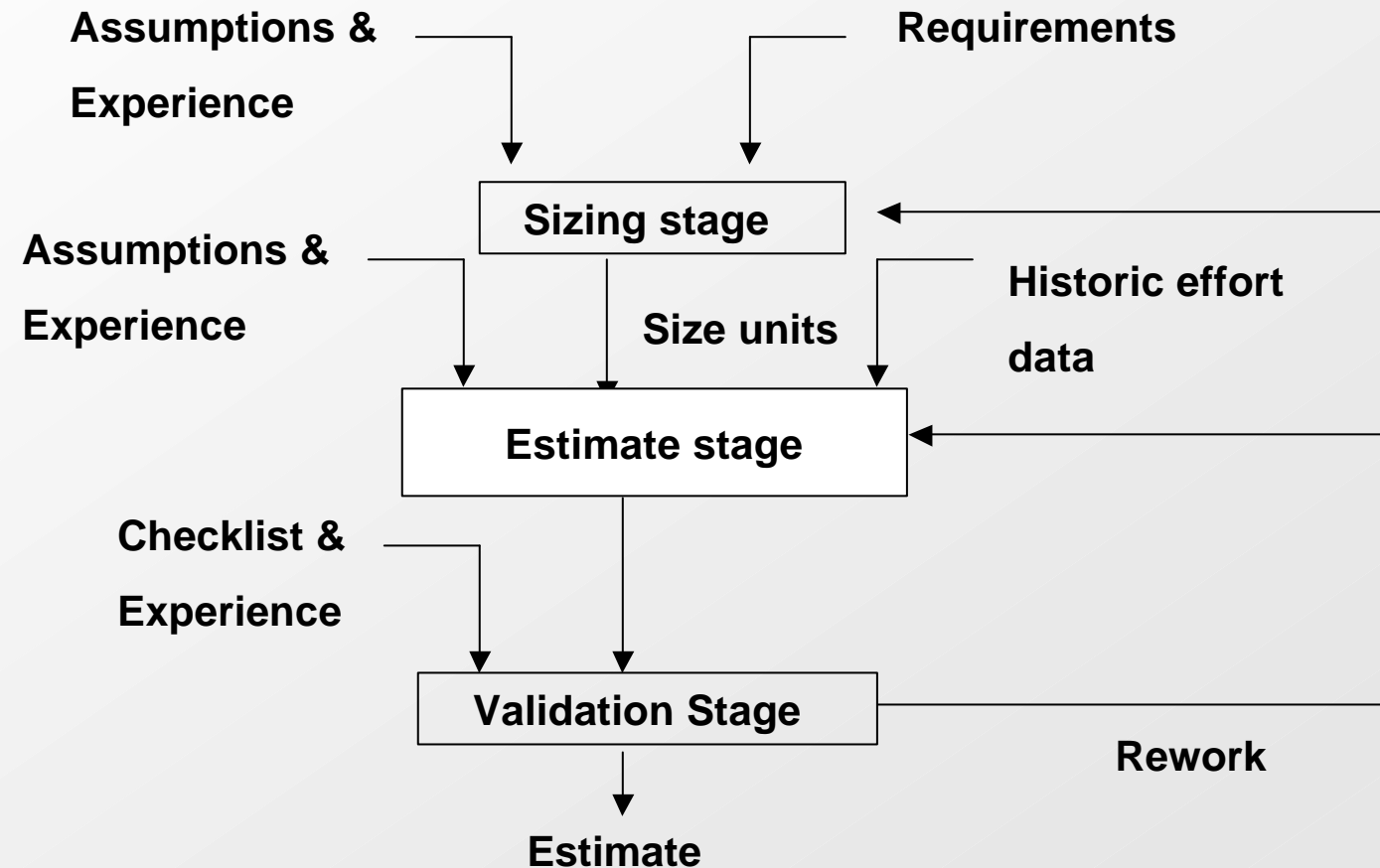
# Estimation Procedures and Techniques

# Agenda

- ✍ Basic estimation procedure
- ✍ Basic estimation methods
- ✍ Estimating size
- ✍ Estimating effort
- ✍ Estimating duration

# Basic Estimating Procedure

# Estimation Procedure



# Output

- ✍ Complete estimate
- ✍ Effort plus estimate range
  - For specified scope
- ✍ Elapsed time
- ✍ Phase/activity estimate
- ✍ Assumptions
  - Dependencies, project characteristics
- ✍ Risk Assessment
  - Estimate contingency

# Sizing Stage

## Sizing process

- Step 1: Identify units (software items to count)
- Step 2: Identify characteristics that affect development effort of each unit
- Step 3: Count the number of units in each category associated with each characteristic
- Step 4: Identify any assumptions and risks

# Examples

## “Units”:


 Windows

 Objects

 Functions

 Modules

## Characteristics:

 History - new/adapted/bought-in

 Size: small/medium/large

 Complexity: low/medium/high

# Module Size Table

Modules	Origin	
Size	New	Adapted
Large	1	4
Average	2	3
Simple	5	7

# Assumptions and Risks

## Accuracy of counts:

- Minimum/Most Likely/Maximum

## Requirements:

- Stability
- Completeness

## Risks

- Impact of incorrect or invalid assumptions

# Estimation Stage

## Estimation Process

- Step 1. For each unit type, identify average productivity from historical data
- Step 2. Identify scaling factors
- Step 3. Use basic estimation model to calculate effort
- Step 4. Sum individual estimates to produce total estimate
- Step 5. Identify any assumptions and risks

# Assumptions and Risks

## Scaling factors:

- Complete
- Impact accurately assessed

## With multiple item types:

- Effort for each item type is independent

## Risks

- Impact of incorrect or invalid assumptions

# Validation stage

- ✍ Estimate checked by other people
  - Another engineer
  - The project team
  - Manager
    - ✍ (Engineering, Technical, Administration, Programme)
- ✍ Using checklist
  - Is the estimate realistic?
  - Is sizing complete?
  - Are all required components of estimate present ?
- ✍ Another estimation method?

# Applying the Estimation Process

✍ Need methods to:

- Identify and “count” units
- Select characteristics and categories
- Use historic data to determine productivity
- Identify level to estimate effort
  - ✍ All software development, individual phase, individual task
- Select scaling factors
- Identify level to apply scaling factor
  - ✍ Unit, phase, all software development

# Basic Estimating Techniques



# Estimating Methods

## Expert Opinion

- Personal experience

## Analogy

- Reference to example

## Algorithm

- Functional relationship

# Sizing v. Estimating

## Sizing:

- Estimating size of software
- Preliminary to effort/duration estimate

## Sizing is another estimation problem:

- Same basic estimation techniques

# Expert Opinion

- ✍ Estimator uses personal judgment
  - Based on many different personal experiences
- ✍ Example
  - Estimating duration of presentation
    - ✍ Usually 2 minutes per slide for a presentation
    - ✍ Exclude title slide
    - ✍ Double time for training session
- ✍ Most common method of estimation

# Pros and Cons

## Pros

- Flexible
- Utilises understanding of problem and solution
- Doesn't require past data

## Cons

- Depends on expert
- Usually non-transferable
- Difficult to check

# Analogy

## ✍ Estimator uses example

- Actual results of a similar project
- Systematic assessment of difference

## ✍ Example

- Estimating time to decorate room
  - ✍ Last room of similar size and in similar state took 3 days
  - ✍ Assume 3 days
  - ✍ Adjustment for specific difference
    - ✍ One extra door to prepare add 0.5 day

# Pros and Cons

## Pros

- Can be more accurate than other methods
- More visible process

## Cons

- Depends on availability of past data
- Depends on ability to recognise a good precedent
- Adjustments are based on expert opinion

# Exercise 1 - Analogy & Expert Opinion






## Developing software engineering tools

- Object-oriented (Objective C)
- SUN platform

## First tool:

- Analogy tool
- Total effort = 53 days
- Function size = Medium
- Classes = 34

# Estimate new tools

 Configuration	Large, 2 x Analogy
 Data entry	Medium, 1 x Analogy
 Sizing	Medium, 1 x Analogy
 Statistical estimation Analogy	Medium, 1 x
 EQF	Small, 0.5 x Analogy

# Algorithms

✍ Estimator uses a predefined formula

- Equation

  - ✍  $\text{Effort} = 2.5 \times \text{Size (LoC)}$

- Look-up table

  - ✍ Modules to new or changed LoC

Module type to LoC conversion		
Size	New	Change
Large	5000	500
Average	1000	100
Small	50	5

# Pros and Cons

## Pros

- Transferable
- Basis of estimate clear
- If formula derived statistically upper and lower bounds can be calculated

## Cons

- Assumes future same as past
- Formula must be created and updated
- Provides “average” result but not all projects are equal
- False sense of accuracy

# Exercise 2: Productivity Assessment

## Results for first 3 tools:

- Analogy: 34 classes, 53 days
- Configuration: 64 classes, 96 days
- Data entry: 43 classes, 55 days

## Sizing tool:

- Estimate: 40 classes



## Estimate total effort

# Basic Estimation Strategies

## Top-down:

- Estimate project as a whole
- Break-down estimate into phases/activities

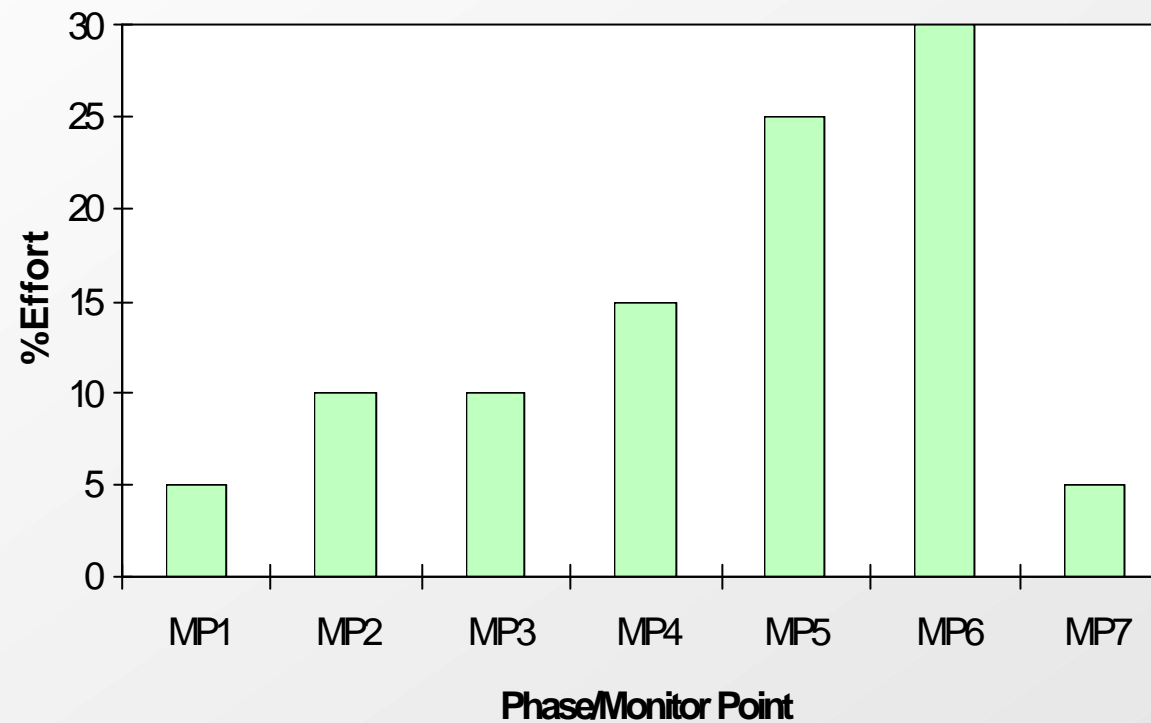
## Bottom-up

- Estimate for components
  -  Processes (tasks/phases/activities)
  -  Products (subsystems/modules/functions)
- Sum component estimates to give project estimate

# Top-down Estimating

- ✍ Useful when internal characteristics unknown
- ✍ Usable with all estimation methods except look-up tables
- ✍ Pros
  - Includes overheads
    - ✍ Non-task related activities (Management, QA)
- ✍ Cons
  - Break-down decreases accuracy

# Effort Decomposition



# Exercise 3 - Effort Profiles

<b>Project</b>	<b>Phase 1 Effort</b>	<b>Phase 2 Effort</b>	<b>Phase 3 Effort</b>
<b>Analogy</b>	<b>17</b>	<b>23</b>	<b>13</b>
<b>Configuration</b>	<b>45</b>	<b>32</b>	<b>19</b>
<b>Data entry</b>	<b>21</b>	<b>22</b>	<b>12</b>

- ✍ Estimate the effort per phase for the sizing tool

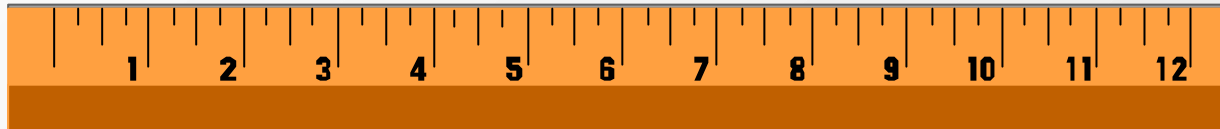
# Bottom-up Estimating

- ✍ Useful when tasks or product components defined
- ✍ Usable with all estimation methods
  - Look-up tables in particular
- ✍ Pros
  - Accumulating estimates improves accuracy
- ✍ Cons
  - Overheads can be overlooked
  - At a very low level - tasks may be overlooked

# Size and Effort

<b>Size</b>			
<b>Top-down</b>		<b>Bottom-up</b>	
<b>Effort</b>		<b>Effort</b>	
<b>Top-down</b>	<b>Bottom-up</b>	<b>Top-down</b>	<b>Bottom-up</b>
<b>Yes</b>	<b>No</b>	<b>Yes</b>	<b>Yes</b>

# Estimating Size



# Basic Sizing Tactics

- ✍ Estimate size of problem
  - Count units visible/understandable to client
    - ✍ Function Points - inputs/outputs/master files/queries/interfaces
- ✍ Estimate size of solution
  - Count units visible to developers
    - ✍ E.g. Lines of code, Modules
- ✍ Some overlap
  - Specification metrics
    - ✍ Entities, Functional primitives, Objects

# Size Dimensions

## Control

- States & Transitions

## Functions

- Processes, calculations, input facilities, reports

## Retained data

- Databases, data tables

# Size Units Selection

- ✍ For estimates early in the project
  - Problem measures preferable
  - Available from requirements specification
- ✍ For estimates of later phases
  - Solution measures preferable
  - Identify what developers will do
- ✍ Measures required from all relevant dimensions
  - Check which are needed

# Sizing Issues

- ✍ Must define basic measurement units per phase or per project
  - ✍ What items do you produce or manipulate?
  - ✍ What are the characteristics by which the items are differentiated
- ✍ When can you reasonably assess number & characteristics of items?
  - ✍ Need to know which measures to use at which estimation stage
- ✍ Unit depends on
  - ✍ Application type
  - ✍ Development methods

# Example - Sizing Options

- ✍ Requirements Phase
- ✍ Define and analyse requirements and constraints
- ✍ Requirements types:
  - Function
  - Data
  - Control
  - Quality (reliability, performance, capacity)




# Example - continued

## Software System Specification

### Units depend on specification method:

- Structured analysis (DeMarco)
- Operational analysis (JSD)
- Object-oriented analysis

## Objects





- Production effort dependent on:
  -  Data: Number of state variable
  -  Functions: Number of methods
  -  Control: Number of messages
- Categorise according to reuse

# Example - continued

## Implementation

- Number of methods
- Categorised by Size, Complexity, Origin

## System test (Validation)

- Test effort
  -  Number of tests
  -  Categorised by type, or complexity (pre-conditions, input data elements)
- Development effort
  -  Number of faults to correct
  -  Categorised by type

# Estimating Effort



# Effort Estimation

## Basic model

- Effort = Productivity ? Size ? SF

## Once size measured



- Need to determine productivity

# Productivity Evaluation

## Expert opinion

- Ask expert to assess basic productivity for each unit type and each category

## Analogy


- Identify similar project(s)
  -  Size the past project(s)
  -  Calculate average effort per unit for each unit category
- Assumes effort related to units in historical database

# Productivity Measures - 1


## Size related to whole product

- Productivity = (Tot effort)/Size

- Example

 Code size = 200KLoC

 Development effort = 20K person days

 Productivity = 0.1 day per LoC

- LoC refers to code phase but can use

 Code effort for code phase predictions

 Total development effort for total effort predictions

# Productivity Measures - 2

✍ Several size categories or size units for product

- $CP = CE/CS$

- CP = Category Productivity

- CE = Category Effort

- CS = Category Size

✍ Effort estimate for new product

- Composed from estimates for each category

# Example of Several Size Measures

- ✍ Mark II Function Points
- ✍ For each “transaction” count:
  - Number of input data elements
  - Number of entities used
  - Number of output data elements
- ✍ Sum each count for product
- ✍ Base unit productivity on weights

*Effort ?  $W_I I$  ?  $W_E E$  ?  $W_O O$*

# Productivity Measures - 3

## Different size measures for different phases

- Productivity = (Phase effort/Phase size)
- Effort estimate composed by summing phase estimates

## May have:

- Several size measures per phase
- Different categories for each size

# Derivation of module type productivity

Modules	Number	Effort(days)	Productivity
New			
Large	1	25	25.00
Average	2	37	18.50
Small	5	20	4.00
Amended			
Large	4	30	7.50
Average	3	20	6.67
Small	7	20	2.86


# Statistical Methods

- ✍ Preparation of productivity look-up tables
  - Calculate ranges:
    - ✍ Mean +/- 2 standard deviations
- ✍ Development of algorithmic models


# Statistically-derived Models


## Pros

- More flexible models
- Cater for multiple size measures automatically

  $\text{Effort} = a + bS1 + cS2$

- Can cater for non-linear relationships

  $\text{Effort} = a\text{Size}^b$

  $\text{Effort} = a + bS1^2 + cS2^2$

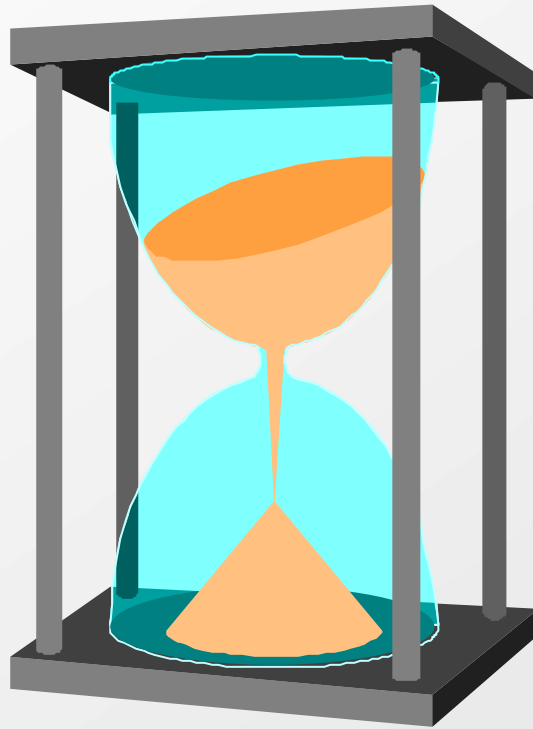
## Cons

- Usually need 7+ past projects

# Accurate Effort Recording

- ✍ Are your effort records valid?
- ✍ Factors that affect effort recording
  - Unpaid overtime
  - Inconsistent overhead recording
    - ✍ Technical authors
    - ✍ Senior managers
    - ✍ Quality Assurance
    - ✍ Tool support

# Estimating Duration



# Duration Estimation Issues

- ✍ Duration is non-linear
  - With respect to effort and size
- ✍ Can develop algorithmic models
  - Duration =  $c \cdot \text{Effort}^d$
- ✍ But, duration is not usually a “free variable”
  - Depends on effort and team size
  - Affected by overheads
  - Affected by overtime

# Preliminary Estimates

- ✍ Duration =  $(1.25 \times \text{Effort}) / (\text{Teamsize})$ 
  - 1.25 allows for 4 project days per 5 working days
  - Teamsize = Effective full time staff
  - Part-timers =  $(\text{Effort}/\text{day}) \times \text{AF}$  where  $\text{AF} = 0.8$  for developers and  $\text{AF}=1$  for managers/QA
- ✍ Applies to phase or development
  - But phases overlap so duration doesn't sum

# Conclusions

- ✍ One simple model
  - Size ? Estimate ? Validate
- ✍ Implies *many* different techniques
  - Which should you use?
- ✍ Depends on
  - Stage in project
  - Extent and relevance of past data
- ✍ Need an *estimation process*