Tutorial: Estimation and Planning Examples

Friday, October 25th

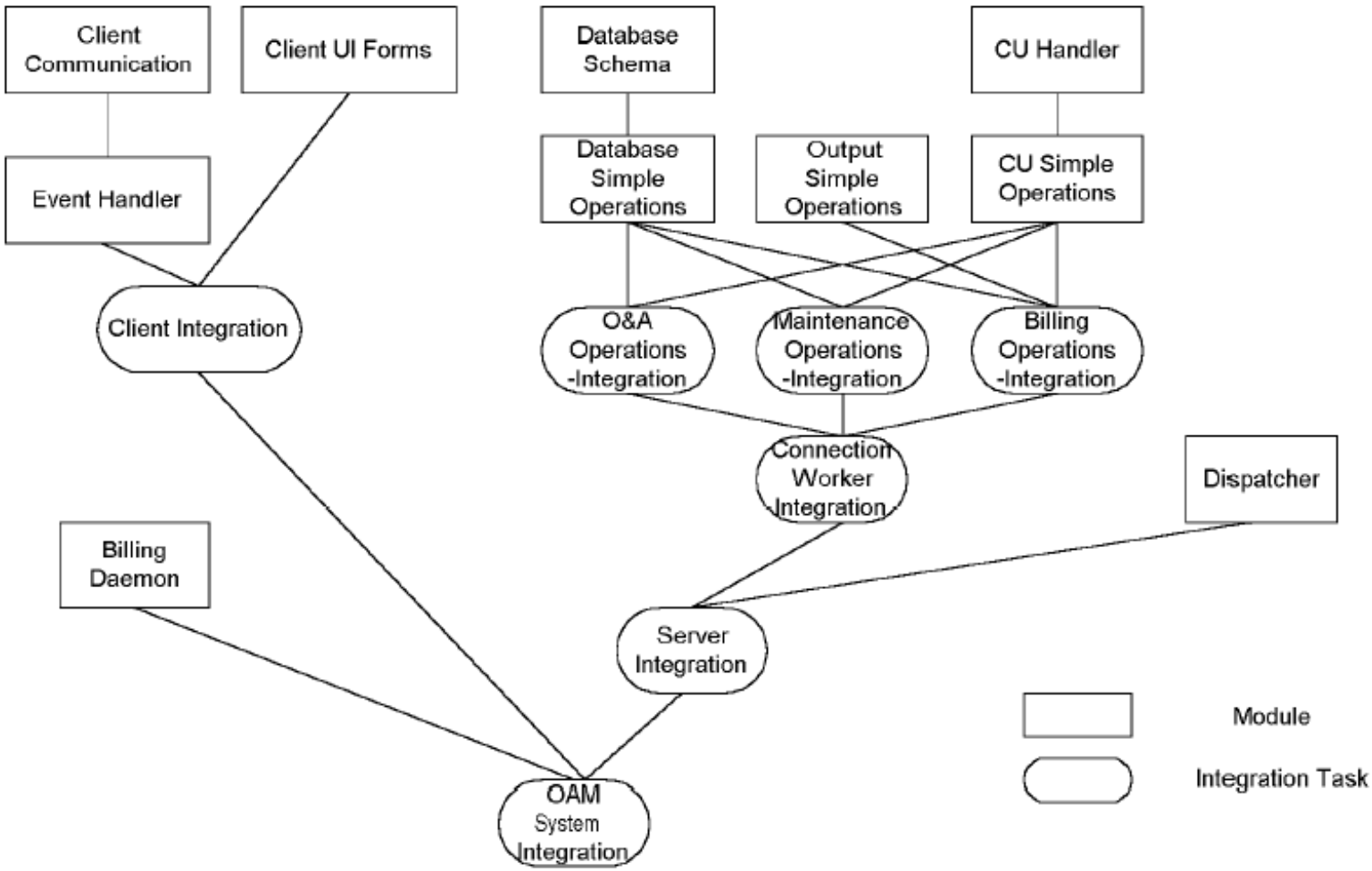
Overview

- Task Integration Example
- Pert/CPM Graph
- Gantt Chart
- COCOMO
- Quick SDL refresher

Task Integration

- Modules taken from design doc
- Integration tasks taken from architecture doc
- Should reinforce architecture chosen
  - Be wary of odd dependencies

Integration Task Diagram



Task Integration Matrix

Integration Task	Client	O&A	Maintenance	Billing	Connection Worker	Server	System
Client Comm.	X						X
Client Event Handler	X						X
UI Forms	X						X
Database Schema		X	X	X	X	X	X
DB Simple Ops		X	X	X	X	X	X
CU Handler		X	X	X	X	X	X
CU Simple Ops		X	X	X	X	X	X
Print Simple Ops				X	X	X	X
O&A C.O.		X			X	X	X
Maintenance C.O.			X		X	X	X
Billing C.O				X	X	X	X

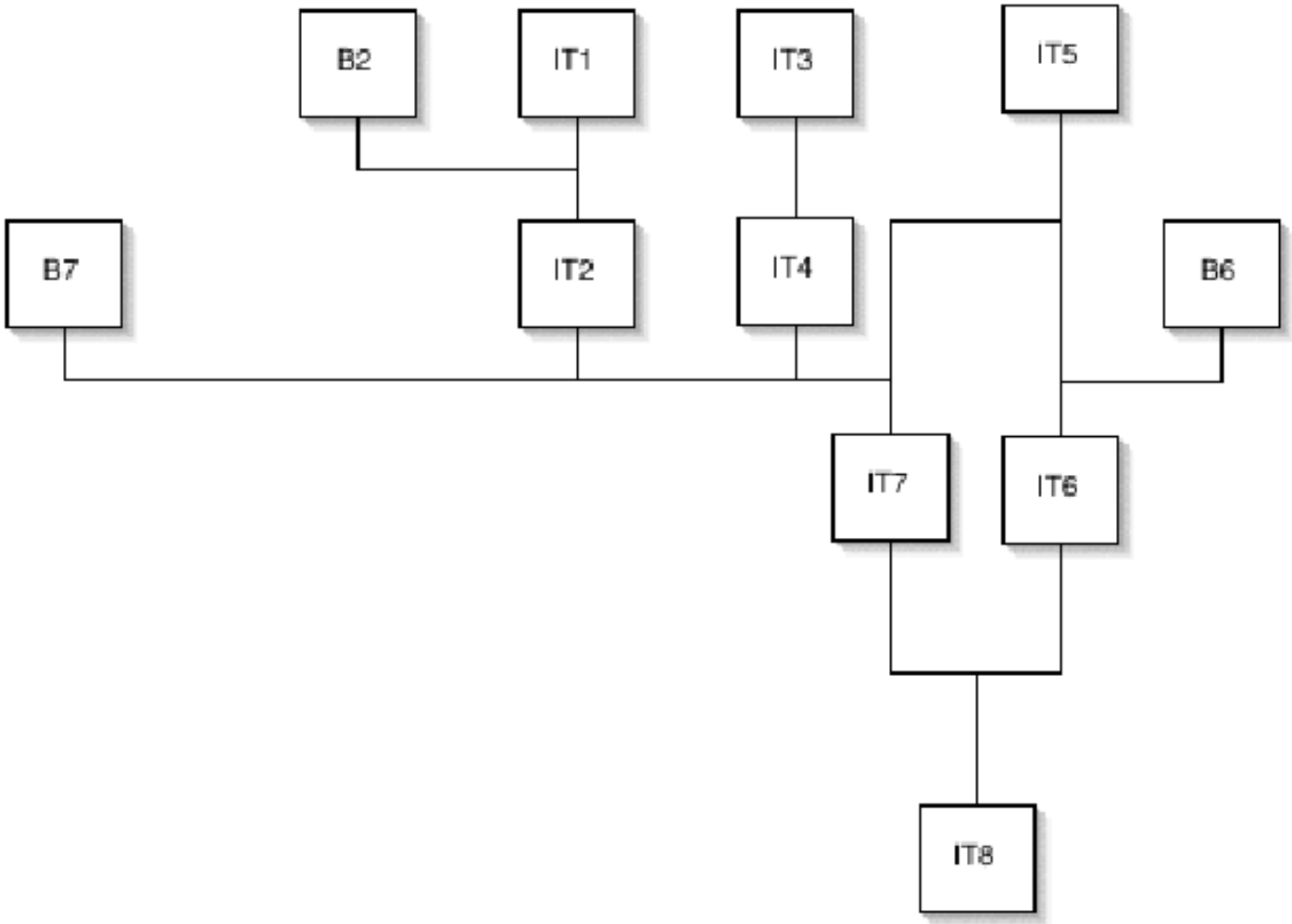
Test Plan Notes

- Give reasons for why test plan was chosen

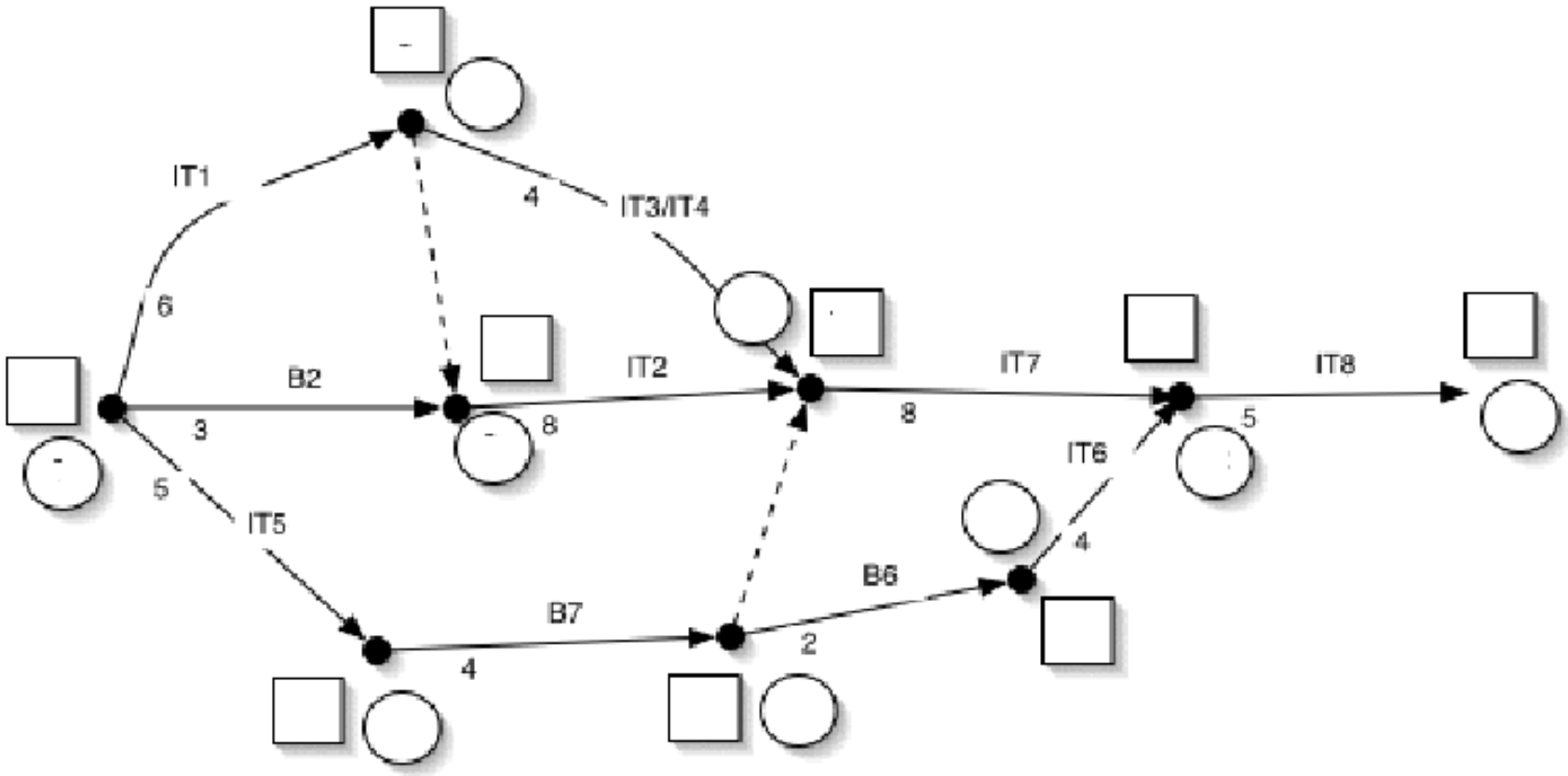
Creating Pert/CPM Graph

- Integration task diagram often shows what tasks can be done concurrently
- In real world setting, priorities not necessarily follow ITG
  - This is why we need schedule priority charts
  - Certain tasks may need to be done by certain people (domain knowledge, etc)

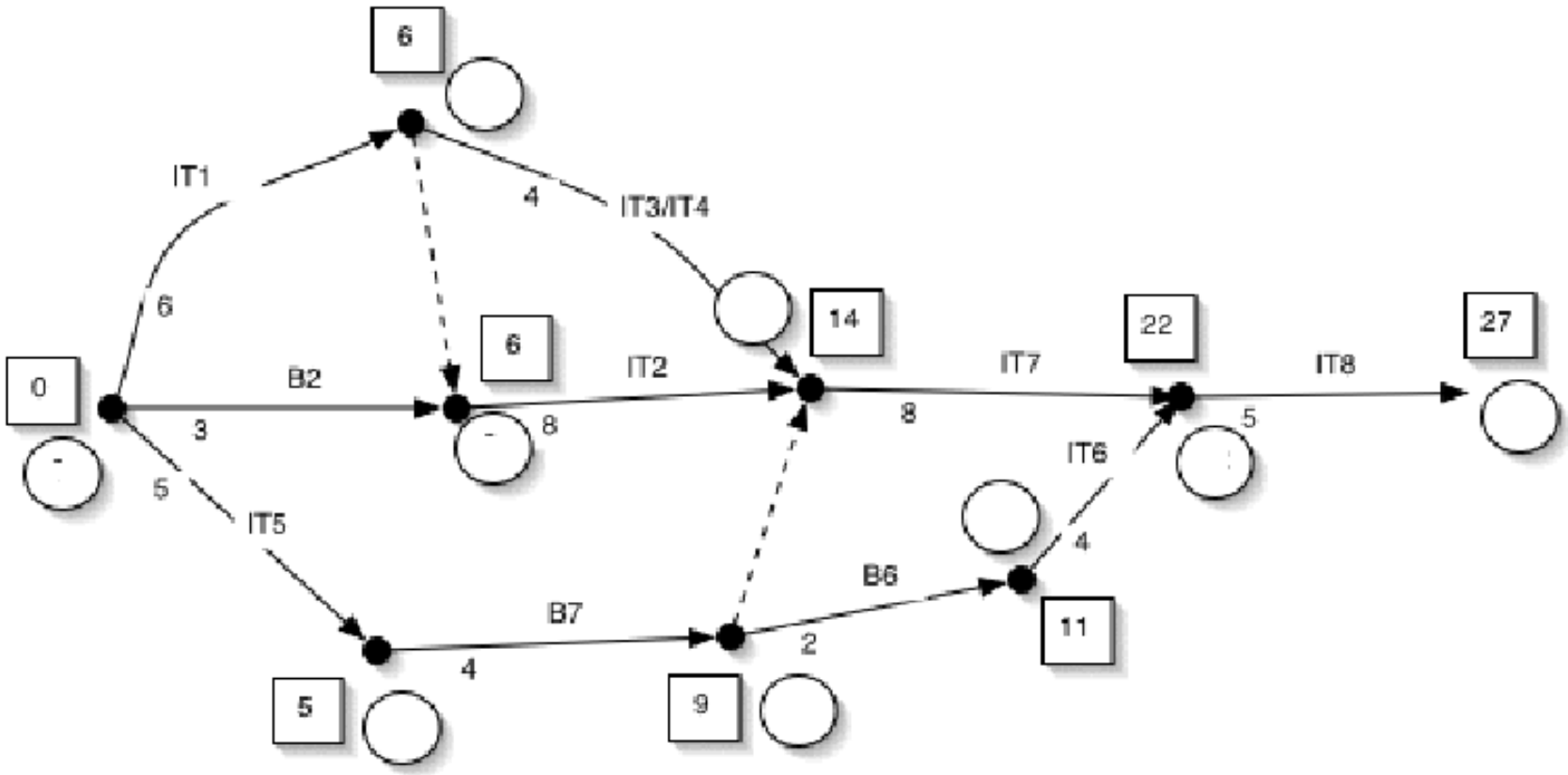
Vaguely Familiar Ground



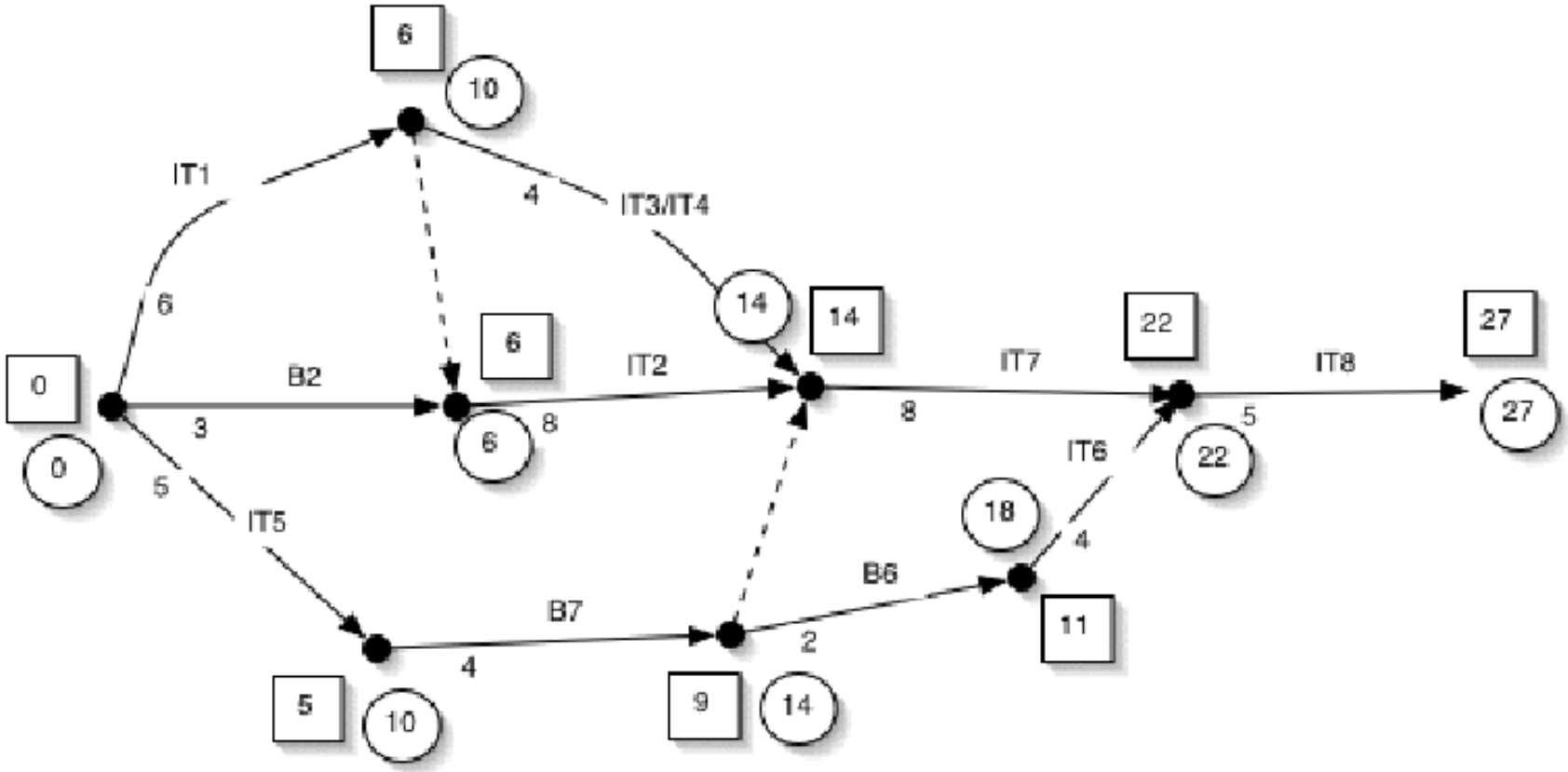
CPM Graph



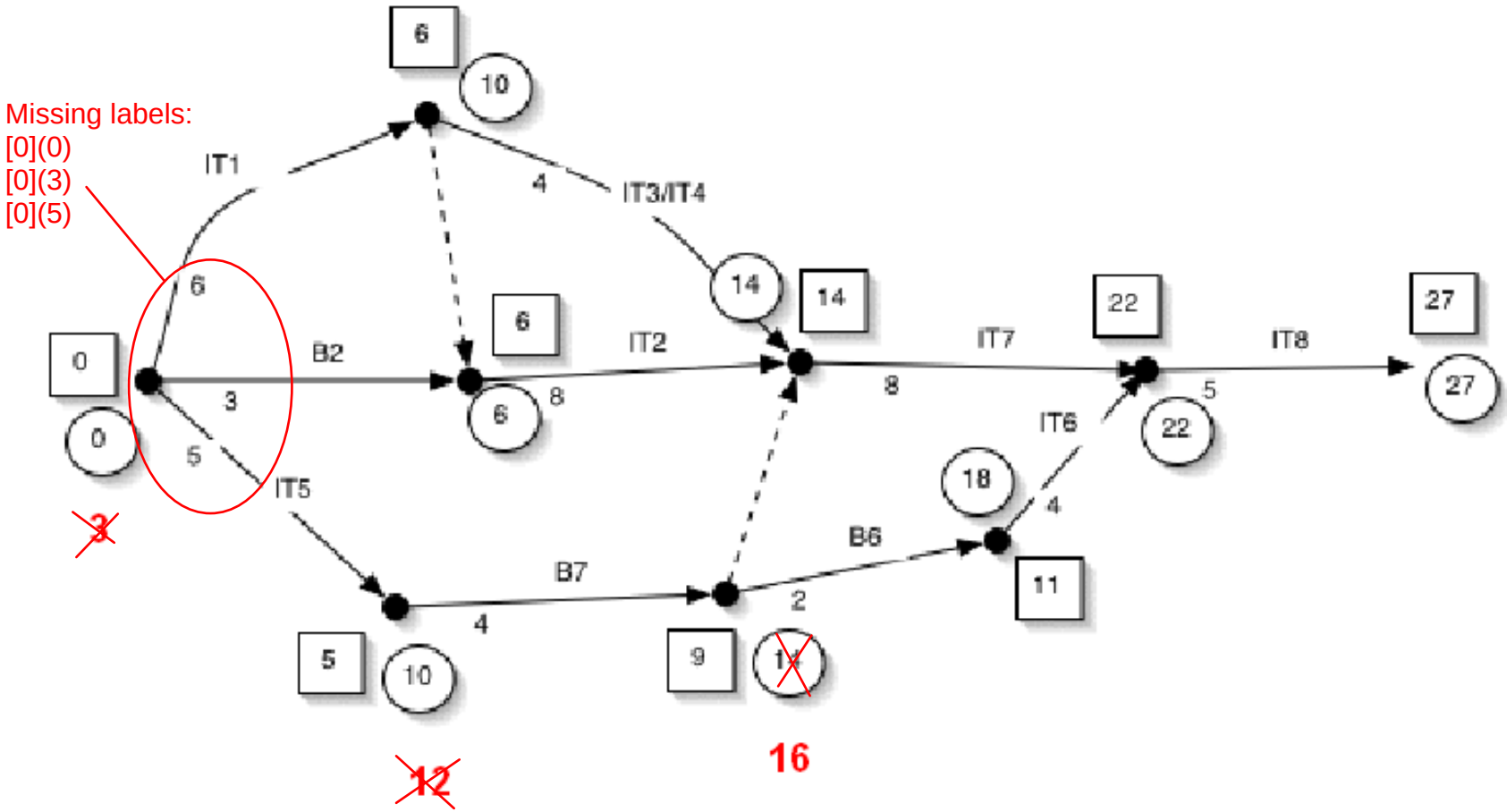
Earliest Completion Times



Latest Completion Times



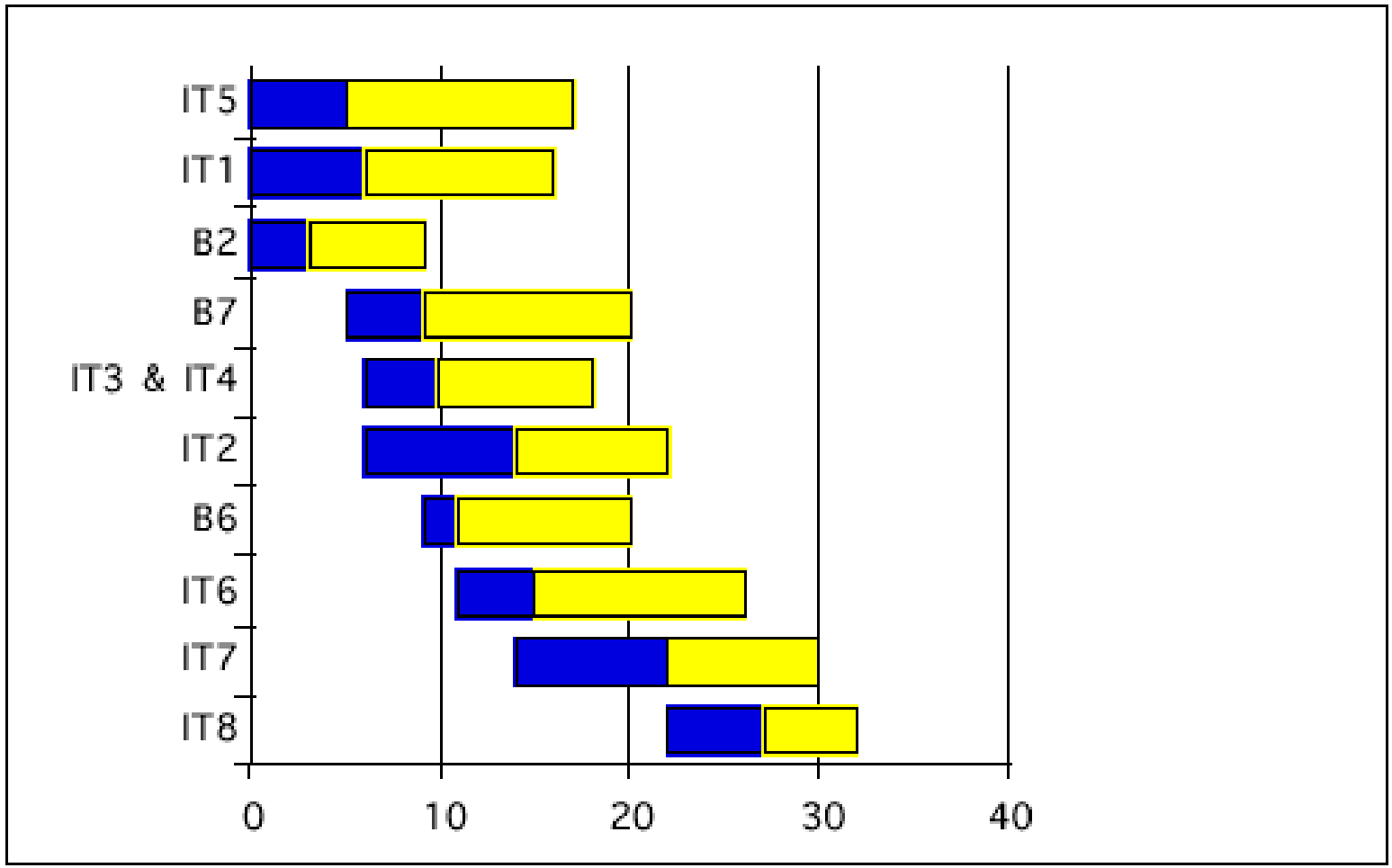
Fixed CPM Graph



Coming up with Gantt Chart

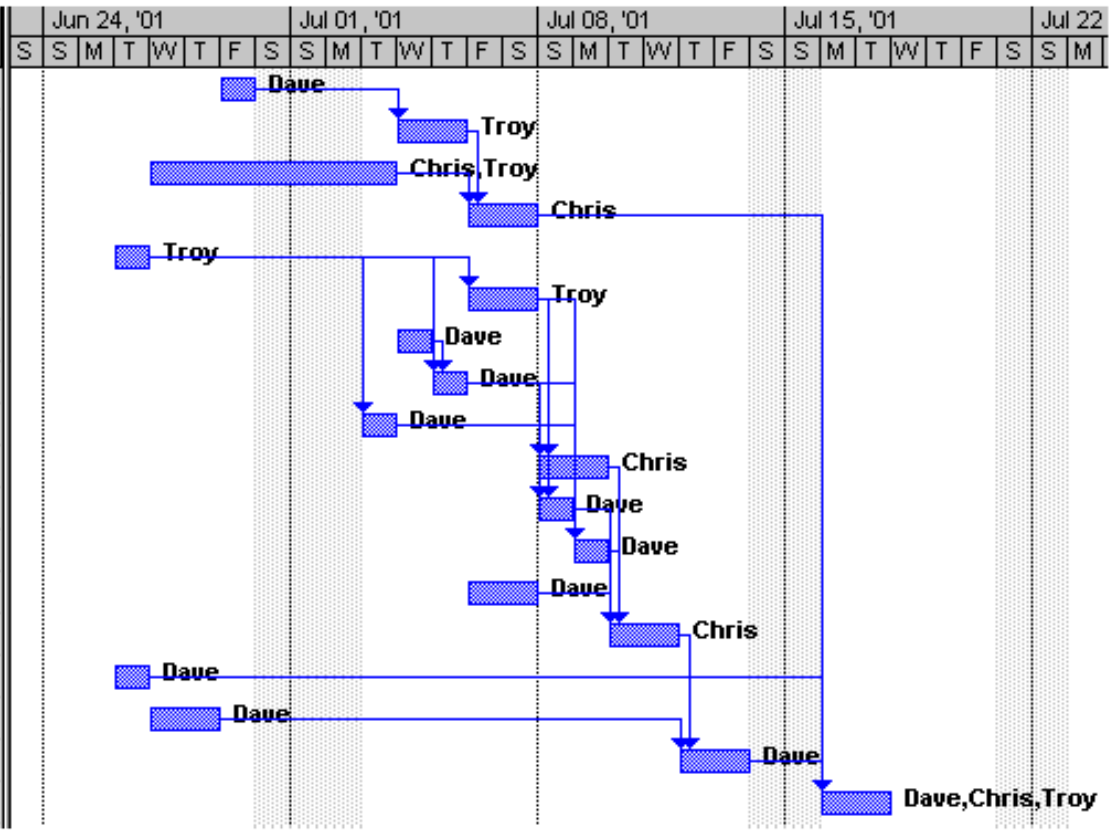
- Try to pick reasonable timelines
  - Usually earliest completion time is chosen
  - Factor in vacations, breaks, etc
- Make note of critical path
  - Helpful to indicate on Gantt chart
- Try to evenly divide up workload

Gantt Chart Example 1



Gantt Chart Example 2

	Task Name	Duration
1	Client Communication	1 day
2	Client Event Handler	2 days
3	UI Forms	4 days
4	Client Integration	2 days
5	Database Schema	1 day
6	Database Simple Operations	2 days
7	CU Handler	1 day
8	CU Simple Operations	1 day
9	Print Simple Operations	1 day
10	O&A Complex Operations	2 days
11	Maintenance Complex Operations	1 day
12	Billing Complex Operations	1 day
13	Connection Worker Process	2 days
14	Connection Worker Integration	2 days
15	Billing Daemon	1 day
16	Dispatcher Process	2 days
17	Server Integration	2 days
18	OAM System Integration	2 days



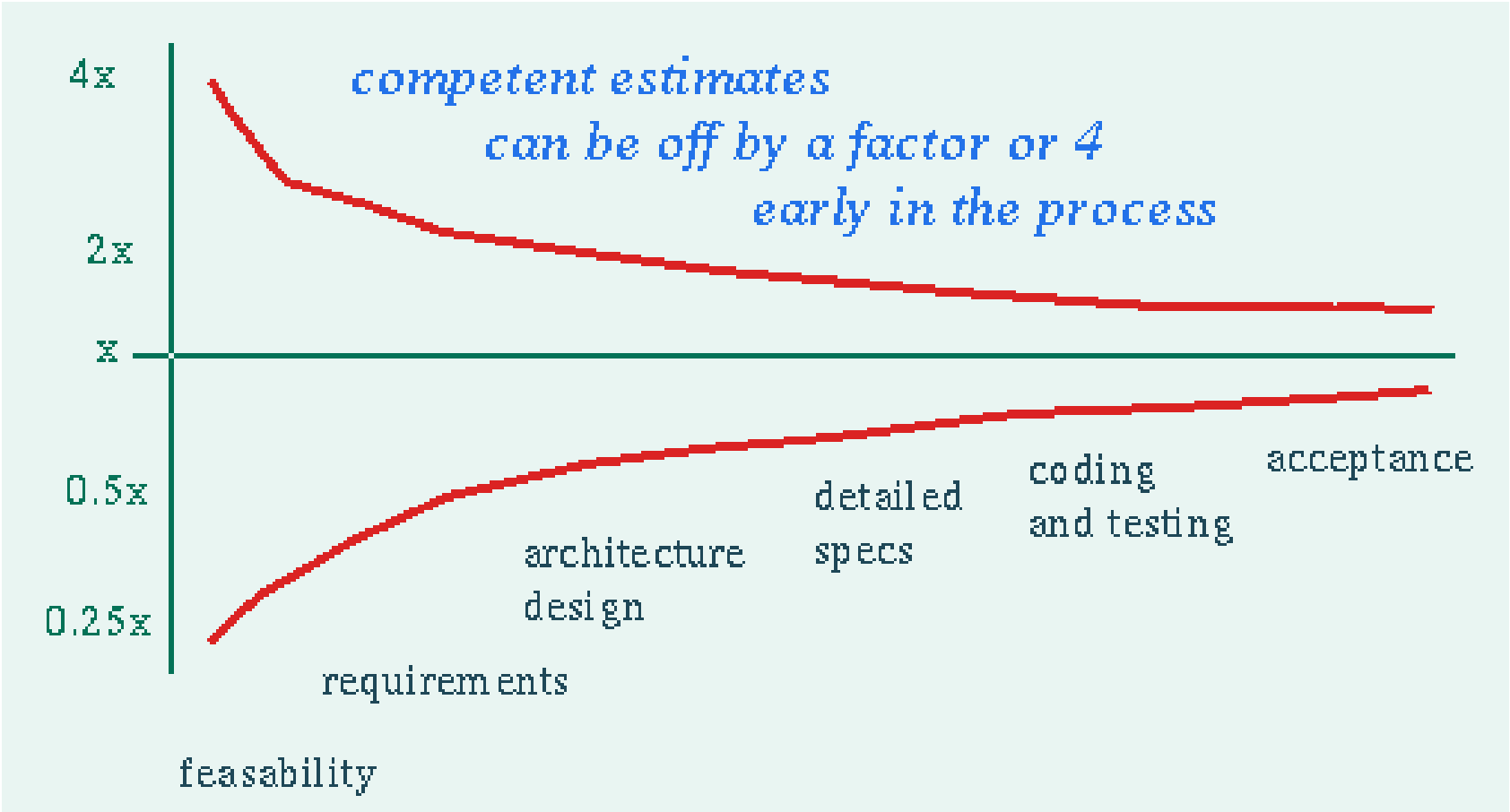
Why are we doing this?

- Poor management is the downfall of many software projects.
- Delivered software may be late, unreliable, cost several times the original estimates and often exhibits poor performance characteristics.
- Software project management is different from other engineering management.
- Product is intangible to a certain extent.
- Most software projects are new and technically innovative. (myth?)
- Good management cannot guarantee project success but bad management usually results in project failure.

COCOMO

- COnstructive COst MOdel
- Created by Barry Boehm in 1981
- Essentially: $EFF = a * SIZE^b$
- Based on a large number of projects from the 70's
- Very simple estimation tool which may or may not work depending on how closely project fits in with original study

Estimating Is Hard To Do



Basic Idea

- Small Projects
  - Small teams (2-3 people)
  - Easy to have mental model
  - Fewer things in the way of completion
  - EFFORT = a * SIZE + b
- Large Projects
  - The more people there are, the harder it becomes
  - EFFORT = a * $SIZE^b$
- a and b are scaling factors

Project Types

- Organic
  - Routine project
  - Well understood domain
  - Team works well and efficiently together
  - Project expected to run smoothly
  - Typically a smaller system

Project Types

- Embedded
  - Difficulties expected
  - Project that is hard (control software for a nuclear plant, or spacecraft)
  - Team has little experience in domain
  - New or inexperienced team
  - Tend to be large projects with lots of constraints

Project Types

- Semi-Detached
  - In the middle
  - Complex system, but something the company is familiar with
  - Teams may be made up of experienced and inexperienced members
  - System not huge, but not small either

What is *a* and *b*?

- Organic
  - Person months = 2.4 * KDSI$^{1.05}$
- Semi-Detached
  - Person months = 3.0 * KDSI$^{1.12}$
- Embedded
  - Person months = 3.6 * KDSI$^{1.20}$

Slightly More Advanced

- Add in an adjustment factor
  - EFFORT = EAF * a * $SIZE^b$
  - Adjustment factors cover wide range of development aspects
  - Factor all adjustment factors together to get EAF

Slightly More Advanced

- The factors *a* and *b* differ
- Organic
  - a = 3.2, b = 1.05
- Semi-Detached
  - a = 3.0, b = 1.12
- Embedded
  - a = 2.8, b = 1.20

Project Characteristics Table

Cost adjustments for computing the EAF (Effort Adjustment Factor)

	v.low	low	nominal	high	v.high	ex.high
product attributes						
required software reliability	0.75	0.88	1.00	1.15	1.40	
database size		0.94	1.00	1.08	1.16	
product complexity	0.70	0.85	1.00	1.15	1.30	1.65
computer attributes						
execution time constraints			1.00	1.11	1.30	1.66
main storage constraints			1.00	1.06	1.21	1.56
virtual machine volitility	0.87	1.00	1.15	1.30		
computer turnaround time		0.87	1.00	1.07	1.15	
personnel attributes						
analyst capability	1.46	1.19	1.00	0.86	0.71	
applications experience	1.29	1.13	1.00	0.91	0.82	
programmer capability	1.42	1.17	1.00	0.86	0.70	
virtual machine experience	1.21	1.10	1.00	0.90		
programming language experience	1.14	1.07	1.00	0.95		
project attributes						
use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
use of software tools	1.24	1.10	1.00	0.91	0.83	
required development schedule	1.23	1.08	1.00	1.04	1.10	

How to use COCOMO

- Simply: Plug and Chug
- What if it doesn’t work?
  - Play with adjustment factors
  - Tweak *a* and *b* to make the equation fit the result
  - Use a more complex metric

Example of Intermediate COCOMO

From section 3.1.5 in Jalote...

Consider a database system needed for an office automation project.
The requirements document shows 4 modules needed.

Sizes are estimated as follows:

data entry	0.6 KDSI
data update	0.6 KDSI
query	0.8 KDSI
report gen	1.0 KDSI
system SIZE	3.0 KDSI

- **How can SIZE be estimated in the first place?**

The project is judged to be ORGANIC (so a=3.2, b=1.05)

The manager rates project details as follows:

characteristic	level	EAF
complexity	high	1.15
storage	high	1.06
experience	low	1.13
programmer capabilities	low	1.17

All others: nominal (1.0)

Person Months for the project:

$$PM = (1.15*1.06*1.13*1.17) * 3.2 * (3.0)^{1.05}$$
$$PM = 1.61 * 3.2 * 3.17$$
$$PM = 16.33$$

> 16 person months estimated

- How many people shoud we hire for this effort?

Duration and Staffing

Once an estimate is obtained for Effort (Person Months), a manager must determine how many persons to put on the job. This will ultimately determine the calendar-time duration of the project.

People and months are not interchangeable.

More people does not mean proportionately less calendar time. More people complicate communications and this complexity translates into a project slowdown.

The model:

Organic: $DURATION = 2.5 * EFFORT^{0.38}$

Semi-Detached: $DURATION = 2.5 * EFFORT^{0.35}$

Embedded: $DURATION = 2.5 * EFFORT^{0.32}$

As before, these equations were derived from observed data.

Example (continued)

◦ (original PM esitmate)

We had 16.33 person-months estimated for the database project.
The project was judged **ORGANIC**

$$DURATION = 2.5 * (16.33)^{0.38}$$
$$DURATION = 2.5 * 2.89$$
$$DURATION = 7.23$$

> 7 months to complete

How many people to hire?

$$\frac{16.33 \text{ person-months}}{7.23 \text{ months}} = 2.26 \text{ persons}$$

2 or 3 team members needed

Estimating SIZE

◦ other methods

FUNCTION POINTS is one method that has been researched, for example. Like COCOMO, the methods of Function Points has been developed from empirical evidence. Many projects were examined with respect to the characteristics discussed here, and the sizes of the final products examined, and a model produced to fit the data.

Functions Points methods works best with business-type information processing applications, since those are the projects that were examined to produce the model. It is also not as widely known, used, or trusted for estimating SIZE as COCOMO is for estimating EFFORT. We give it here just as an example.

5 items determine the ultimate complexity of an application... take the weighted sum of these counts to get the # of function points in a system.

characteristic	weight
number of inputs	4
number of outputs	5
number of inquiries	4
number of files	10
number of interfaces	7

inputs, outputs: groups count as one item (screenful, file)
inquiries: interactive "cases" or transactions a system must be able to deal with
interfaces: hook-up to other systems (OS, payroll system, company DB, etc.)

Once the # function points FP is calculated (from requirements), other data is used to estimate module/system SIZE. For various languages, data was obtained on how many lines of code are usually needed to implement one function point...

For example, let's say Ada programs tend to require 70 DSI per FP.

A system with FP=124 would have SIZE = FP * 70 = 124*70 = 8680

SO, the estimate for SIZE in this case is 8.68 KDSI

In Reality

- Estimating comes with experience
- Using something like Function Points to come up with code size doesn’t really work
  - (personal belief)
- COCOMO is too simple and too old to really be of use
  - Good starting point, more advanced models available
  - Tweaking formulas might yield good results

Quick Look At SDL

Credits

- Lots of COCOMO info taken from
  - http://www.cs.unc.edu/~stotts/COMP145/cocomo.html
- CPM slide “Why Are We Doing This” taken from another presentation (can’t find URL)