

General Approach to Estimation

1. Estimate size of project
2. Convert from size to staff time
3. Determine resource availability
4. Convert to calendar time
5. Track and refine

1. Approaches to Size Estimation

- Based on project characteristics
 - COCOMO
- Based on requirements
 - Function Points
- Based on conceptual architecture
 - PROBE

Other Estimation Techniques

- Brooks' Effort Matrix

	Program	Product
Application	1	x3
System	x3	x9

- Fantasy factor (x1.3)
 - Estimates routinely underestimate actuals

COCOMO

- Based on project characteristics
- COnstructive COst MOdel
 - Barry W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981
 - http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html
- Project data interpolation
- Cost factors

COCOMO - Continued

- Large database of product histories
- *Basic, intermediate, advanced* versions
- Cost factors
- Organic, semi-detached, embedded projects
- Estimate costs within 20%, 70% of the time
 - Precision and reliability

BASIC COCOMO

- $SM = \text{staff months} = 152 \text{ hours} = 2.4 * KDSI^{1.05}$
 - $KDSI = 1000$ delivered lines of source instructions
 - 3.2 staff months for first 1000 lines
 - Rises faster than linearly
- $TDEV = \text{development time in calendar months} = 2.5 * SM^{0.38}$
 - 2.6 months for first 1000 lines
 - Maximum compressibility

Example Cost Drivers

- Product Factors
 - Reliability (RELY)
 - Data (DATA)
 - Complexity (CPLX)
 - Reusability (RUSE)
 - Documentation (DOCU)
- Platform Factors
 - Time constraint (TIME)
 - Storage constraint (STOR)
 - Platform volatility (PVOL)
- Personnel factors
 - Analyst capability (ACAP)
 - Program capability (PCAP)
 - Applications experience (APEX)
 - Platform experience (PLEX)
 - Language and tool experience (LTEX)
 - Personnel continuity (PCON)
- Project Factors
 - Software tools (TOOL)
 - Multisite development (SITE)
 - Required schedule (SCED)

Function Points

- *Function points are a measure of the size of computer applications and the projects that build them. The size is measured from a functional, or user, point of view. It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.* - Function Point FAQ
 - <http://www.ifpug.org/>
 - Albrecht, IBM GUIDE organization, late 1970's

FP Overview

- Based on requirements document
- Count of external inputs, external outputs, external queries, internal logic files, external interface files
- Low / average / high gradations; table lookup
- Weighted combination (4, 5, 4, 10, 7)
- Adjustment based on system characteristics
- Conversion to LOC or time required

PROBE

- Based on conceptual design
- PRoxy-Based Estimation
 - Use classes or functions as proxies
- Part of Personal Software Process (PSP) / Team Software Process (TSP) - SEI
 - Watts Humphrey, 1995

Steps

1. Produce conceptual design document
2. Identify objects
 - Methods, object type, relative size, reuse category (base, reused-from-library, new, new-to-be-reused)
3. Compare unit with historical database
4. Estimate relative sizes
 - Very large, large, medium, small, very small
 - Based on number of standard deviations from mean
5. Combine estimates
 - Linear regression (Program size = β_0 + Object size * β_1)

Selecting a Proxy

- Proxy size should relate to effort
- Automatically countable
- Early project visualization
- Customizable to project
- Sensitive to variations

Possible Proxies

- Function points
- Objects (and their methods)
- Screens
- Files
- Scripts
- Document chapters
- Features
- Functions

Conceptual Design

- Early-stage solution breakdown into components for purposes of estimation
 - *Component* = object, function, etc.
 - Not used for actual implementation
- Techniques (top down)
 - Noun-verb analysis
 - Use cases
- Result is a set of units small enough to use for computing overall program size

Example Historical Data Table

(From Humphrey, *A Discipline of Software Engineering*)

C++ - Lines of code per method

Category	Very small	Small	Medium	Large	Very Large
Calculation	2.34	5.13	11.25	24.66	54.05
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

2., 3., 4. Scheduling

- Split project into tasks hierarchically
 - Define each task
 - Estimate time and resources required for each
- Inventory available resources
- Organize tasks concurrently to make optimal use of resources
 - Minimize task dependencies to avoid delays
 - Level work loads
 - Find Critical Path

Task Planning Template

- Name (verb or verb phrase)
- Description (paragraph)
- Entrance criteria
 - Activity inputs and how you know that they have sufficient quality
- Exit criteria
 - Deliverable document description and how you know it has been completed satisfactorily
- Estimates
 - Effort hours - total number of team-member hours spent on the activity
 - Lines of code/text - total number of lines of source code in your product
 - Defects - total number of defects you will find while testing your code

Problems with Scheduling

- Scheduling is difficult
- Failure to include all activities
- Productivity is not proportional to the number of people
 - Bigger projects require more communications per person
- Adding people to a late project makes it later
- The unexpected always happens
 - Risk planning

Scheduling Mechanisms

- Process modeling
 - Data flow diagram
 - Process programming language
- Work Breakdowns
 - Tasks and milestones
- PERT Charts - Critical Path
- Gantt Charts - responsibilities

Visual Process Programming

(Little-JIL)

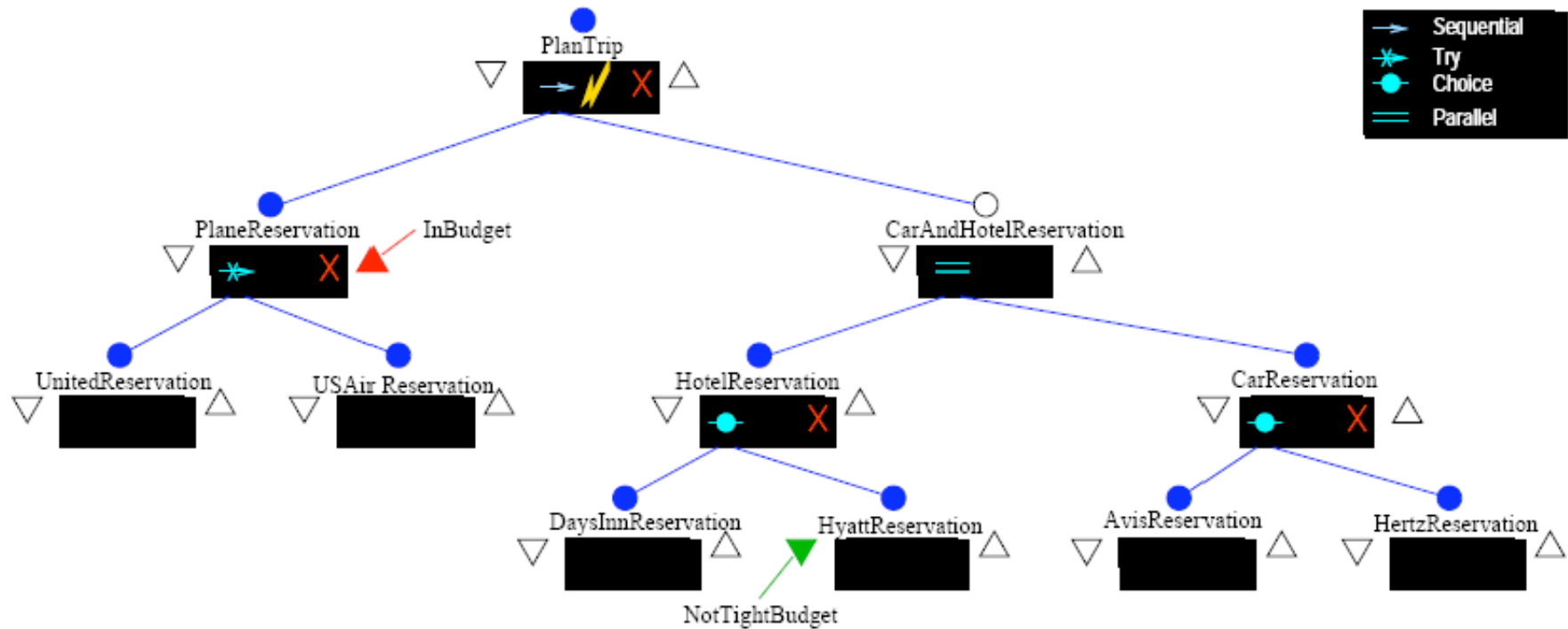
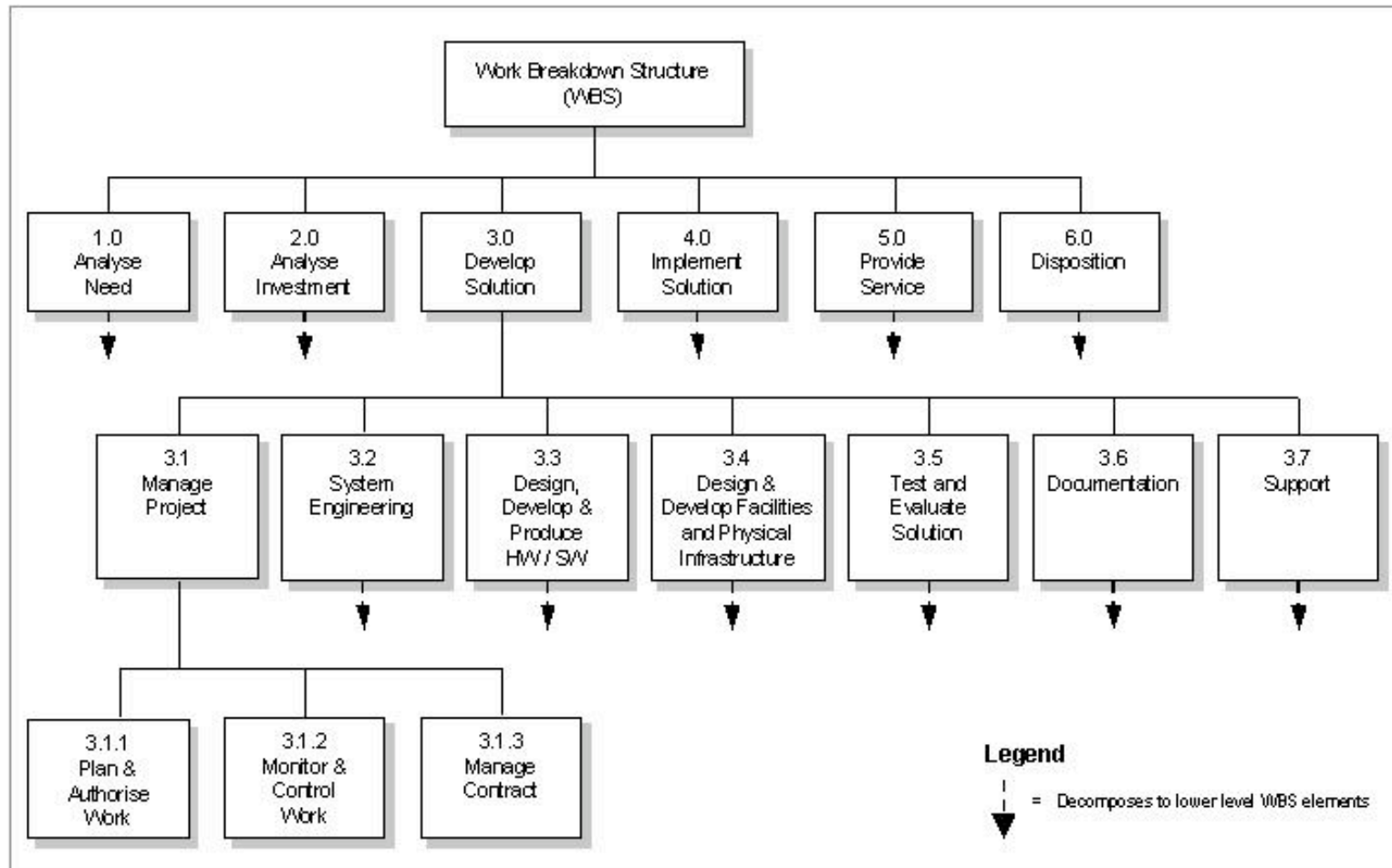


Figure 2. Reservation process showing proactive control: step kinds, requisites.

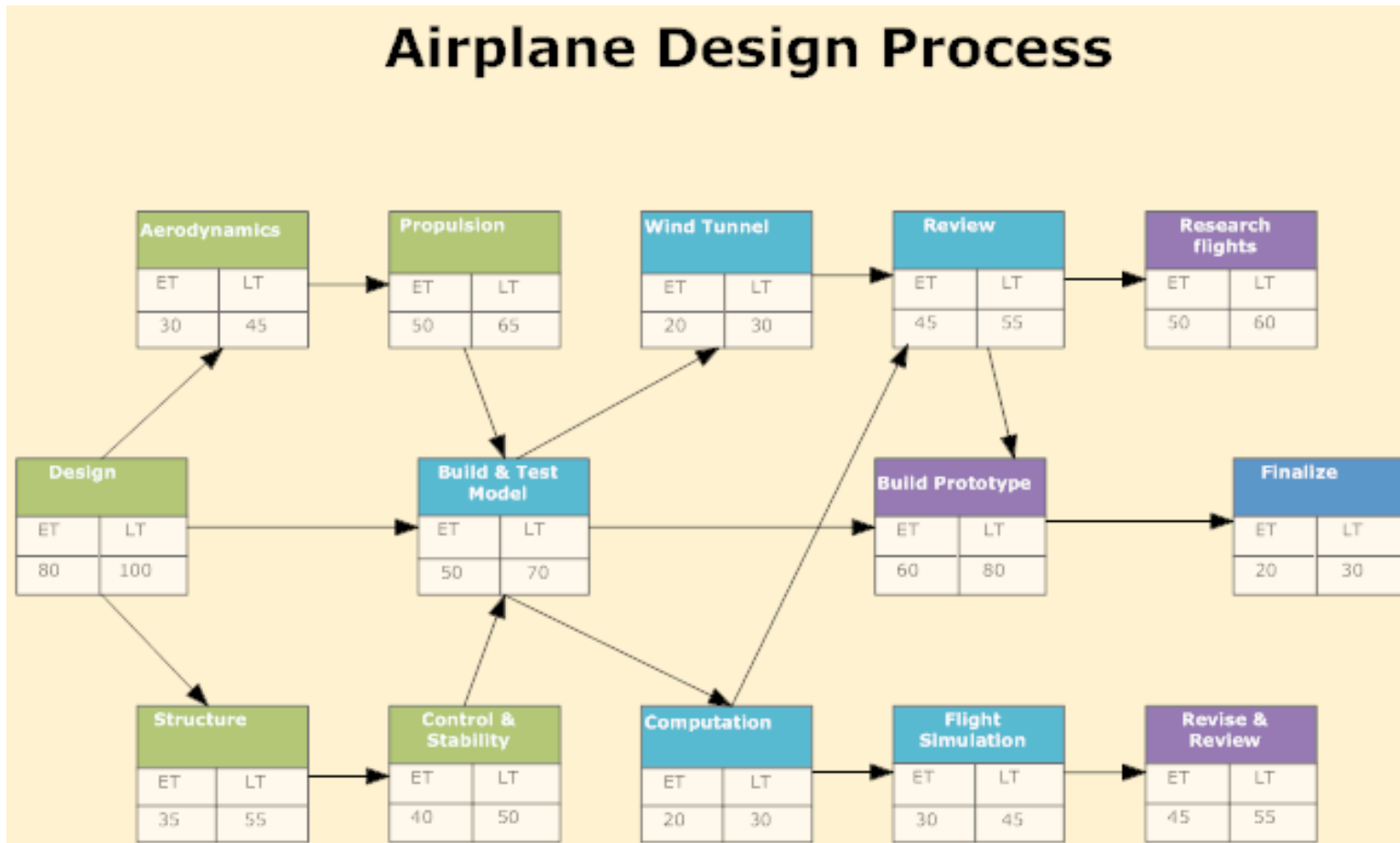
Work Breakdown Structure

(Protracq)



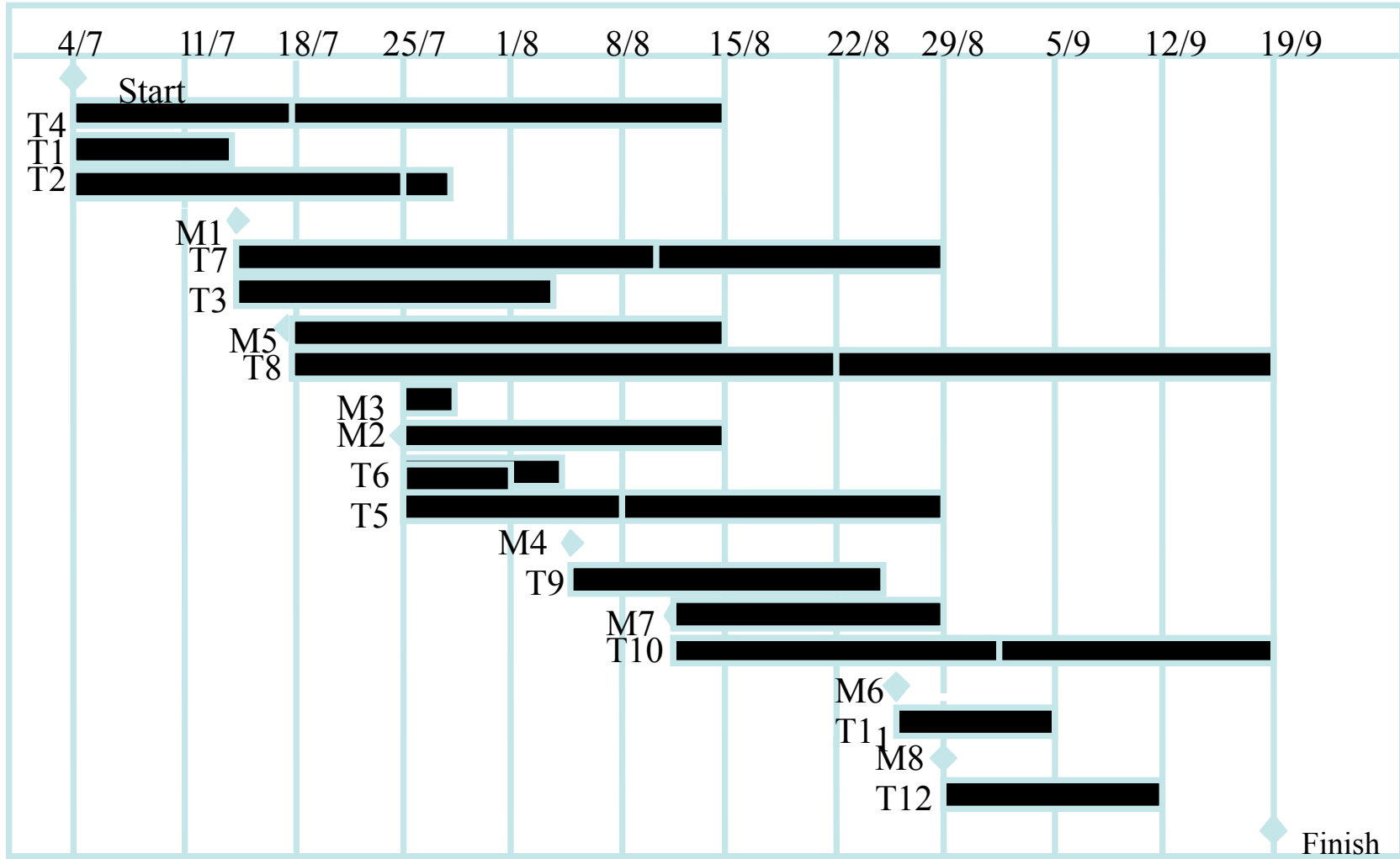
PERT Chart

(NOWECO)



Gantt Chart

Sommerville, 1995



Time Breakdown for Software Developers

- Productive individual work 30%
- Communicating 50%
- Non-productive 20%

Time Breakdown for Small Projects

- Architecture/Design 10%
- Detailed Design 20%
- Code/Debug 25%
- Unit Testing 20%
- Integration 15%
- System Test 10%

Observations

- Estimation and scheduling are notoriously difficult and error prone
 - At beginning of project, you only have a fuzzy idea of problem, therefore your estimates of time and effort will be fuzzy too
 - If this is the first time the team has worked together, you may also only have a limited idea of its capabilities and productivity
- Gradual refinement
 - Only as the project develops and the problem and solution become clearer, will the estimates increase in accuracy
 - This implies that tracking and review are essential