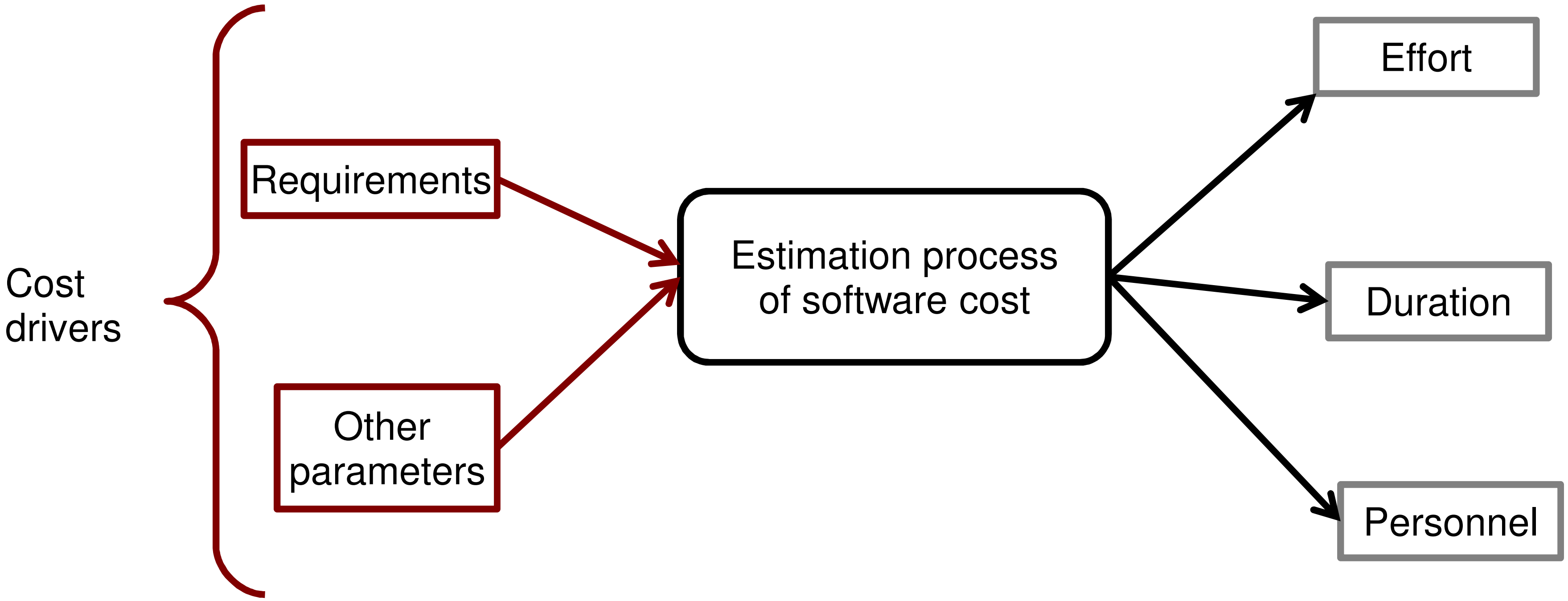
Project estimation

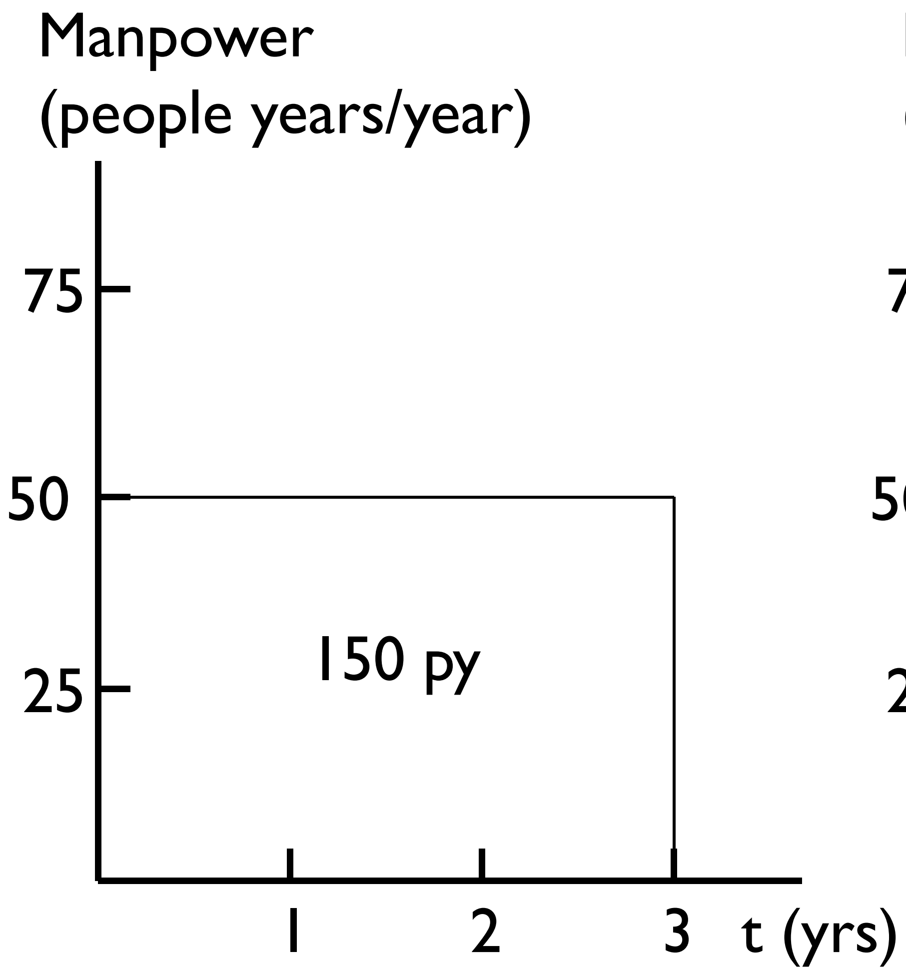
What are we interested in?

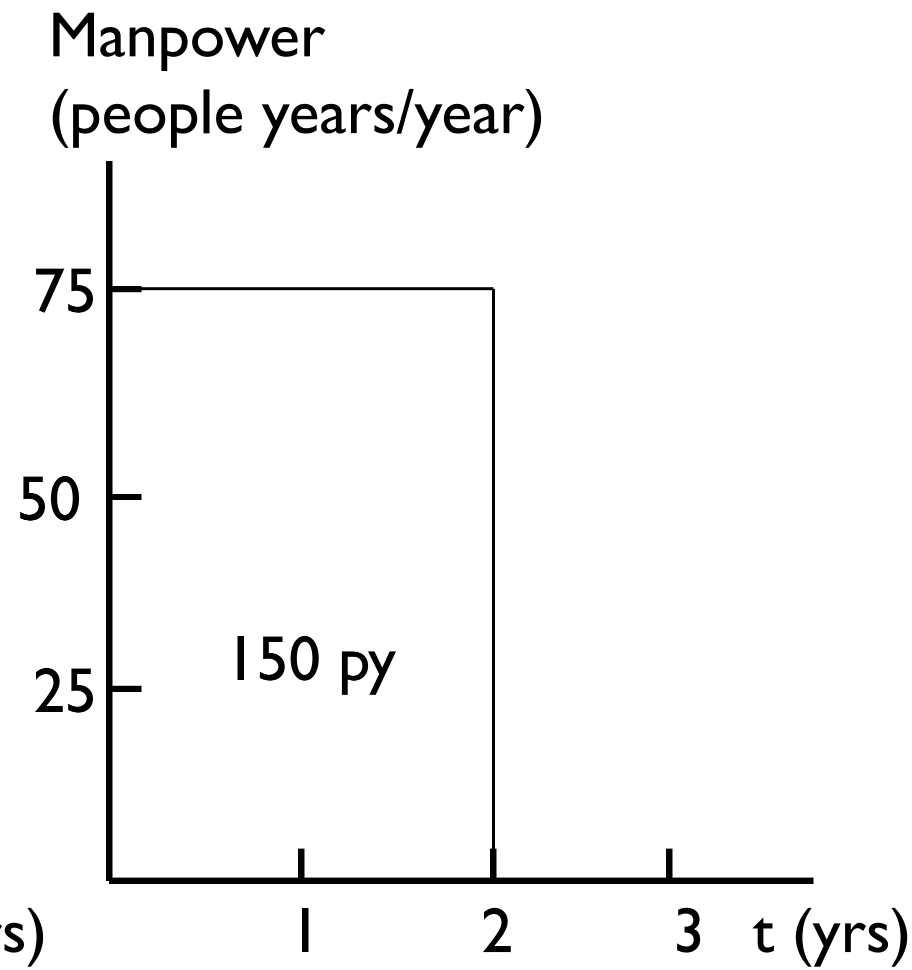
- We aim at estimating the time needed for the development of the software
  - Not only the "coding" time, but also the time needed for the other phases
  - The cost is proportional to the time of the development
- In particular, we are interested in the **delivery** time
  - The time needed to delivery the software to the customer
  - Represented by t_d

What are we interested in?



The software (false) myth





Estimation kinds of techniques

- Algorithmic cost modeling
- The “guru” approach
- Estimation by analogy
- Parkinson’s law
- Pricing to win

Algorithmic cost modeling

- The estimation is calculated by an **algorithm** that considers
  - Historical costs
  - Size of the project
  - Type of the project
  - Company's features
- Pro
  - **Accurate**
  - **Independent** of contingency
- Cons
  - Requires a **lot** of information
  - **Complex** computation

The “guru” approach

- The **expert** (aka “guru”) is in charge of estimating the time and cost of development
  - Often the chief of the company, or the chief analyst
- Estimation defined on the base of:
  - Experience
  - Company way of work
  - Intuition
- Pros
  - **Cheap**
  - Accurate if **true** expert
- Cons
  - **Empirical** estimation
  - Very inaccurate if **no** experts

Estimation by analogy

- The cost of a project is estimated by **comparing** it to similar projects
- Pros
  - **Accurate** if similar project data are available
- Cons
  - Impossible if **no** comparable projects are available
  - The data must be kept **updated**

Parkinson's law

- The project costs **whatever resources** are available
- Pros
  - **No** overspend
- Cons
  - System is usually **unfinished**

Pricing to win

- The project costs whatever the **customer** has to spend on it
- Pros
  - The contract is **granted**
- Cons
  - The customer is likely to **NOT** have the desired system
  - Costs do **not** accurately reflect the required work

Which technique?

- The **algorithmic** technique has some **advantages** over the others:
  - More reliable
  - Not bound to a single person
  - Not bound to contingency (resource available, customer money)
- They are **still estimation**, so
  - They still rely on
    - Personal experience
    - Way of work
    - Personal skills
  - They are **not** exact

Which technique? (2)

- BUT: if there are no other information, "price to win" can be the **only** technique to apply
- In fact, the other techniques require "internal" information
  - Historical costs
  - Size of the project
  - Type of the project
  - Company's resource
  - Other projects' cost

Top-down vs. Bottom-up

- Top-down
- Start at the **system level** and assess the overall system functionality and how this is delivered through sub-systems
- Pros
  - No detailed knowledge is required
  - Considers also integration, configuration and documentation costs
- Cons
  - Can underestimate costs of low-level issues

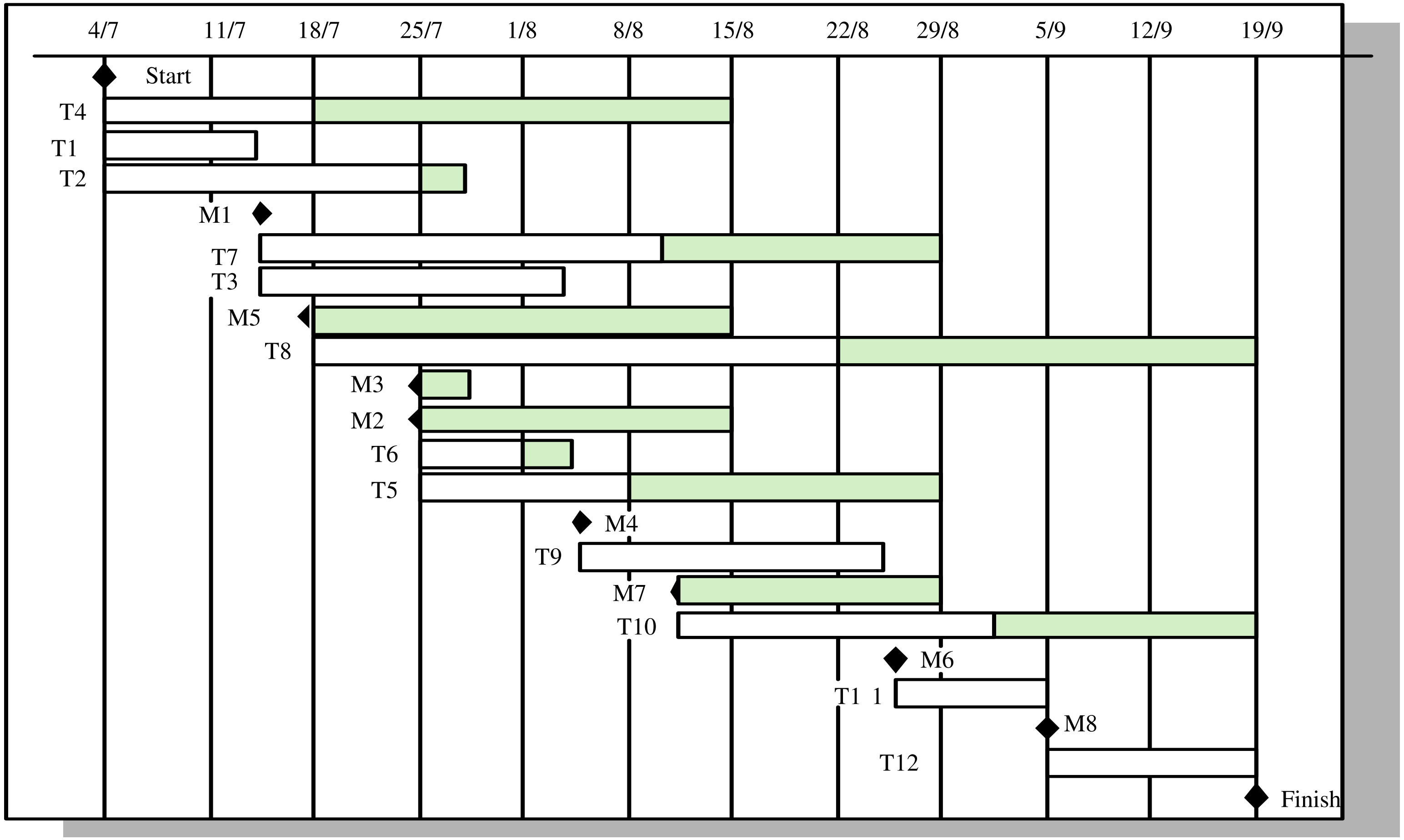
Top-down vs. Bottom-up

- Bottom-up
- Start at the **component level** and estimate the effort required for each component; add these efforts to reach a global estimation
- Pros
  - Accurate if the system has been designed in detail
- Cons
  - May underestimate system-level costs such as integration and documentation

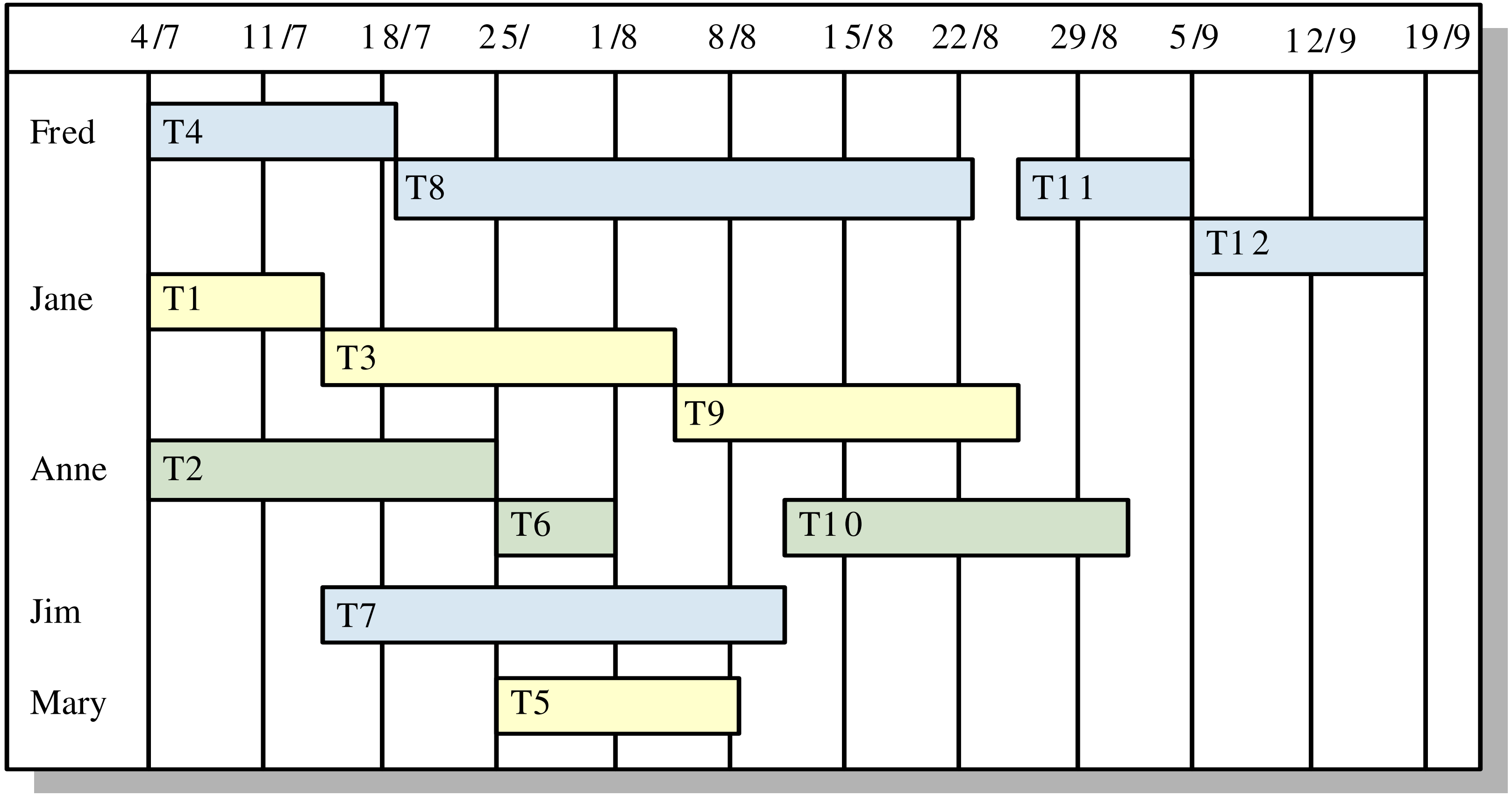
What we aim to achieve

- Feasibility study
- GANNT chart
- Person-month chart
- Estimation of maximum development time
- Estimation of the costs and time of delivery
- Peaks of work
- Kind of skills required

Example of Gannt chart



Example of person-month chart



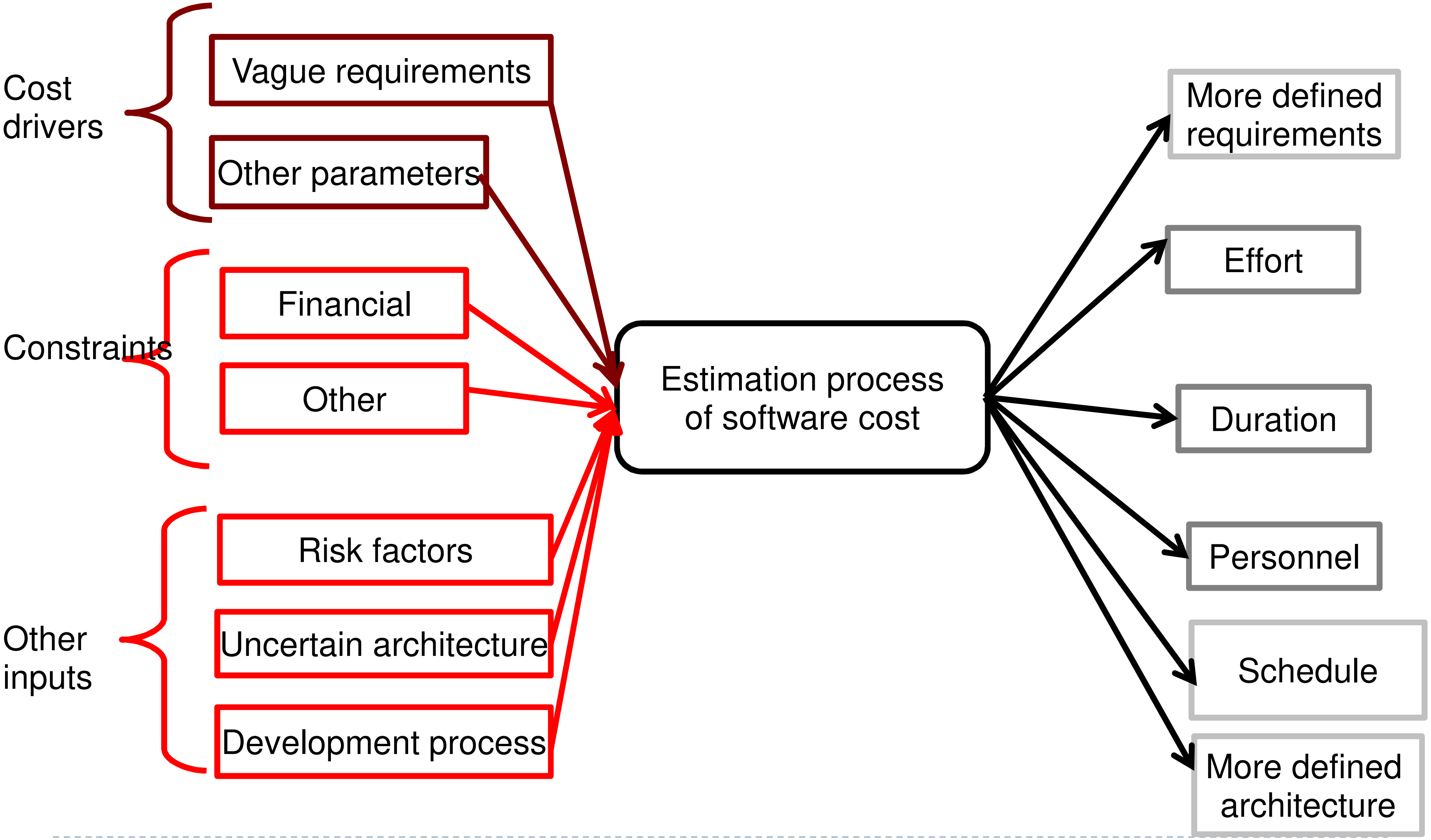
What we aim to achieve (2)

- A means to estimate the critical issues
  - Critical Path Method
- A diagram of the activities
  - PERT
- Estimation of the needed people and resources
  - Per activity
  - Per week
  - Load diagrams
    - Load of people work
    - Peaks
    - It is supposed to be parabolic

Estimation of post-development activities

- Maintenance (around 15% - 20% of the cost)
  - Corrective maintenance
    - bugfix
  - Adaptive maintenance
    - New environment requirements
  - Perfective maintenance
    - New features (user requirements)
- Technical assistance
  - Helpdesk
- Negotiation of a Service Level Agreement (SLA)
  - Temporal limits
  - Kinds of failure

A more complex model



Combining more approaches

- Estimation should be based on several methods
  - Possibly **different** methods
- So to confirm the cost from different ways
- If they do not return (approximately) the same result, available information is not enough
  - Price to win is still available

Adjusting parameters

- An estimation method is useful **not only** to estimate the time to delivery
- It can be exploited also to see how the required effort **changes** depending on the single parameters
  - And to adjust them
- E.g.: what changes if I add a personnel unit in the project?

Metrics

Metrics

- The cost of the development depends on the complexity of the software to be developed
- How to measure the complexity of the software?
- *You cannot control what you cannot measure*, Tom De Marco
- Different metrics have been proposed
  - Lines of Code (LoC)
  - Token Metrics
  - Function points
  - Code complexity

LoC

- There is no precise definition of what a line of code is
  - With or without comments
  - Multiple instructions in the same line
  - Single instruction in different lines
- Non-Commenting Source Statement (NCSS or NCLC)
- Simple to calculate
- Depending on the language
  - The LoC of a Prolog program cannot be compared to the LoC of a C program
- Available only **after** the implementation
- Currently, the most exploited metrics

Token Metrics

- Halsted has proposed to count the number of token, not lines
- A token can be either an operand or an operator
- Different kinds of lines are counted in different way

Function Point

- In ‘70, Albrecht proposed the Function point method
- The idea is to compute the software complexity, rather than its size
- This method is based on the decomposition of the software in smaller parts
  - Easier to estimate
- Considers the **functional** requirements
- Estimation based on different factors

Function Point factors

- Inputs
  - Unique input data user types
- Outputs
  - Unique output data user types
- Inquiries
  - Unique input/output data user types
- Files
  - File types used and shared
- Interfaces
  - Logical groups of information entered or output

Function Point computation

- The values of the factors are modified on the base of different influencing parameters
  - Mainly related to the environment
- The resulting function point value is calculated as:
- Fp= a*inputs+b*outputs+c*inquiries+d*files+e*interfaces
- From Fp, an estimation of the size S of the code to be produced, in LOC/NCSS, is derived
- For instance, in COBOL:
- Cobol S=118.7*Fp-6490

Code complexity

- Different approaches that evaluate the intrinsic complexity of the code
- The most famous one is the Mccabe Cyclomatic Complexity Metrics
  - It measures the number of independent paths inside the code

CoCoMo

CoCoMo

- Constructive Cost Model
- Invented by Barry W. Bohm (1981)
- It is a static method
- Based on the analysis of several processes in several languages
- Three kinds of projects: Organic, Semi-Detached and Embedded
- Three models: Basic, Intermediate, Detailed

Organic projects

- Limited code size
- Small and close-knit team
- Known problem
- Flexible requirements

Semi-detached projects

- Middle-size team
- Involved developers with different experiences
- Mix of flexible and rigid requirements

Embedded projects

- Large-size team
- Many and rigid constraints
- Many competences required
  - Some not in the team
- Innovative project

Estimation

- Based on the American development model
  - Oversized with respect to the European software companies
- Estimation of some values
  - K_m = cost of the project in person-month (4 weeks per month)
  - t_d = time to deliver, in months
- Only one input parameter: S_k = size in kNCSS

Formulas – basic model

Kind of project	Person-months	Time to deliver
Organic	$K_m=(2.4\times S_k)^{1.05}$	$t_d=(2.5\times K_m)^{0.38}$
Semi-detached	$K_m=(3.0\times S_k)^{1.12}$	$t_d=(2.5\times K_m)^{0.35}$
Embedded	$K_m=(3.6\times S_k)^{1.20}$	$t_d=(2.5\times K_m)^{0.32}$

Costs

- To calculate the costs, we must multiply the number of the needed person-months by the cost of a developer for a month
- In Italy, a reasonable cost of a developer is 5000 € per month

Example – 40 kNCSS

Kind of project	Person-months	Time to deliver
Organic	120 pM (=600k€)	8.7 M
Semi-detached	213 pM (=1.065M€)	9 M
Embedded	389 pM (=1.945M€)	9 M

Example – 80 kNCSS

Kind of project	Person-months	Time to deliver
Organic	249 pM (=1.248M€)	11.5 M
Semi-detached	463 pM (=2.316M€)	11.8 M
Embedded	893 pM (=4.469M€)	11.8 M

Pros and cons of the basic model

- Pro
  - Good for a quick estimation
- Cons
  - Does not consider many factors that can influence the development

Intermediate model

- Considers more factors specific of the project
  - Product requirements
  - Hardware requirements
  - Involved personnel
  - Development way
- The cost of the project is modified according to these factors

Intermediate model – product requirements

- They are based on 15 attributes
  - Called cost drivers
  - Subjective
- Each attribute can range from "Very low" to "Very high"
- Each factor provide a coefficient between 0.75 and 1.66

- First, a nominal cost K_n is estimated (see next slide)
- Then, the estimation of the number of month is calculated multiplying K_n for the product of the cost drivers: $K_m = K_n \prod c_i$

Formulas – intermediate model

Kind of project	Person-months	Time to deliver
Organic	$K_n = (3.2 \times S_k)^{1.05}$	$t_d = (2.5 \times K_m)^{0.38}$
Semi-detached	$K_n = (3.0 \times S_k)^{1.12}$	$t_d = (2.5 \times K_m)^{0.35}$
Embedded	$K_n = (2.8 \times S_k)^{1.20}$	$t_d = (2.5 \times K_m)^{0.32}$

$$K_m = K_n \prod c_i$$

Cost drivers

- Product attributes
  - Required software reliability (RELY)
  - Size of application database (DATA)
  - Complexity of the product (CPLX)
- Hardware attributes
  - Run-time performance constraints (TIME)
  - Storage constraints (STOR)
  - Volatility of the virtual machine environment (VIRT)
  - Required turnabout time (TURN)

Cost drivers (2)

- Personnel attributes
  - Analyst capability (ACAP)
  - Applications experience (AEXP)
  - Software engineering and programming capability (PCAP)
  - Virtual machine experience (VEXP)
  - Programming language experience (LEXP)
- Project attributes
  - Application of software engineering methods (MODT)
  - Use of software tools (TOOL)
  - Required development schedule (SCED)

Cost drivers values

Cost	Very Low	Low	Nom	High	Very High	Extra High
RELY	0.75	0.88	1	1.15	1.40	-
DATA	-	0.94	1	1.08	1.16	-
CPLX	0.70	0.85	1	1.15	1.30	1.65
TIME	-		1	1.11	1.30	1.66
STOR	-		1	1.06	1.21	1.56
VIRT	-	0.87	1	1.15	1.30	-
TURN	-	0.87	1	1.07	1.15	-
ACAP	1.46	1.19	1	0.86	0.71	-
AEXP	1.29	1.13	1	0.91	0.82	-
PCAP	1.42	1.17	1	0.86	0.70	-
VEXP	1.21	1.10	1	0.90		-
LEXP	1.14	1.07	1	0.95		-
MODT	1.24	1.10	1	0.91	0.82	-
TOOL	1.24	1.1	1	0.91	0.83	-
SCED	1.23	1.08	1	1.04	1.10	-

Manpower estimation

- The use of manpower is not uniform
  - Lower at the beginning and at the end of the project
- Usually, we have a fork, where the maximum estimation of double of the minimum
- Three models
  - Square
  - Triangular
  - Trapeze

Subprojects

- If the project can be divided into independent subprojects, the cost is cheaper
- Example:
  - $S_k = 60$
  - Three subprojects $S_{k1}=10$, $S_{k2}=20$, $S_{k3}=30$
- Single block
  - $S_k = 60 \rightarrow 294$ Pm
- Subprojects
  - $K_{m1} \rightarrow 40$Pm
  - $K_{m2} \rightarrow 86$Pm
  - $K_{m3} \rightarrow 135$Pm
  - Total: 261 Pm (< 294 Pm)

Detailed model

- Classifies each module in the most precise way
- Choses cost drivers for each module
- Applies cost drivers for each development phase
  - Specification (20%)
  - Design (30%)
  - Coding (40%)
  - Test (10%)

- A specialized software is required because the computation is complex

COCOMO II

- In 1995, COCOMO II was proposed, and in 2000 published, with the following differences
- COCOMO I requires software size in KNCSS as an input, instead COCOMO II is based on KSLOC (logical code)
  - The major difference between DSI and SLOC is that a single Source Line of Code may be several physical lines. For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI.
- COCOMO II addresses the following three phases of the spiral life cycle: applications development, early design and post architecture
- COCOMO I provides point estimates of effort and schedule, but COCOMO II provides likely ranges of estimates that represent one standard deviation around the most likely estimate.
- The estimation equation exponent is determined by five scale factors (instead of the three development models)

COCOMO II (2)

- Changes in cost drivers are:
  - Added cost drivers (7): DOCU, RUSE, PVOL, PLEX, LTEX, PCON, SITE
  - Deleted cost drivers (5): VIRT, TURN, VEXP, LEXP, MODP
  - Alter the retained ratings to reflect more up-do-date software practices
- Data from past projects in COCOMO I: 63 and COCOMO II: 161
- COCOMO II adjusts for software reuse and reengineering where automated tools are used for translation of existing software, but COCOMO I made little accommodation for these factors
- COCOMO II accounts for requirements volatility in its estimates

Summary

- COCOMO is still exploited in software companies
  - Even if it was conceived in 1981
  - And updated in 1995
- Cons
  - It is still a subjective approach
  - It does not provide all needed answers in the development, in particular in large projects
- Online simulator:
- http://softwarecost.org/tools/COCOMO/

Putnam model

In a perfect world…

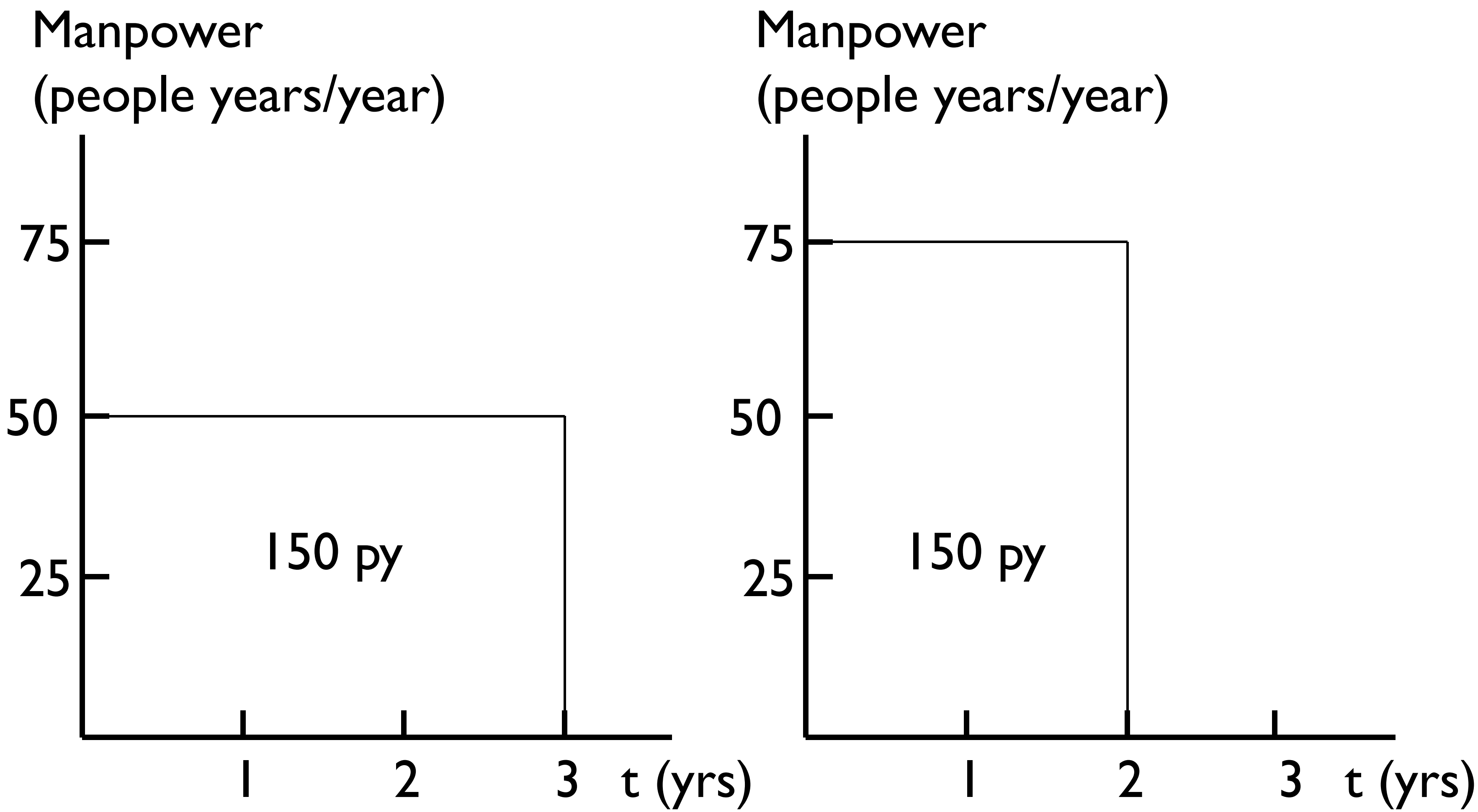
- Every manager aims at relying on a system that enables her to:
  - Estimate the number of needed **persons**
  - Assign them the **tasks**
  - **Divide** the estimated time by the number of people
  - Estimate the **delivery** time

- The real world is **different**

In the real world…

- Peter Norden (of IBM, in 1963) observed that a project is **not** a single monolithic activity, to be accomplished by a single team. Rather, it is a sequence of distinct but overlapping phases, each of which has its own natural team size and composition
- Frederick Brooks in his book "The Mythical Man-Month: Essays on Software Engineering" observed that time and people are **not** interchangeable
  - Brooks's law: Adding manpower to a late software project makes it later

The software myth (again)



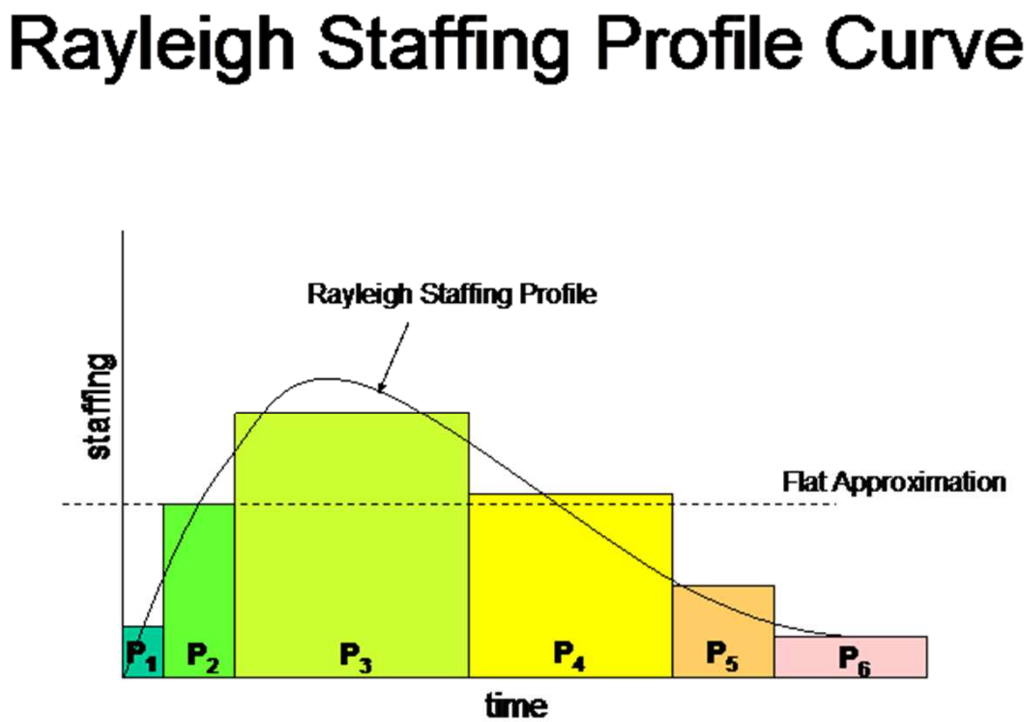
The software myth (2)

- **(Wrong) assumptions** which generate the myth:
  - Productivity (S/PY) is **constant** and can be determined by management
  - Product is directly **proportional** to effort (PY)

- Where S is the size of the project and PY is the person/year manpower effort
- The idea behind the assumptions is that we can double the manpower to carry out a project of double size

As a consequence

- Every project has a staffing curve
  - Natural
  - Consistent
  - Predictable



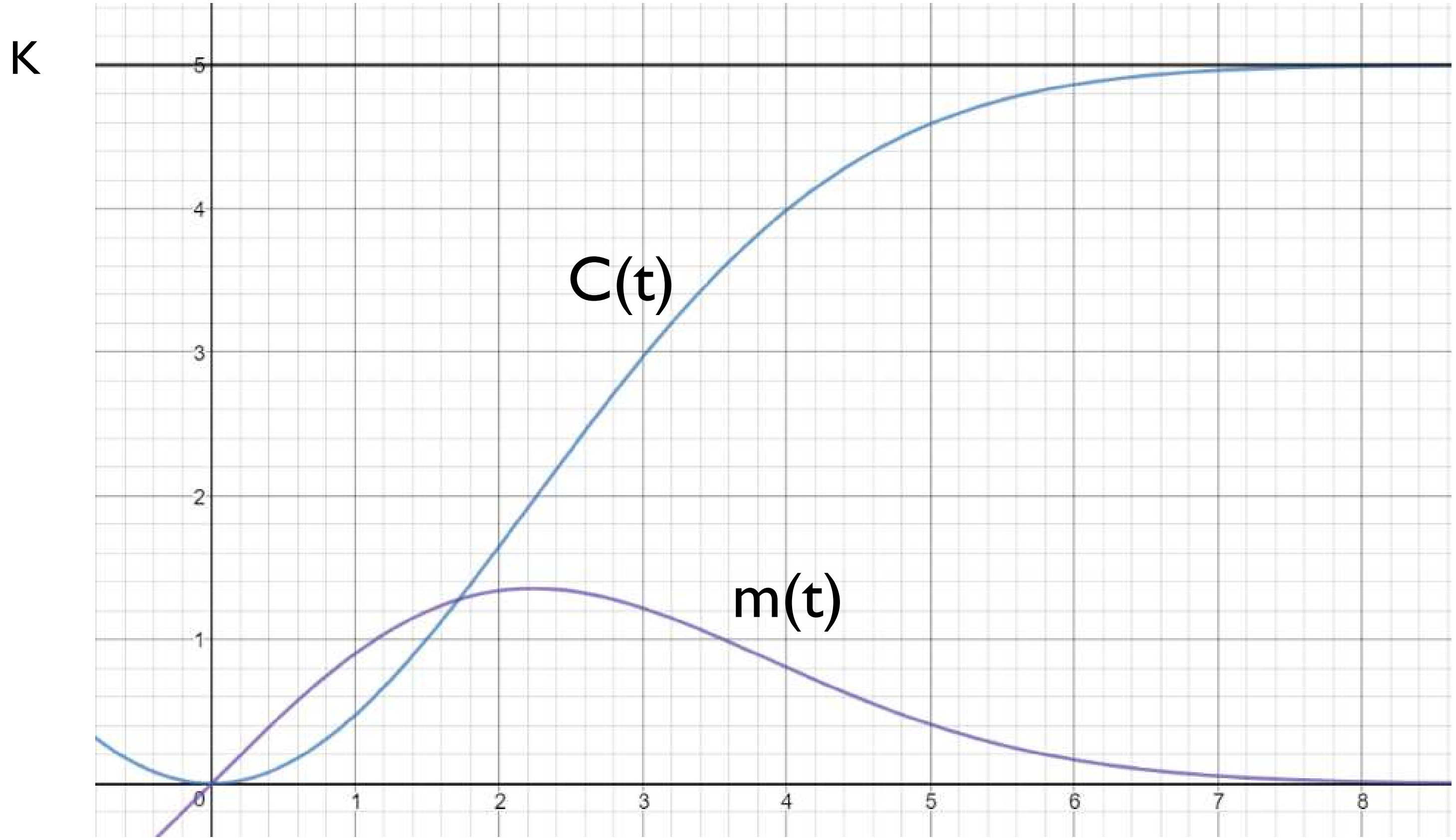
The Putnam model (SLIM)

- Lawrence Putnam exploited the work of Norden and Rayleigh to produce the SLIM method (Software LIfecycle Management)
  - Simply known as **Putnam model**

The Putnam model starting functions

- Putnam considers two functions:
  - m(t) is the distribution of the **manpower** in time
  - C(t) is the **cumulative** manpower in time, i.e. the integration of m(t) starting from 0
  - C(t) tends to reach the whole cost K
    - K is a constant value and is an asymptote for C(t)
    - K is considered in person/year
- Note: K has been introduced to reason about the projects, but it represents the *actual* cost only in some cases

The Putnam model starting functions (2)



The Putnam model assumptions

- The Putnam model is based on some assumptions
  1. The number of the problems to be solved is unknown but **finite**
  2. Problems are analyzed and solved by **human** work
  3. The solution actions of the problems are independent and random (Poisson distribution)
  4. The number of **people** activated at time *t* is proportional to the number of **problems** still to be solved

The Putnam equation

- The **required people** can be expressed by the derivative of C(t):

$$\frac{dC(t)}{dt}$$

- And also by the **work to do** $K - C(t)$ multiplied by the **required persons** per work unit x :

$$x[K - C(t)]$$

- So, the resulting (differential) equation is:

$$\frac{dC(t)}{dt} = x[K - C(t)]$$

- x increases with **time** and can be represented by $2\ a\ t$

The Putnam equation (2)

- The solution of the differential equation is:

$$C(t) = K\left(1 - e^{-(at^2)}\right)$$

- We can calculate also m(t):

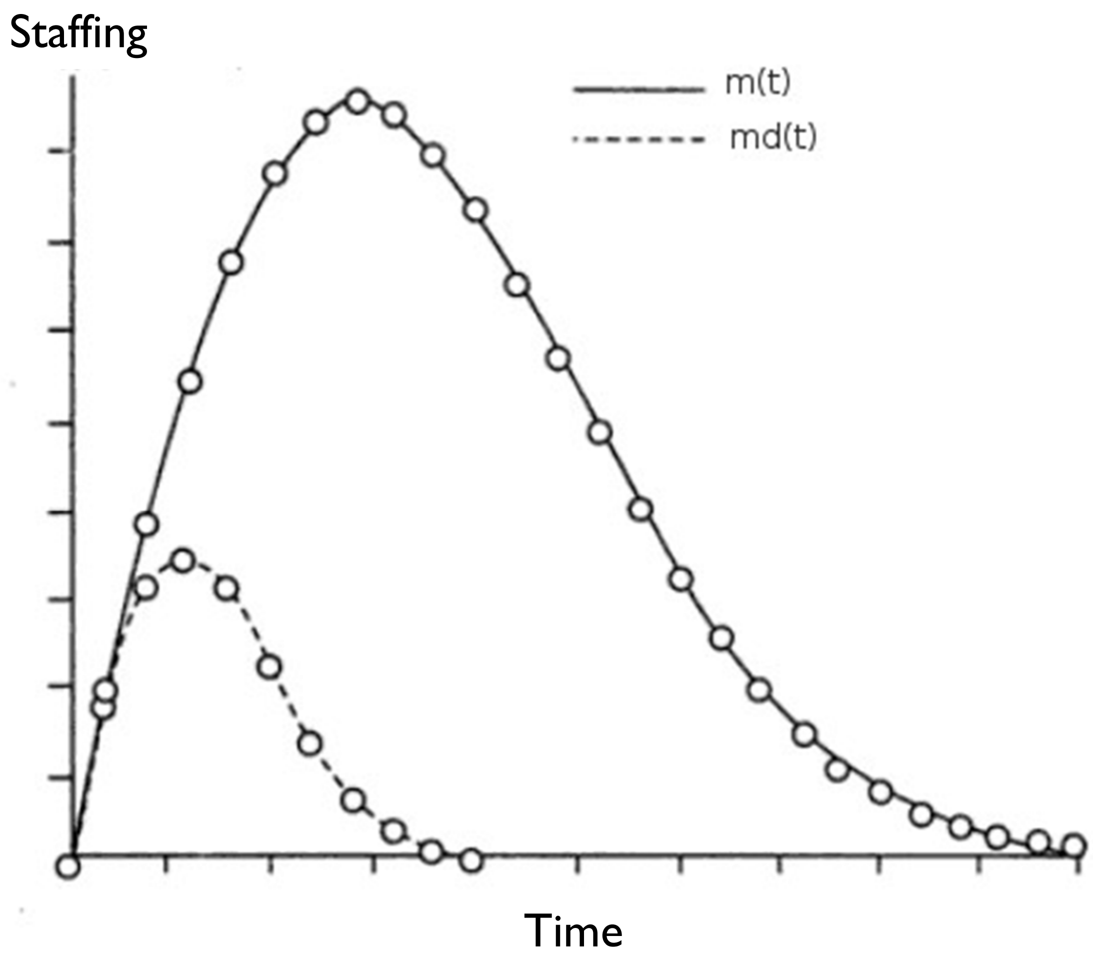
$$m(t) = C'(t) = 2K\,a\,t\,e^{(-at^2)}$$

- Now, we must calculate a

Project and development

- Putnam decomposes the *project* curve in three **subcurves**:
  - Development
  - Integration and test
  - Maintenance
- The most important is the first one, called $m_d(t)$
- Assumptions:
  - The curves $m(t)$ and $m_d(t)$ are similar
  - They are tangent in the origin

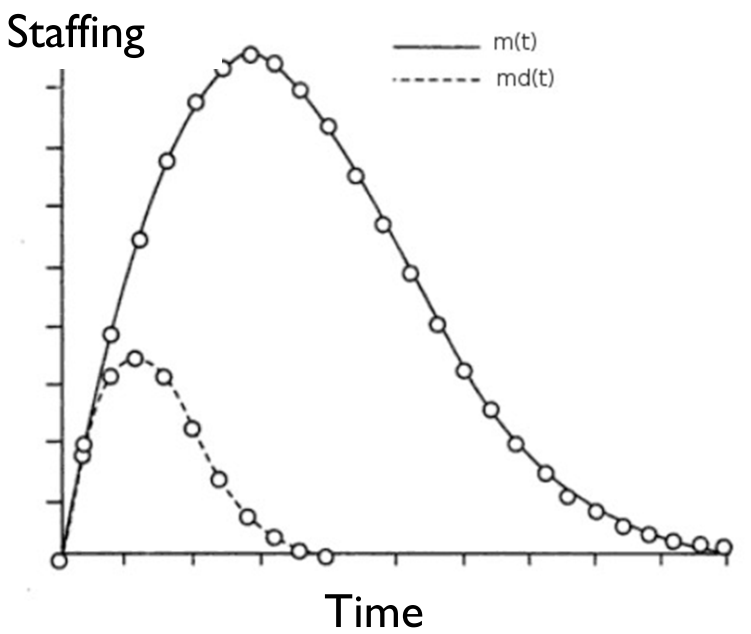
Project and development (2)



Project and development (3)

- Putnam supposes that the delivery time t_d corresponds to the peak of m(t)
  - After that, there is maintenance and so on
- It correspond also to the end of $m_d(t)$
  - Actually, when it reaches

  a given percentage
  - $C_d(t_d) = 0.95\ K_d$



The Putnam equation (3)

- We can calculate a in the sticking point of m(t), which happens at the delivery time t_d (end of $m_d(t)$):

$$\frac{dm(t)}{dt} = 2Ka\,(1 - 2at^2)e^{(-at^2)}$$

- We put the derivative equals to 0 in t_d:

$$2Ka\,(1 - 2at_d^2)e^{\left(-at_d^2\right)} = 0$$

- From which we calculate a :

$$a = \frac{1}{2t_d^2}$$

The Putnam equation (4)

- Putting all together:

$$m(t) = \frac{Kt}{t_d^2} e^{\frac{-t^2}{2{t_d}^2}}$$

- And also

$$C(t) = K\left(1 - e^{\frac{-t^2}{2{t_d}^2}}\right)$$

- More later…

Manpower peak

- The **peak** of manpower m_0 is:

$$m_0 = \mathrm{m}(t_d) = \frac{K}{t_d}\sqrt{e}$$

- And also

$$\mathrm{C}(t_d) = 0.39\ K$$

Difficulty

- Given the derivative of m(t):

$$m'(t) = \frac{K e^{\frac{-t^2}{2{t_d}^2}} (t_d^2 - t^2)}{t_d^4}$$

- Putnam defines the **difficulty** D of the project, calculated as the slope of the curve at t = 0:

$$D = m'(0) = \frac{K}{{t_d}^2}$$

- The idea is that more work is required and shorter is the time to deliver, more effort is required from the beginning
  - More people are required **early**

Difficulty (2)

- We can also calculate the derivative of $D(t_d)$:

$$D'(t_d) = -2\frac{K}{{t_d}^3}$$

- And we define D_0 as a simplification:

$$D_0 = \frac{K}{{t_d}^3}$$

- D_0 does **not** have a *real* meaning
- But it has revealed as an **important parameter** of the project to be developed

Remarks

- From the analysis of the past projects, we can see that values of D_0 aggregate around three values:
  - 8 for new and complex projects
  - 15 for new but simple projects
  - 27 for reworking existing projects
- This result is very similar to the Organic, Semi-detached and Embedded projects of CoCoMo (!)

The Putnam equation (5)

- Let now consider also the **size** of the software and the **productivity**
- S = size of the software in NCSS (Non-Commenting Source Statement)
- Productivity P_r: ratio between the produced code and the manpower requested to produce it

$$P_r = \frac{S}{C(t_d)}$$

- Applying the formula at slide 71:

$$P_r = \frac{S}{0.39 \cdot K}$$

The Putnam equation (6)

- Putnam considers also the place where the software is developed
- Putnam introduces an **environment** factor **E**, also called "technology coefficient"
- Putting all together, the resulting **software equation** is:

$$S = E \cdot K^{\frac{1}{3}} \cdot t_d{}^{\frac{4}{3}}$$

- From which:
- Person years invested: $K = \left(\frac{S}{E\ t_d^{4/3}}\right)^3$
- Time to develop: $t_d = \left(\frac{S}{E\ K^{1/3}}\right)^{\frac{3}{4}}$

The Putnam equation (7)

- For old projects we have S, K and t_d
  - → we can calculate E and D_0
- For new projects we know E and D_0 and can estimate S
  - → we can calculate K and t_d

Example 1

- Let's consider the following data:
- S = 50000 NCSS (estimation)
- E = 12712 (from previous projects)
- K =12 Py (from previous projects)

- Applying the previous formula we obtain:

$$t_d = \left(\frac{S}{E\ K^{1/3}}\right)^{\frac{3}{4}} = \left(\frac{50000}{12712 \cdot 12^{1/3}}\right)^{\frac{3}{4}} = 1.5\ years$$

Multi variable

- As we can see from the previous formulas, we can fix some variables and compute the remaining one
- Or we can fix some, make the other vary and compute the remaining one

Example 2

- Let's consider the following data:
- S = 100000 NCSS
- E = 10040
- t_d= various

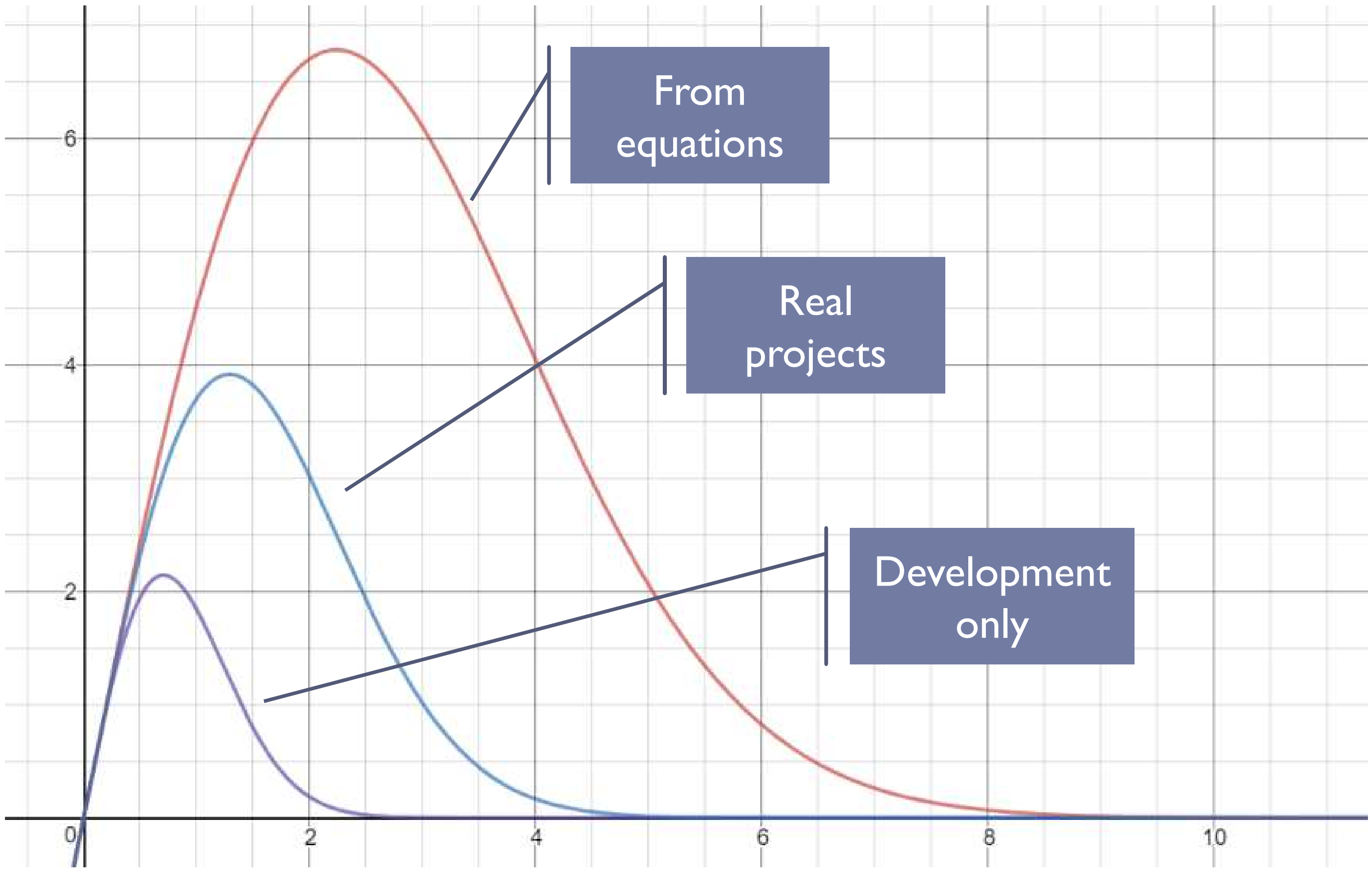
- K can be computed depending on t_d as follows
- $K = \left(\frac{S}{E\ t_d^{4/3}}\right)^3 = \left(\frac{100000}{10040\ t_d^{4/3}}\right)^3$

t_d	K
1	988
1.5	195
2	62

Real projects

- The estimation by Putnam model are likely to be **oversize** for small projects
- In small projects, the total costs is similar to the development cost
- To correct the estimations, the parameter α is introduced
  - α depends on the code size

Real projects (2)



Corrections

Size	Manpower	Year	Persons
S<18000	K_d	t_{0d}	m_{0d}
18000 < S < 70000	$K/_{\alpha^2}$	$t_d/_{\alpha}$	$m_0/_{\alpha}$
S>70000	K	t_d	m_0

- Where K_d, t_{0d}, m_{0d} are the value of the development curve
- $\alpha = 1 + 6.23 \cdot e^{-0.079 S_k}$
- $\alpha = \begin{cases} 1 & S > 70000 \\ \sqrt{6} & S < 18000 \end{cases}$

Example 3

- Let's consider the following data for a new project:
- S = 36000 NCSS (estimation)
- D_0 = 16 p/y^3 (from similar past projects)
- E = 9600 (company parameter)
- Applying the formula at slide 73 we obtain:
- $D_0 = \frac{K}{{t_d}^3} \rightarrow K = D_0 \cdot {t_d}^3$
- Applying the software equation at slide 75:
- $S = E \cdot K^{\frac{1}{3}} \cdot {t_d}^{\frac{4}{3}} \rightarrow E \cdot D_0^{\frac{1}{3}} \cdot {t_d}^{\frac{7}{3}}$
- From which:
- $t_d = \left(\frac{S}{E \cdot D_0^{\frac{1}{3}}}\right)^{\frac{3}{7}} = \left(\frac{36000}{9600 \cdot 16^{\frac{1}{3}}}\right)^{\frac{3}{7}}$ so $t_d = 1.18$ y

Example 3 (2)

- Because 18000 < S < 70000 we can/must correct the estimation
- $\alpha = 1 + 6.23 \cdot e^{-0.079 S_k} \rightarrow \alpha = 1.36$
- From the table, the number of years is:
- $\frac{t_d}{\alpha} = 0.87\ \mathrm{y}$
- From these values, we can calculate others

Pros

- Provides tools support during **all** the development cycle
- Encourages **good practices**
- Enables the **scheduling** of the value added costs
- Provides estimations for **staffing**
- Simplifies the **strategic** decisions and the "what if" scenarios
- Generates graphs and estimations that can be **updated**

Cons

- Is better with **large** projects (> 5kNCSS, > 6 months)
- The size of code must be estimated **in advance**, which depends also on technological aspects
- The model is very **dependent** on the t_d and S values
- Assumes a **waterfall** development model, works worse with others
- Is a **complex** model with complex computation

Summary

- Putnam model provides more estimations than CoCoMo
- It is easier to evaluate different scenarios, modifying the values of the parameters
- But the computation is more complex

PERT/CPM

PERT

- Program Evaluation and Review Technique
- Shows the **activities** and their **dependencies**
  - In particular, the **precedence** between activities
  - Before an activity can begin, the ones that precede it must be completed
- Provides also estimations of the **duration** of the activity

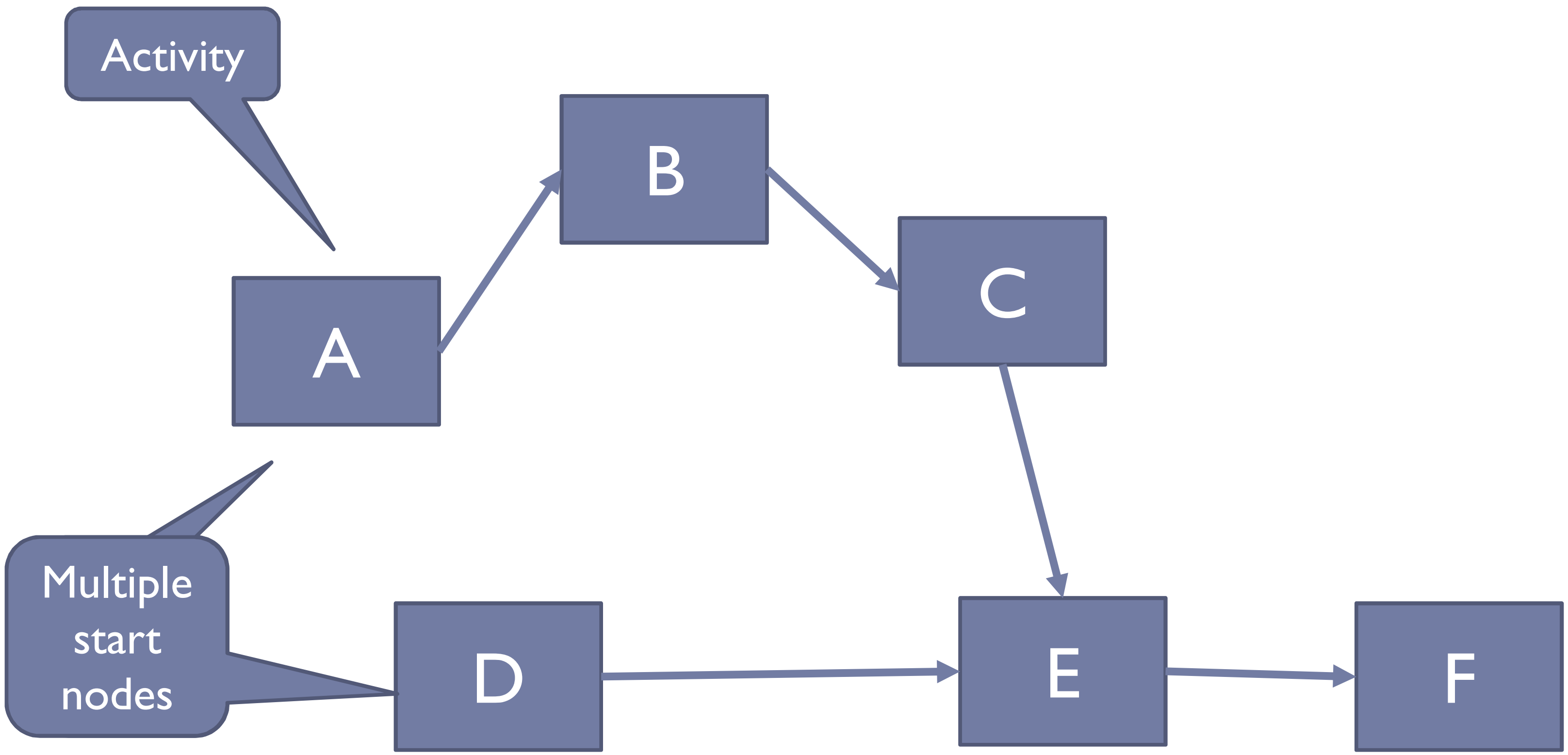
Dependencies

- Types of dependencies:
  - Start-start
    - The second task must start after the first task has started
  - Start-finish
    - The first task must start before the second task has finished
  - Finish-start
    - The second task must start after the first task has finished
  - Finish-finish
    - The first task must finish before the second task has finished

PERT AON diagram

- Activities on **node** (AON)
  - **Nodes** of the graph represent **activities**
  - **Arrows** specify **precedence**
  - There can be **more than one** single start and end node
  - Used more in the **past**
    - Easier application in linear programming
    - Less intuitive
    - Can be ambiguous
- Sometimes referred to as "CPM charts"

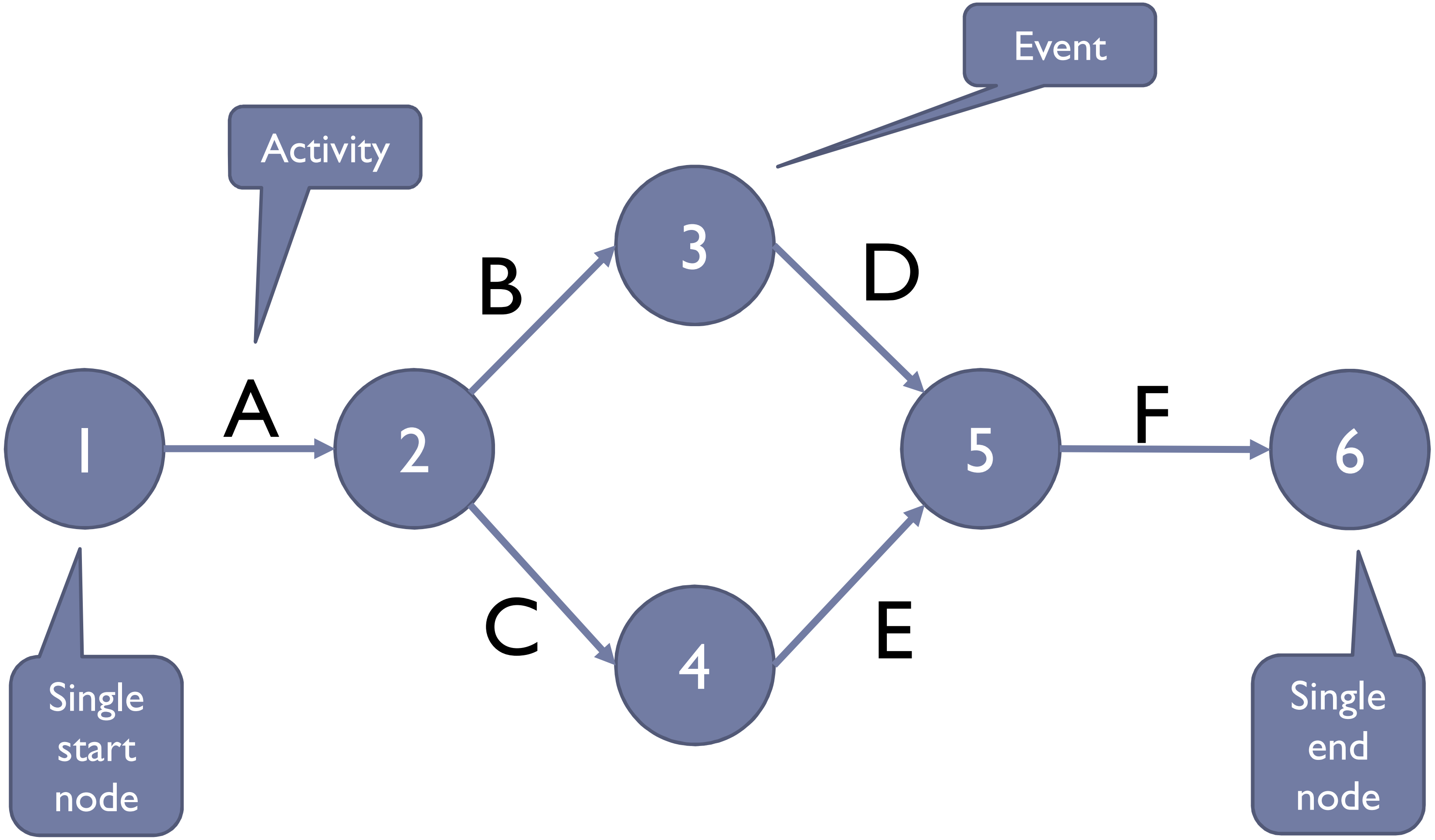
Example (AON)



PERT AOA diagram

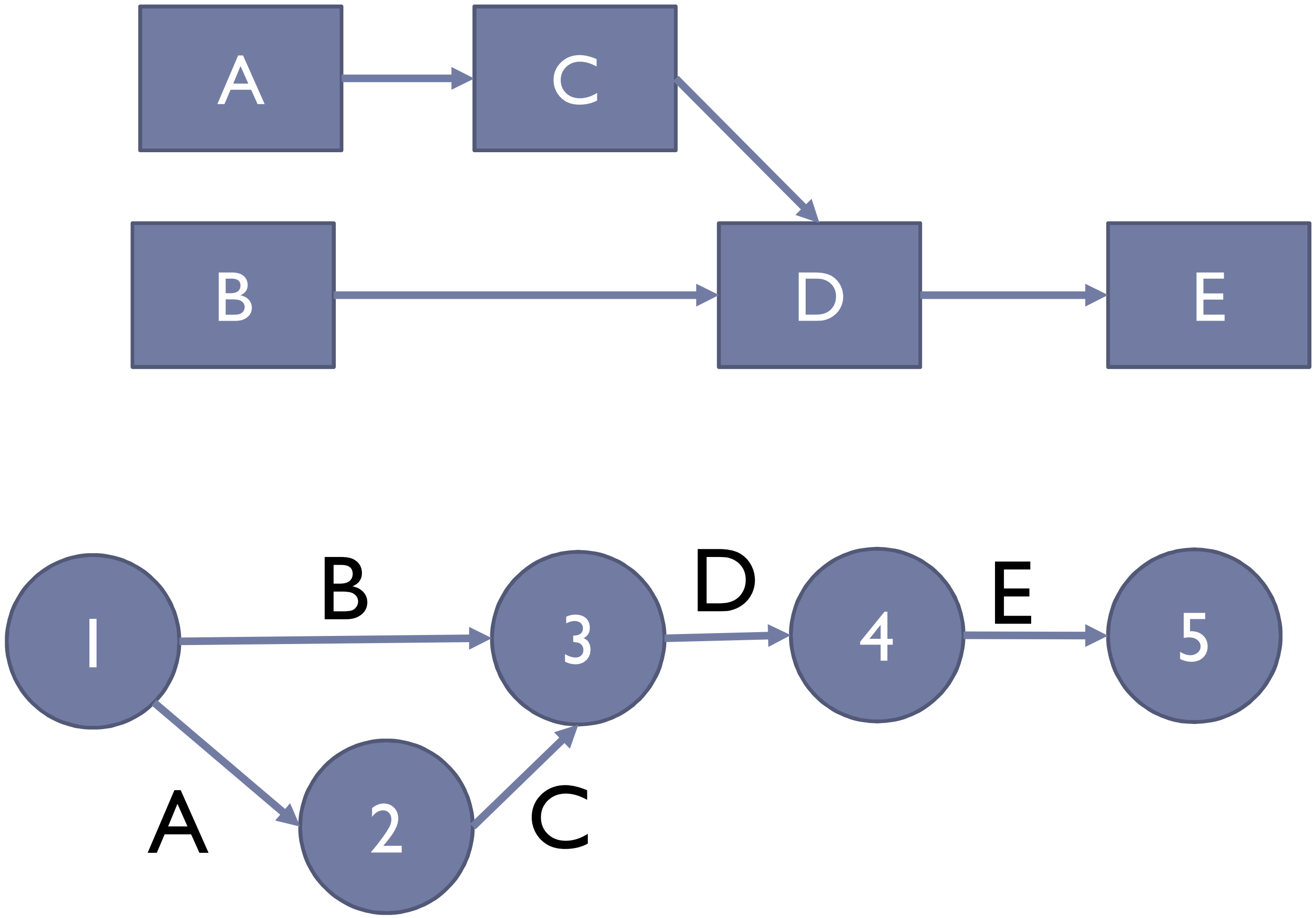
- Activities on **arrows** (AOA)
  - **Arrows** of the graph represent **activities**
  - **Nodes** specify beginning and end of activities (milestones)
  - **One** single start and end node
  - **Currently** exploited

Example (AOA)


Event
Activity
B
3
D
A
1
2
5
F
6
C
4
E
Single start node
Single end node

Comparison

Act.	Preced.
A	
B	
C	A
D	B,C
E	D

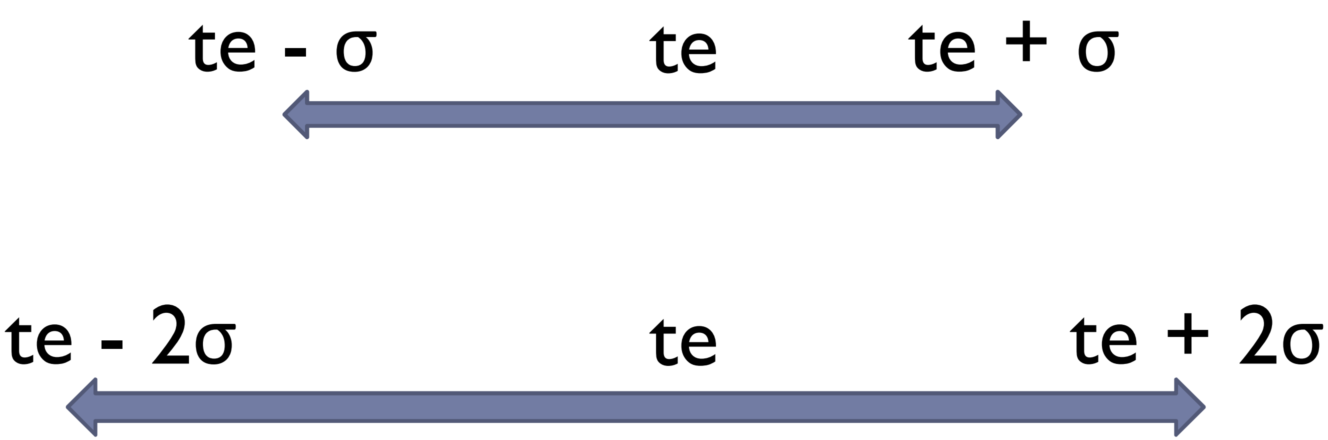


Duration estimation

- PERT provides an estimation of the duration of each activity based on **three values**:
  - to = **optimistic** time
  - tm = **most likely** time
  - tp = **pessimistic** time
- The **expected** time is:
  - te = (to + 4*tm + tp) /6
- The **standard deviation** is:
  - $\sigma = (tp - to) / 6$
- It is useful the define how much an **estimated** duration can be different from the **real** duration

Duration estimation (2)

- The range of the estimated project duration is te ± σ
  - This provide a likelihood of 68.3%
- To increase the likelihood to 95.5% the range must be extended to te ± 2σ



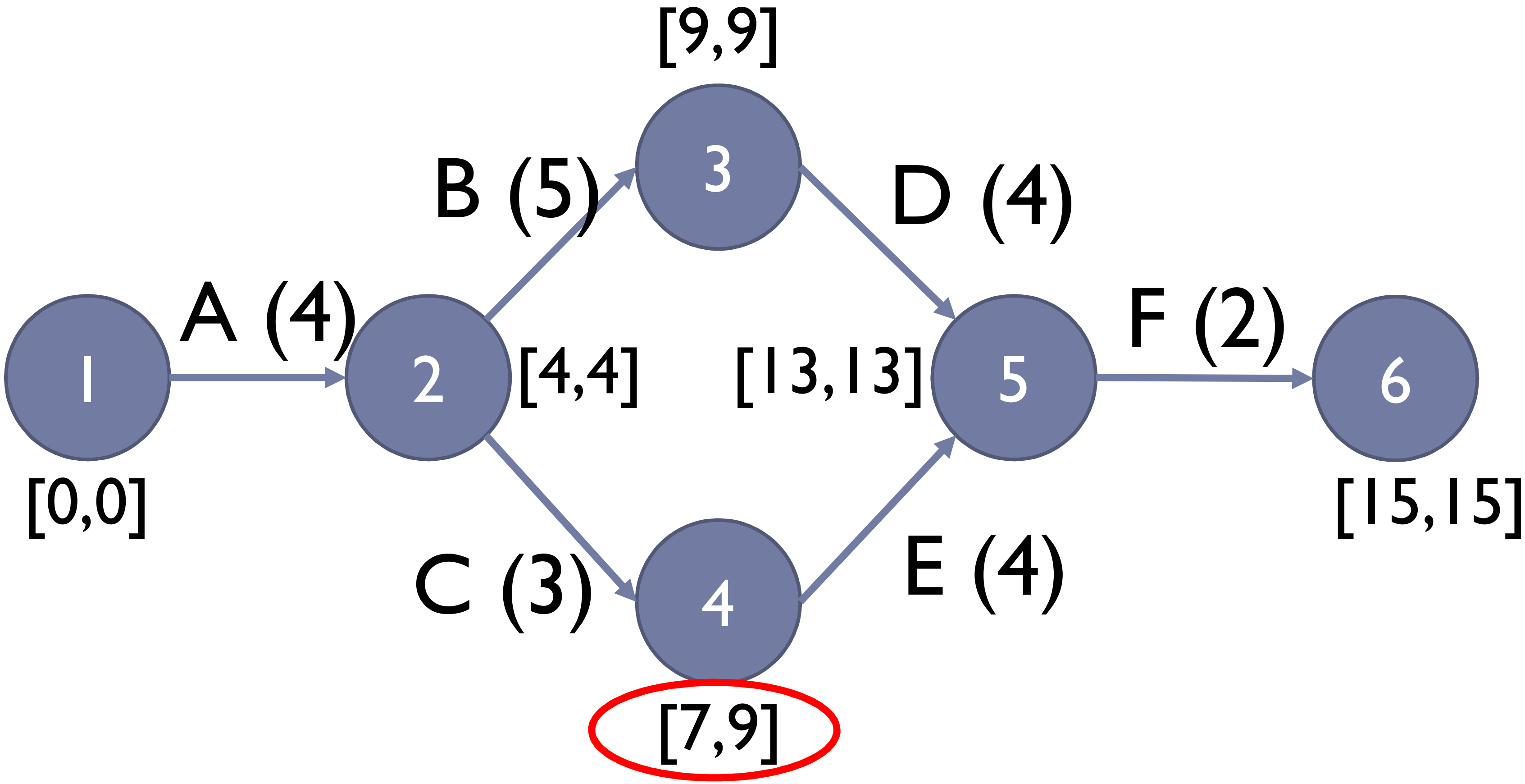
CPM

- Critical Path Method
- Is exploited to calculate the duration of a whole project given:
  - The duration of the single activities
  - The dependencies between activities

Time

- Each activity has a duration
- Each event has a minimum and a maximum time
  - $[t_{min}, t_{max}]$
  - The minimum time is the **maximum** of the times of the **incoming** arrows
  - The maximum time is the **minimum** of the times of **outgoing** arrows
- **Dummy** activities can be introduced to specify dependency between nodes without real activities
  - 0 duration
  - Dashed line

Example



UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Hints

- In the **start** node:
  - $t_{min} = t_{max} = 0$
- In the **end** node:
  - $t_{min} = t_{max}$
- When there is only one incoming activity A:
  - $t_{min} = t_{min\_prev} + t_A$
- When there is only one outgoing activity B:
  - $t_{max} = t_{max\_next} - t_B$

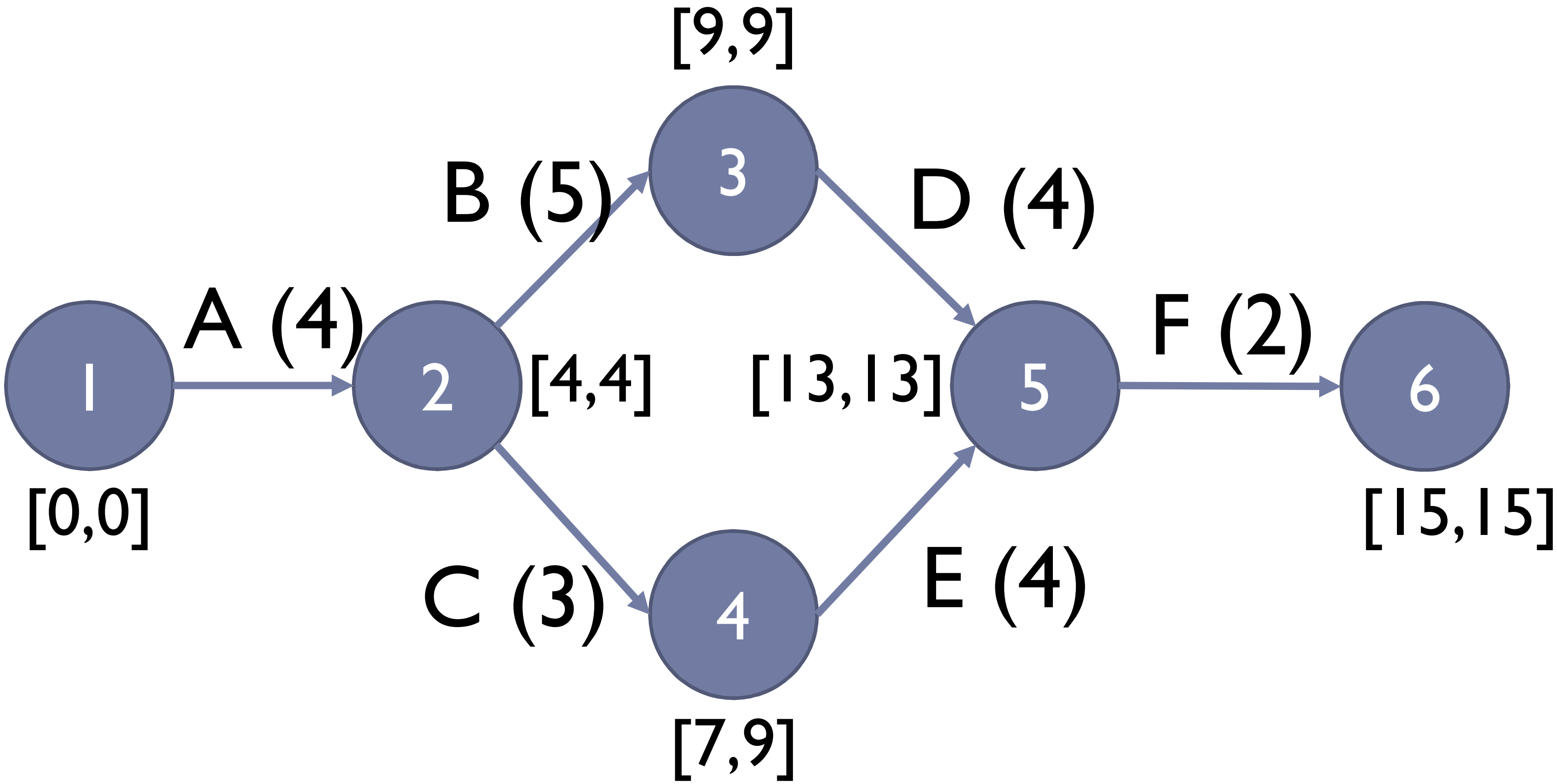
Critical path

- The critical path is the path from the start node to the end node that determines the duration of the whole project
- It is the path where $t_{min} = t_{max}$ for every node
- If an activity in the critical path is delayed, the **whole** project is **delayed**
- Activities **outside** the critical path can be delayed up to reaching t_{max} for the next node

Critical path (2)

- From a **formal** point of view, it is the path from the start node to the end node where the sum of the activities' durations is maximum
  - Given n_s the start node and n_e the end node
  - $p_i = \langle n_s, n_{i_1}, n_{i_2}, \dots, n_{i_k}, n_e \rangle$ is a path so that there exist an activity $A_{i_{m,m+1}}$ between every couple of consecutive nodes n_{i_m} and $n_{i_{m+1}}$
  - The total time of a path is the sum of the corresponding activities $t_{p_i} = \sum t_{A_i}$
  - $P = \{p_i\}$ is the set of all paths in the graph
  - The critical path cp is the path for which $t_{cp} = max_{p_i \in P}(t_{p_i})$ holds

Example



Putting all together

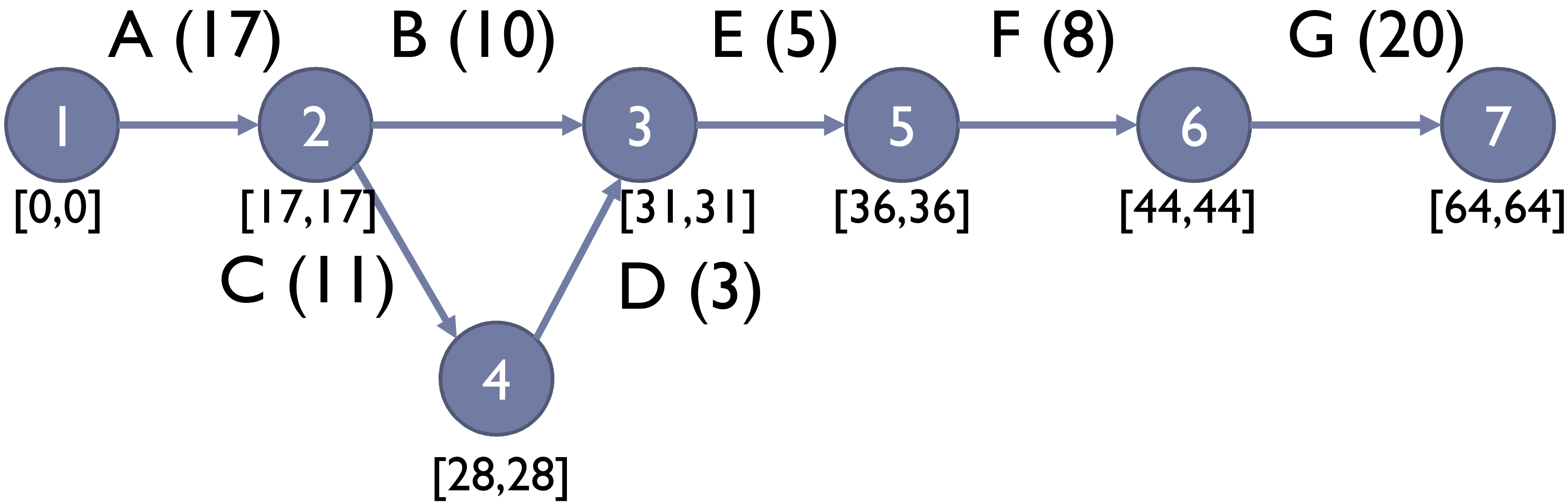
- Identify the activities
- Define the sequence of the activities
- Build the PERT diagram
- Estimate the duration of each activity
- Determine the critical path

- Optimize the activities in the critical path
- During the project development, control that the duration is the expected one

Example 1 – activities and times

Activity	Description	Precedence	to	tm	tp	te
A	Initial design		12	16	26	17
B	Survey market	A	6	9	18	10
C	Build prototype	A	8	10	18	11
D	Test prototype	C	2	3	4	3
E	Redesigning	B,D	3	4	11	5
F	Market testing	E	6	8	10	8
G	Set up production	F	15	20	25	20

Example 1 – PERT and CPM



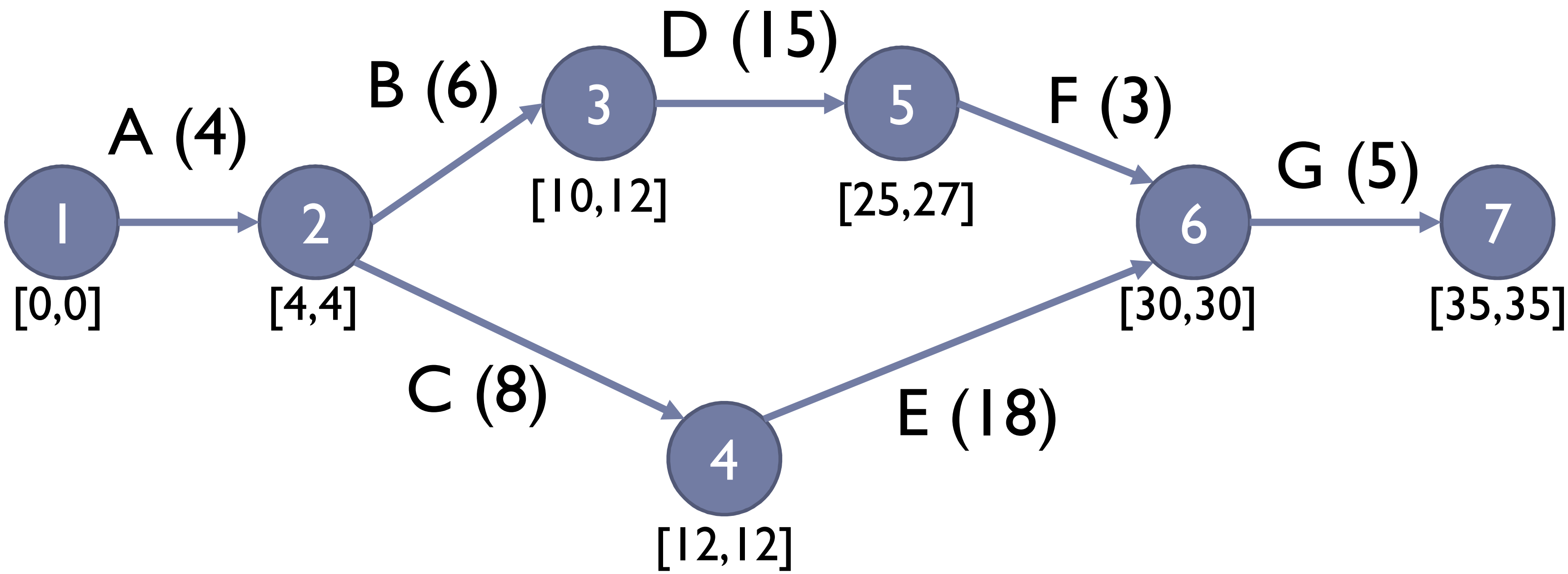
Critical path = A, C, D, E, F, G → 64

Slack of B = 4

Example 2 – activities and times

Activity	Description	Precedence	to	tm	tp	te
A	Architecture definition		2	4	6	4
B	Client definition	A	2	5	12	6
C	Server definition	A	4	7	15	8
D	Client implementation	B	10	15	20	15
E	Server implementation	C	12	18	26	18
F	GUI design	D	2	3	4	3
G	System test	E,F	3	4	11	5

Example 2 – PERT and CPM



Critical path = A, C, E, G → 35

Slack of B = 2

Slack of D = 2

Slack of F = 2

But not all!

Pros

- Estimation of the **expected** time of the project
- Identification of the **critical** activities
  - Should be optimized
  - Cannot be delayed
- Identification of the activities with **slack** time
  - Can be delayed
  - Can lend resources to critical activities

Cons

- Some **computation** is needed
  - Not NP-hard because it is direct acyclic graph
- Graphs can be big and **difficult** to manage for projects with many activities
- The graph is supposed to be updated **during** the project development
- Emphasis on **time** factors

Summary

- PERT/CPM provides not only a means to estimate the duration of a project, but also to estimate:
  - Which activities are critical
  - Which activities can be delayed
- Moreover, it is useful to control the project schedule during the development

Wrapping up


HOW LONG WILL YOUR PROJECT TAKE IF I ADD TWO PEOPLE?
ADD ONE MONTH FOR TRAINING, ONE MONTH FOR THE EXTRA COMPLEXITY, AND ONE MONTH TO DEAL WITH THEIR DRAMA.
BUT AFTER ALL OF THAT...
THEY'LL BE AS USEFUL AS THIS MEETING.
Dilbert.com DilbertCartoonist@gmail.com
4-29-10 © 2010 Scott Adams, Inc./Dist. by UFS, Inc.