SSW-540: Fundamentals of Software Engineering

Software Project Management and Estimation

Dr. Richard Ens
School of Systems and Enterprises

Python Pointers

Python has a very neat function for breaking strings into smaller strings.

Split( ) is a method that tokenizes a string and returns a list of “words”, a string array as it were, splitting the string on whitespace unless directed otherwise and optionally limiting the number of splits.

str.split(*delimiter*=“ ”, *num*=string.count(str))

Multiple delimiters may be specified using a | to separate them.

Examples:

```
x = ‘See the cat’
y = x.split ( )                #this will split on whitespace
print y                        #this prints a list: [’See', ’the', ’cat']

a, b, c = x.split(” “)
print c                        #prints: cat
print b                        #prints: the
```

Managing software projects

Managing software projects is different than classic project management because:

- The product is intangible.
  - Software cannot be seen or touched. Software project managers cannot see progress by simply looking at the artefact that is being constructed.
- Many software projects are 'one-off' projects.
  - Large software projects are usually different in some ways from previous software projects. Even managers who have lots of previous experience may find it difficult to anticipate problems.
- Software processes are variable and organization specific.
  - We still cannot reliably predict when a particular software process is likely to lead to development problems.

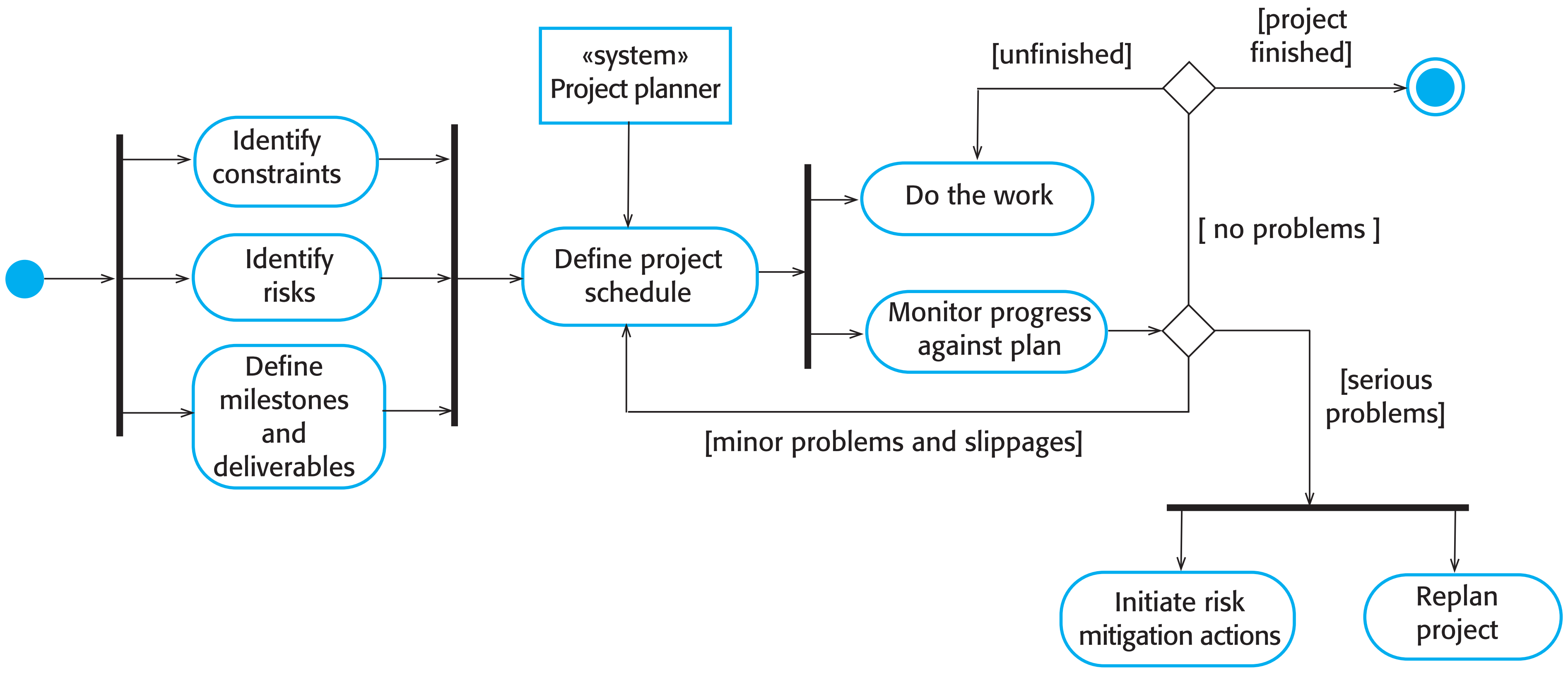
Universal management activities

- *Project planning* - Project managers are responsible for planning. estimating and scheduling project development and assigning people to tasks.
- *Risk management* - Project managers assess the risks that may affect a project, monitor these risks and take action when problems arise.
- *People management* - Project managers have to choose people for their team and establish ways of working that leads to effective team performance.
- Additional activities in many, but not all projects
  - *Reporting* - Project managers are <u>usually</u> responsible for reporting on the progress of a project to customers and to the managers of the company developing the software.
  - *Proposal writing* - The first stage in a software project <u>may</u> involve writing a proposal to win a contract to carry out an item of work.

Project planning

- Project planning involves breaking down the work into parts and assigning these to project team members, anticipating problems that might arise and preparing tentative solutions to those problems.
- The project plan, which is created at the start of a plan-driven project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.
- A startup plan, supporting decisions about budget and staffing, is needed even for agile development to allow resources to be allocated to the project.
- The project plan should be regularly amended as the project progresses, as you know more about the software and its development.
- The project schedule, cost-estimate and risks have to be regularly revised.

The project planning process for plan-driven development



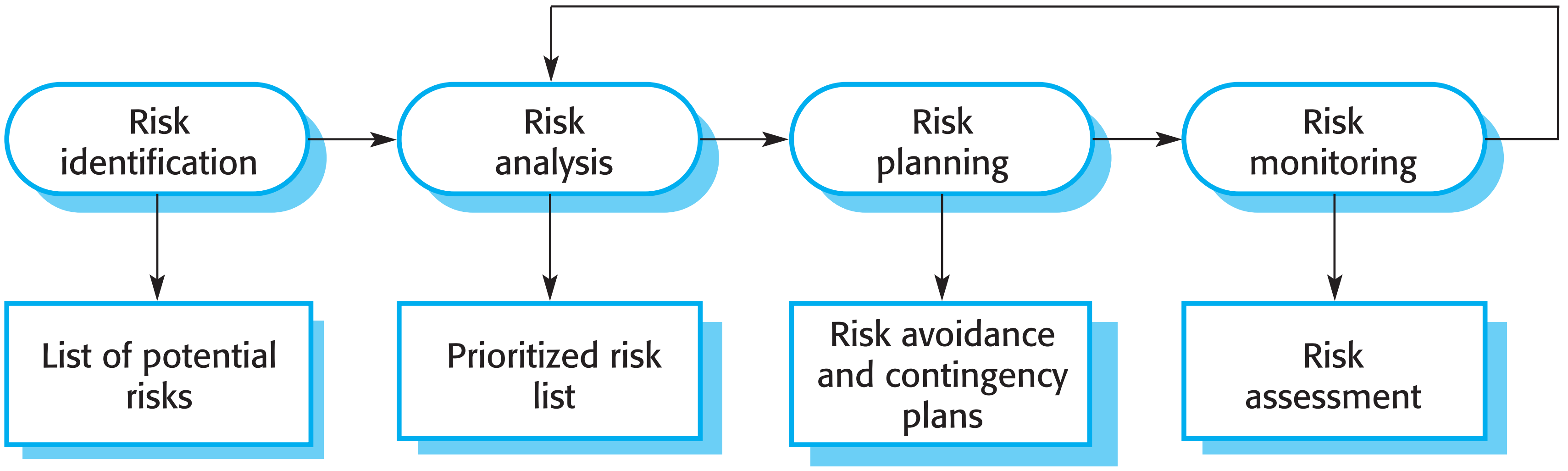
Risk management

- Risk management is concerned with identifying risks and drawing up plans to minimize their effect on a project.
- Software risk management is important because of the inherent uncertainties in software development.
  - These uncertainties stem from loosely defined requirements, requirements changes due to changes in customer needs, difficulties in estimating the time and resources required for software development, and differences in individual skills.
- You have to anticipate risks, understand the impact of these risks on the
  - Project - impacting schedule or resources
  - product - affecting the quality or performance of the software
  - business - affecting the organization developing or procuring the software
- take steps to avoid these risks.

Examples of risks

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organizational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool underperformance	Product	CASE tools, which support the project, do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The risk management process



Identify project, product and business risks	Assess likelihood & consequences	Create plans to avoid or minimize	Monitor the risks throughout

Quantification of risks

- Risk exposure is the product of risk probability and risk impact, the latter usually estimated in dollars. E.g.,
  - The impact of a software defect found in released software can be measured in terms of cost to repair, cost to reputation, cost to customer, etc.
  - The probability of a expensive defect being found might be estimated based on history, code complexity, or even "gut feeling".
- The Air Force uses a matrix and categories of probability and impact:

Very High	5	7	9	11	13
High	4	6	8	10	12
Medium	3	5	7	9	11
Low	2	4	6	8	10
Very Low	1	3	5	7	9
Probability / Impact	Very Low	Low	Medium	High	Very High

Why quantify risks?

- Quantification helps set priorities. Greater risk exposure should mean higher priority!
- Quantification can help understand *leveraging* of risk.
    - We can often spend to reduce risk; leverage is the original exposure minus the new exposure, divided by the amount spent or $(RE_1 – RE_2)/AmtSpent$
    - Example: Losing my star developer, Alex, during a critical development period would set my project back several weeks. Each week costs me $100,000 in loaded wages so 2½ weeks delay will cost $250,000. Alex is prone to sickness so the probability of losing her/him at some point in the 6 week project is 50%. My exposure (RE_1) is $125,000.
    - Should I spend $30,000 to have a personal physician tend to Alex during the project if that would reduce the probability of losing him/her to 20%?

$RE_2 = (250{,}000 * 0.20) + (.80 * 30{,}000) = 74{,}000$

Leverage = $(125{,}000 – 74{,}000)/30{,}000 = 1.7$

Any leverage over 1 is worth considering!

This term is money spent even though the risk doesn't happen; it's added to the exposure.

Another risk leverage calculation example

There is a 0.5% probability that a latent defect will execute and lead to a failure, and that failure would cost the customer **$100,000**

So, **RE** = (0.005) * (100,000) = **$500**

We could hold a design review that will cost **$100** in professional time and halve the number of latent faults

The resulting (new) risk exposure is

NewRE = ((0.005/2) * (100,000)) + ((0.9975) * $100) = ~**$350**

The risk reduction leverage (RRL) is (RE-NewRE)/cost of risk reduction

RRL = (500-350)/100 = **1.5**

Any risk reduction leverage equal to or greater than 1 is worth serious consideration.

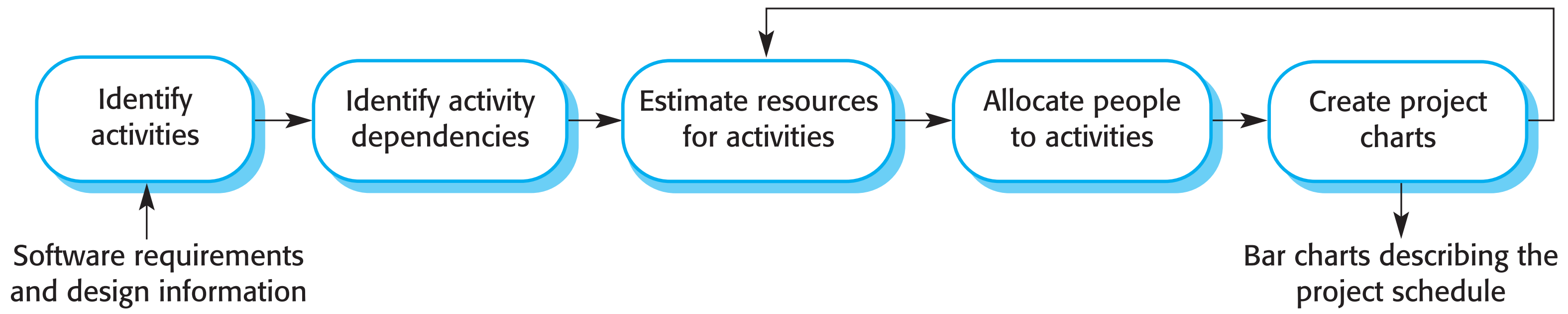
What if the design review had cost $200?

Dealing with risks

- Fundamentally, strategies for dealing with risks fall into three categories:
  - Avoidance strategies to reduce the probability that the risk will arise.
  - Minimization strategies to reduce the impact of the risk on the project or product.
  - Contingency plans to deal with that risk when it does occur.
- Your text offers strategies to help manage many commonly occurring risks in software development.
- Many development organizations set up financial reserves to cover the potential losses of risks occurring during development. These monies are released once the threat of the risk has passed. (Works like warranties.)

Project scheduling

- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- You estimate the calendar time needed to complete each task, the effort required and who will work on the tasks that have been identified.
- You also estimate the resources needed to complete each task, such as the server time, the time required on specialized hardware, such as a simulator, and/or what the travel budget needs to be.


Identify activities
Identify activity dependencies
Estimate resources for activities
Allocate people to activities
Create project charts
Software requirements and design information
Bar charts describing the project schedule

Scheduling problems

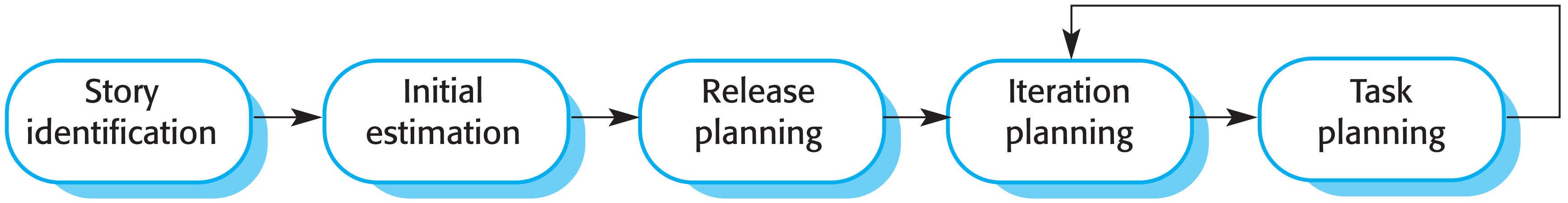
- Estimating the difficulty of problems and hence the cost of developing a software solution is hard.
- Productivity is not proportional to the number of people working on a task.
    - Individual productivity declines as group size grows.
    - Bell Labs estimated the optimal size of software development teams to be 5-6.
- Adding people to a late project makes it later.
    - This is called Brooks' Law, first cited by Fred Brooks Jr. in his essay collection *The Mythical Man-Month*.
    - Projects lag when new staff arrives because of training (ramp-up) needs, additional communication overheads and the limited divisibility of tasks.
- The unexpected always happens. Always allow contingency in planning.

Agile planning

- Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.
- Unlike plan-driven approaches, Agile teams spread the planning process throughout the development cycle.
  - The decision on what to include in an increment depends on progress and on the customer's priorities.
- The customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes.
- Stages of Agile planning:
  - Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system.
  - Iteration planning, which has a shorter term outlook, and focuses on planning the next increment of a system. This is typically 2-4 weeks of work for the team.

Approaches to Agile planning and estimation

- Planning in Scrum
  - Tasks divided into time-boxed sprints.
  - Based on managing a project backlog (things to be done) with daily reviews of progress and problems.
- The planning game
  - Developed originally as part of Extreme Programming (XP)
  - Dependent on user stories as a measure of progress in the project
- Watch the 12 minute film: [Agile Estimation]()



Agile planning plusses and minuses

- Benefits of this approach:
  - Agile planning works well with small, stable development teams that can get together and discuss the stories to be implemented.
  - The whole team gets an overview of the tasks to be completed in an iteration.
  - Developers have a sense of ownership in these tasks and this is likely to motivate them to complete the task.
- Agile planning difficulties
  - Agile planning is reliant on customer involvement and availability.
  - Some customers may be more familiar with traditional project plans and may find it difficult to engage in an agile planning process.
  - When teams must be large, or when team membership changes frequently, it is not practical for everyone to be involved in the collaborative planning that is essential for agile project management.

Estimating *effort* and cost

- The cost of developing software is primarily the cost of personnel.
- Other cost factors (tools, resources, travel, etc.) typically are minor compared to the cost of staff.
- The major determinant of development cost is software size.
- Fred Brooks, in his essays, The Mythical Man Month, notes that software developers are not like farm laborers.
  - Different skills and experience are needed in various amounts and at different times in a project.
  - A project requiring 100 staff-months cannot be completed with 1 staff member working 100 months or 100 staff members working 1 month!
- But the cost of 100 staff-months is the same regardless of the # of staff and the # of calendar months! When we estimate *effort*, we express it in staff-months.

Estimation techniques

- Software organizations need to make software *effort* and cost estimates so they can plan around a budget and estimate time to completion.
- There are two types of technique that can be used to do this:
  - *Experience-based techniques:* The estimate of future *effort* requirements is based on experience of past projects and the application domain. Essentially, the manager or developer makes an informed judgment of what the *effort* requirement is likely to be.
  - *Algorithmic cost modeling:* In this approach, a formulaic approach is used to compute the project *effort* based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.
- Which of these is used in Agile estimation?

Experience-based approaches

- Experience-based techniques use judgments based on the *effort* expended on software development activities in past projects.
- Typically, you first identify the deliverables of software components or systems that are to be produced in a project.
- You estimate them individually and compute the total *effort* required.
- It usually helps to get a group of people involved in the *effort* estimation and to ask each member of the group to explain their estimate.
- But…
  - A new software project may have little in common with previous projects.
  - Software development changes very quickly and a project will often require the use of new and/or unfamiliar techniques.
  - If you have not worked with these techniques, your previous experience may not help you to estimate the *effort* required, making it difficult to produce accurate costs and schedule estimates.

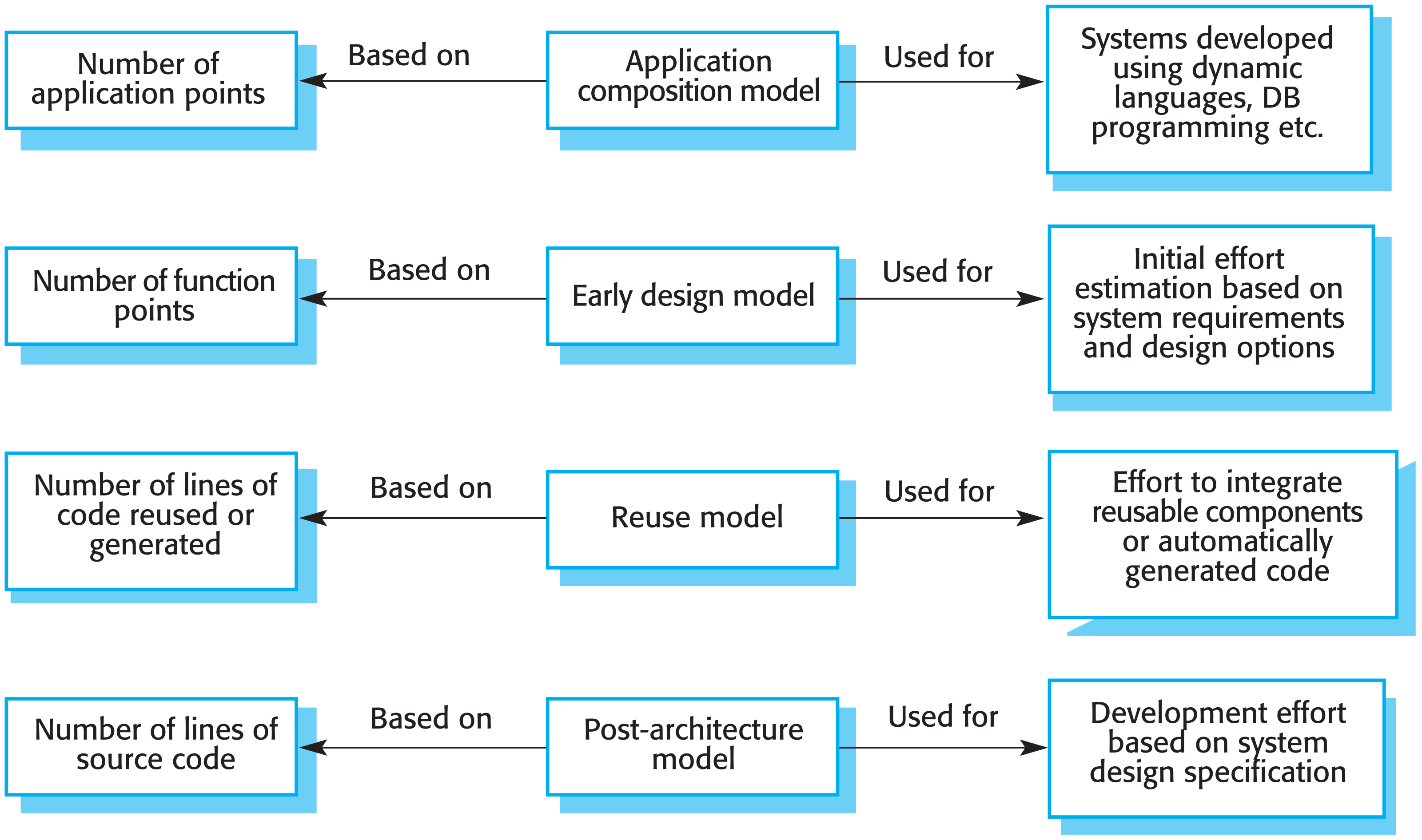
Algorithmic cost modeling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - ***Effort* = A × Size^{B} × M**
  - A is an organization-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes.
- The most commonly used product attribute for cost estimation is code size.
- Most models are similar but they use different values for A, B and M.
- Algorithmic cost models are a systematic way to estimate the *effort* required to develop a system. However, these models are complex and difficult to use, and so mostly are used only by large companies.
- And even then, there are many attributes and considerable scope for uncertainty in estimating their values.

COCOMO cost modeling
(Constructive Cost Model)

- An algorithmic, empirical model based on project experience.
  - Well-documented, ‘independent’ and not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc. and has 4 sub-models:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not yet started.
  - Reuse model. Used to compute the *effort* of integrating reusable components.
  - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

COCOMO estimating models

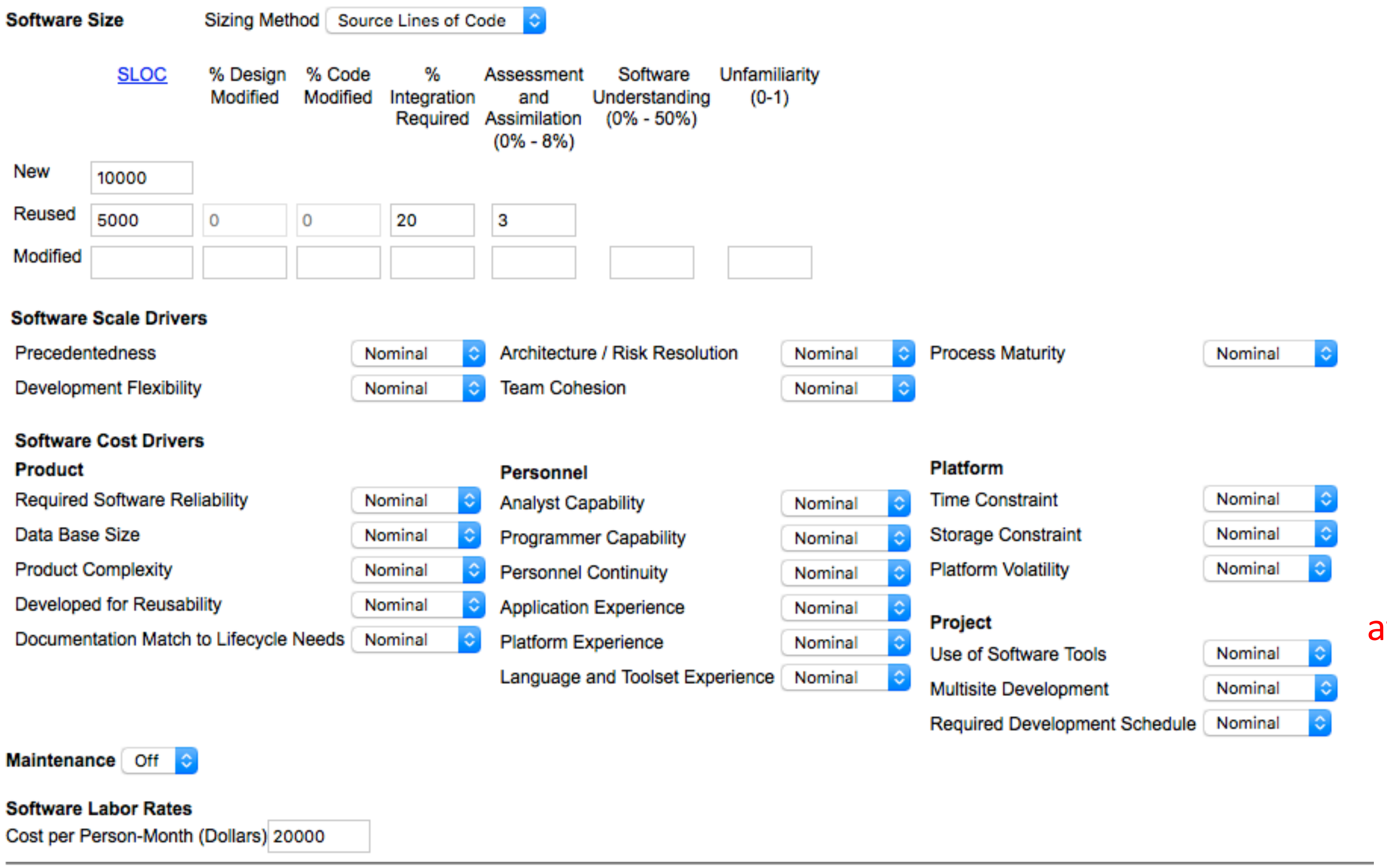


Number of application points
Based on
Application composition model
Used for
Systems developed using dynamic languages, DB programming etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

Described in great detail in the Sommerville textbook.

COCOMO II online tool

- http://csse.usc.edu/tools/COCOMOII.php



Software Size Sizing Method: Source Lines of Code

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New	10000						
Reused	5000	0	0	20	3		
Modified							

Software Scale Drivers

Precedentedness	Nominal	Architecture / Risk Resolution	Nominal	Process Maturity	Nominal
Development Flexibility	Nominal	Team Cohesion	Nominal		

Software Cost Drivers

Product		**Personnel**		**Platform**	
Required Software Reliability	Nominal	Analyst Capability	Nominal	Time Constraint	Nominal
Data Base Size	Nominal	Programmer Capability	Nominal	Storage Constraint	Nominal
Product Complexity	Nominal	Personnel Continuity	Nominal	Platform Volatility	Nominal
Developed for Reusability	Nominal	Application Experience	Nominal	**Project**	
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	Nominal	Use of Software Tools	Nominal
		Language and Toolset Experience	Nominal	Multisite Development	Nominal
				Required Development Schedule	Nominal

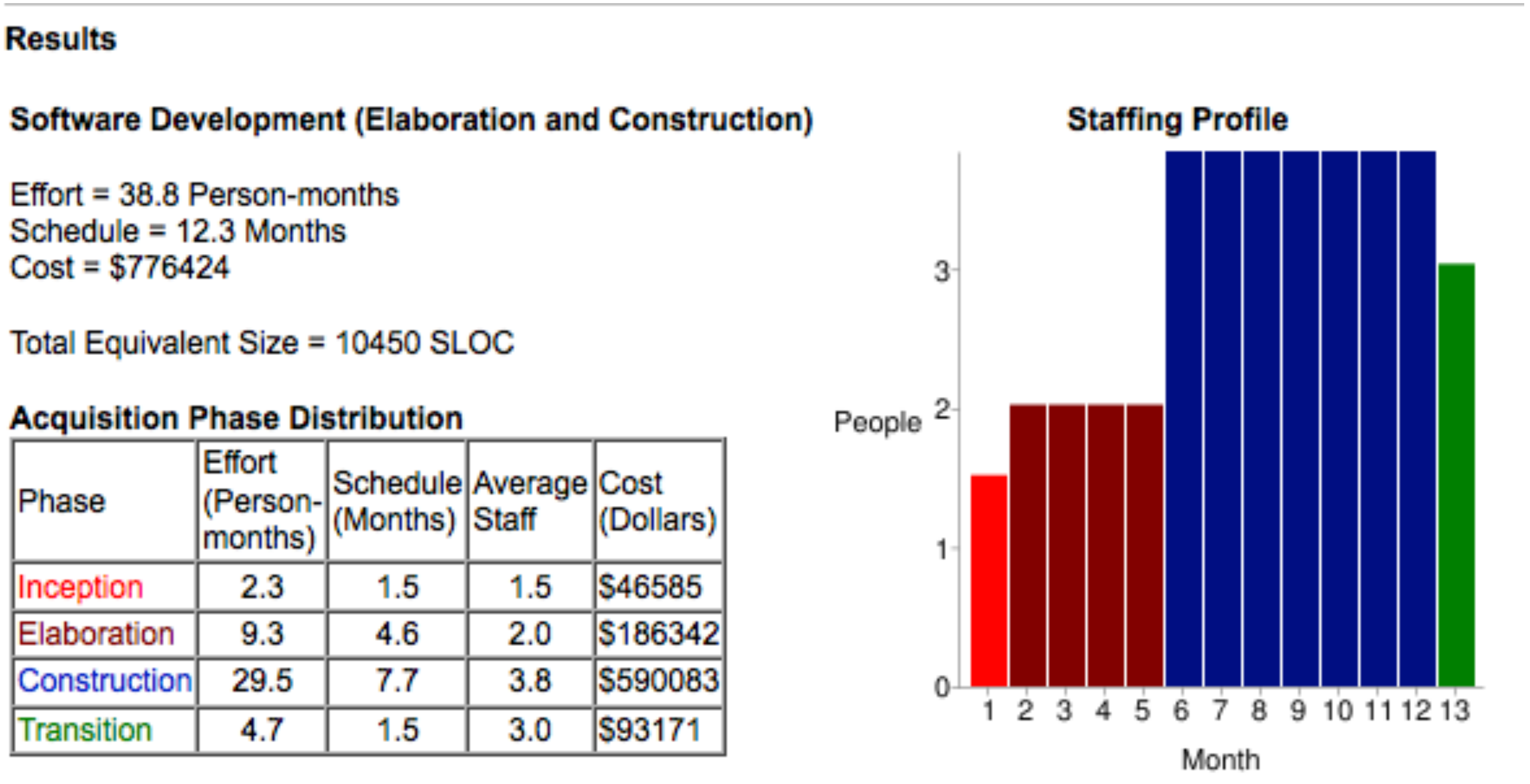
Maintenance Off

Software Labor Rates
Cost per Person-Month (Dollars) 20000

B factors

M attributes

Tool results



Results

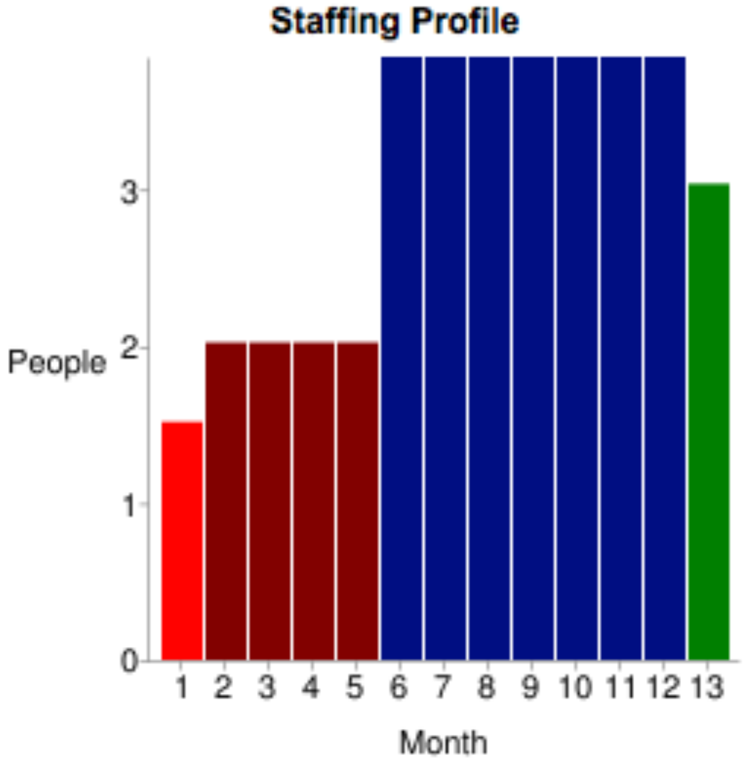
Software Development (Elaboration and Construction)

Effort = 38.8 Person-months
Schedule = 12.3 Months
Cost = $776424

Total Equivalent Size = 10450 SLOC

Acquisition Phase Distribution

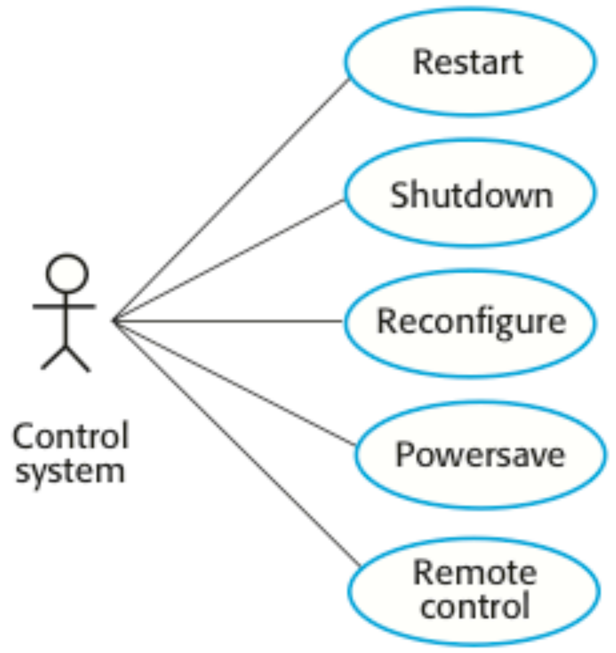
Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	2.3	1.5	1.5	$46585
Elaboration	9.3	4.6	2.0	$186342
Construction	29.5	7.7	3.8	$590083
Transition	4.7	1.5	3.0	$93171



A more contemporary estimation technique is Use Case Points

- Use case points (UCP) are a method for estimating the *effort* required to build software
- Inputs are the detailed use cases showing for all use cases in the software system
  - A weighted count by type of unique actors across all use cases (UAW)
  - A weighted count of use cases, categorized by the number of steps required to carry out a successful case, and, if known, number of object classes needed (UUCW)
  - Factors that add technical complexity to the project (TCF)
  - Factors that add "environmental" complexity to the project (ECF)
  - A productivity factor (PF)
- **UCP = (UAW + UUCW) * TCF * ECF; *Effort* = UCP * PF**

Example project: Control System



	Actors			
Use Cases	Cntl Sys Admin	Cntl Sys Ops	Comm System	# of steps
Restart		√	√	3
Shutdown		√	√	3
Reconfigure	√		√	10
Powersave			√	2
Remote Control	√	√	√	10

Categorize and weight Use Cases

Category	Weight	Description	EX
Simple	5	At most 3 steps in the success scenario; *at most 5 classes in the implementation*	3 =15
Average	10	4 – 7 steps in the success scenario; *5-10 classes in the implementation*	
Complex	15	More than 7 steps in the success scenario; *more than 10 classes in the implementation*	2 =30
			Tot= 45

A count is made for each category of use case and that count is multipliec by the category weight to get an unadjusted use case weight (**UUCW**).

Actor categories

Category	Weight	Description	EX
Simple	1	Actor represents another system with a defined API	1 = 1
Average	2	Actor represents another system interacting through a protocol	
Complex	3	Actor is a person interacting through an interface	2 = 6
			Tot 7

A count is made for each category of actor and that count is multiplied by the category weight to get a unadjusted actor weight (**UAW**). The unadjusted use case points (**UUCP**) **= UUCW + UAW**.

Technical complexity factor (TCF)

- TCF captures some of the difficulties of implementing the non-functional requirements
- A number of factors are evaluated, for example:
  - Is concurrency required?
  - Will the software be part of a distributed system?
  - Are special security implementations needed?
  - Etc.
- The factors reflect characteristics of more modern, non-transactional systems as contrasted with function point analysis.
- TCF may reduce or enlarge the nominal effort measured by UUCP by about 40%

Technical Complexity Factors (TCF)

TCF = 0.6 + (0.01 x Σ W_i * F_i), where F is **0 to 5** and TCF ranges from 0.6 to 1.3.

Technical requirement	Weight
Distributed System	2
Performance	1
End User Efficiency	1
Complex Internal Processing	1
Reusability	1
Easy to Install	0.5
Easy to Use	0.5
Portability	2
Easy to Change	1
Concurrency	1
Special Security Features	1
Provides Direct Access for Third Parties	1
Special User Training Facilities Are Required	1

Example system's TCF

Technical requirement	Weight	Rating	Total
Distributed System	2	5	10
Performance	1	4	4
End User Efficiency	1	2	2
Complex Internal Processing	1	1	1
Reusability	1	1	1
Easy to Install	0.5	2	1
Easy to Use	0.5	2	1
Portability	2	0	2
Easy to Change	1	0	0
Concurrency	1	3	3
Special Security Features	1	5	5
Provides Direct Access for Third Parties	1	0	0
Special User Training Facilities Are Required	1	0	0

$TCF = 0.6 + (0.01 \times \Sigma W_i * F_i)$

TCF = 0.6 + (0.01 x 30) = 0.6 + 0.30 = **0.90**

Environmental Complexity Factors (ECF)

- ECF focuses on the influence that the development team will have on the effort
- The more experienced, more capable, more committed a team is, the faster the development is likely to go
- ECF also examines two other areas of development
  - How stable the requirements are
  - How difficult the programming language is
- ECF does not reflect the requirements (TCF does), but rather the implementers and the environment in which implementation will take place

Environmental complexity factors

Description	Weight
Familiarity with UML	1.5
Part-time workers	-1
Analyst capability	0.5
Application experience	0.5
Object-oriented experience	1
Motivation	1
Difficult programming language	-1
Stable requirements	2

$$ECF = 1.4 - (0.03 \times \Sigma\ W_i * F_i)$$

where F_i is **0 to 5** and ECF ranges from 0.725 to 1.4

Example system's ECF

Description	Weight	Rating	Total
Familiarity with UML	1.5	1	1.5
Part-time workers	-1	5	-5
Analyst capability	0.5	3	1.5
Application experience	0.5	0	0
Object-oriented experience	1	1	1
Motivation	1	5	5
Difficult programming language	-1	0	0
Stable requirements	2	5	10

ECF = $1.4 – (0.03 \times \Sigma W_i * F_i)$

ECF = $1.4 – (0.03 \times 14) = 1.4 - 0.42 = \mathbf{0.98}$

UCP metric

- Unadjusted use case points (unadjusted actor weight plus unadjusted use case weight) is
- adjusted for technical complexity factors (adjustment factor can range from 0.6 to 1.3) and
- adjusted for environmental complexity factors (adjustment factor can range from 0.42 to 1.55).
- This yields a UCP.

Example: UCP = 52 * 0.9 * 0.98 = ~46

- UCP has little meaning unless we know how many use case points a development team can develop in a fixed amount of time. That's where PF, the productivity factor, comes in.

Productivity factor (PF) and UCP calculation

- A PF (Productivity Factor) is like a Velocity – it captures how productive a team is (or probably will be).
  - The Productivity Factor (PF) is a ratio of the number of staff hours per use case point based on past projects. It is expressed as “hours needed to complete a Use Case Point” so the higher the PF, the lower the productivity!!!
  - A team that needs 30 hours to complete a use case point is only half as fast as one needing 15!
- With no historical data,
  - Industry experts suggest a figure between 15 and 30 hours.
  - A typical PF value is 20 hours per use case point.
- Total Estimate of STAFF HOURS = UUCP x TCF x ECF x PF

Using a PF of 20 in our example, 46 x 20 = 920 staff hours = ~25 staff weeks = 6 to 7 staff months.

Summary:
Use Case Points (UCP)

- Use Case Points allow us to estimate development effort based on the use cases we have defined for our project.
- Different actors may require different interfaces so UNIQUE actors are an important contributor to total effort.
- Different goals will require different system solutions so unique Use Cases (each achieving a goal) form another important contributor.
- Technical complexity factors must be accommodated in our estimates as must "environmental" complexity factors.
- Productivity (defined as hours of effort needed per Use Case Point) will differ from team to team.
- For a detailed walk-through of an example, look at the Clemmons paper in this week's Canvas module.

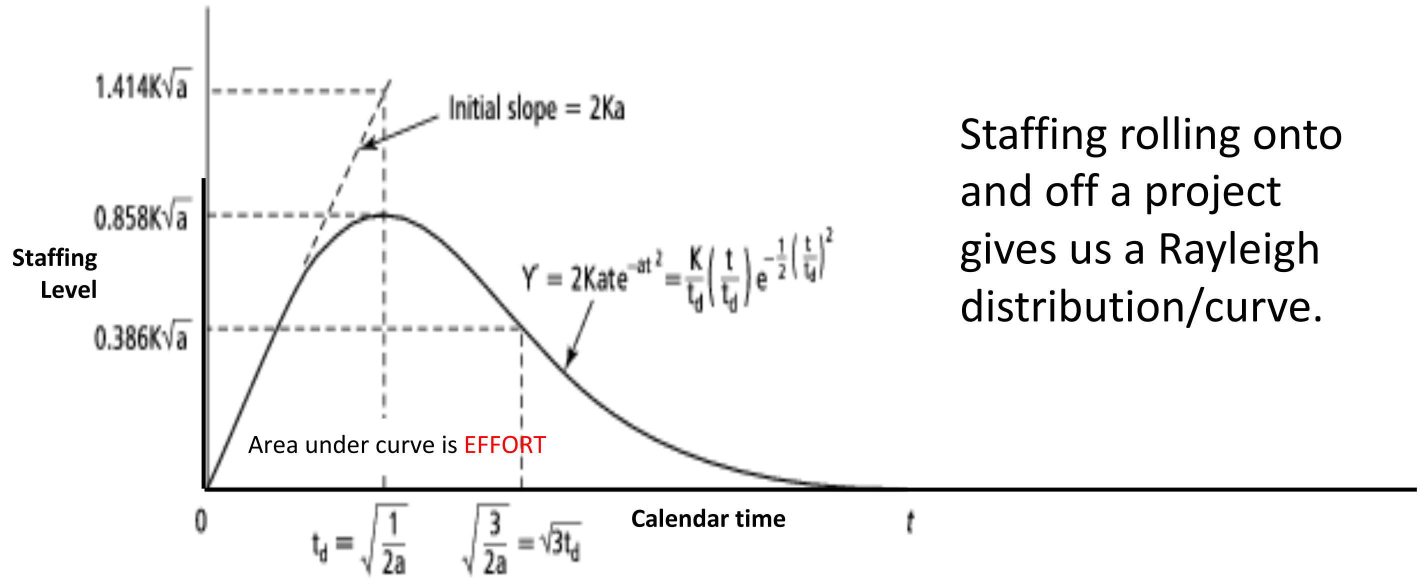
Brooks' Mythical Man Month - Risks

- Usually projects fail for lack of calendar time.
  - Estimation is optimistic, last bug syndrome.
  - Costs vary by staff months, progress does not.
  - Sequential nature of some tasks.
  - Communication is key difference.
    - Farm workers vs. developers.
    - Each new worker must be trained by experienced staff.
    - Brooks' communication heuristic n(n-1)/2.
- Testing is the most poorly scheduled part.
  - Never enough time to adequately test.
  - Development runs late and test eats it.

Estimating staffing requirements

- Staff months is an estimate of budget
  - Not calendar time
  - Not staffing levels
- The number of people working on a project varies depending on the phase of the project and the skills of the people.
- The more people who work on the project, the more total effort is usually required.
  - The more people on a team, the more people with whom each team member must communicate.
- A very rapid build-up of people often correlates with schedule slippage.
  - Brooks' Law: Adding people to a late project makes it later!

Staffing impacts calendar time



Staffing rolling onto and off a project gives us a Rayleigh distribution/curve.

- *Effort* (staff months) does not spread uniformly across time.
- At the project start, a small number of engineers is needed.
- As the project progresses, more work is required that needs more staff. At one point, the number of staff peaks (often system test).
- It then drops down (development finish, installation and delivery).

Estimating calendar time

- COCOMO 2 has a formula for estimating calendar months from *effort*.
  - First convert your *effort* estimate to staff months (I assume a 35 hour week or 150 staff hours per month).
  - Then apply the approximate calendar time formula from COCOMO 2.

$$\mathbf{TDEV = 3 \times \mathit{Effort}^{1/3}}$$

- **The actual COCOMO formula is** TDEV = 3 × $\mathit{Effort}^{(0.33+0.2*(B-1.01))}$, where B is based on the COCOMO II scale factors and has the effect of representing the diseconomy of scale.
- Most critically, calendar time is **NOT** estimated as a function of staffing levels—only *effort*.

Some sample calculations

Staff-months	Calendar-months
5	5.1
10	6.5
15	7.4
20	8.1
30	9.3
40	10.3
50	11.1

- Note:
  - These estimates are computed independently of # of staff members, but clearly more staff will be needed/used when greater *effort* is required.

Caveats for *Effort* estimation

- The best estimates, even when algorithmic, are guided by experience.
- Observing and understanding organization-specific productivity is critically important is all estimation activity.
- UCP estimates are strongly influenced by the detail level of the use cases
    - the more steps, the higher the UCP
    - the more actors, the higher the UCP
- Without experience-based tweaking, UCP estimates tend to be high compared to those of experts, unless organization specific PF's are applied.

So you’ve estimated your cost, what about pricing?

- Estimates of the total cost to the developer of producing a software system take into account, hardware, software, travel, training and, of course, *effort* costs.
- But there is no simple relationship between the development cost and the price charged to the customer.
- Broader organizational, economic, political and business considerations influence the price charged including, but not limited to
    - Contractual terms
    - Cost estimate uncertainty
    - Requirements volatility
    - Financial health of the developing company/organization
    - Market opportunity

Classroom Activity – within your EMSS project teams

Activity:

Provide an estimate of the effort and calendar time (project duration) needed to deliver the EMSS software using Use Case Points (for effort), applied to your Use Cases, and the COCOMO II formula for TDEV (calendar time).

Submission to complete the Canvas Assignment EMSS Week 9:

Submit your estimates, showing how you arrived at them.

Note: When estimating UCP, remember that each actor (role) is counted only once, regardless of how many use cases they participate in. Their value represents the unique aspects of their interface.