06: Managing time (1)

Software Project Management

Philippe Kruchten

Module outline

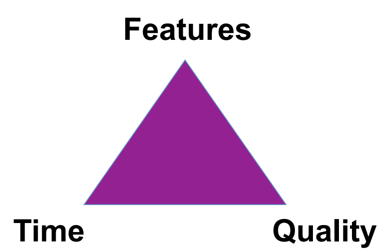
- Effort & Schedule
- Estimation – the black magic
  - Function Points and Use case points
  - COCOMO, SLIM, etc.
  - Wideband Delphi, planning poker
- Time line
  - Phases and Iterations
  - Milestones

Effort and Schedule

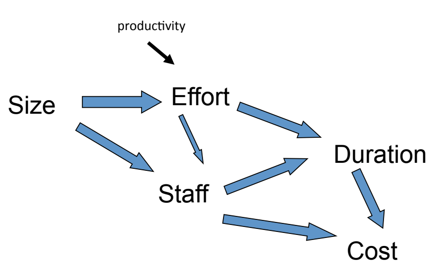
Two questions you will have to answer as a software project manager:

- When will we be done?
  - Schedule
- How much will it cost?
  - Effort

Our Fundamental Conundrum



Entangled Variables



Size, First, then Effort…

- How big is that mountain?
- Units of measure:
  - Functional size: function points, and variants
  - Code size: Lines of code
- Effort: staff-month
- Top-down approaches: analytical
  - Example: COCOMO
- Bottom-up approach: aggregations

… then Schedule

$Duration_{months} = Effort_{staff*month} / Resource_{staff}$

- Cannot trade people and months
  - 7 months with 8 people is not the same as
  - 8 month with 7 people
- Brook's Law:
  - Adding people to a late project make it later

Estimation

- Estimation of Size, Effort, Schedule is the most difficult area of *software* development management.
- We know so very little about size estimation, it is almost embarrassing.
- Black Magic …

Estimation

- Estimate *vs.* measure
- Estimate *vs.* commitment

Estimation

Two main approaches:

- Estimation by analogy
- Estimation by proxy

Example

- By analogy: how long will it take?
  - On average it took us 24 person-months to develop a new "BLA" so far (16 data points)
  - This is like a normal "BLA", with some niceties, so let us say 28 person-months
- By proxy: how high is this building?
  - Using the shadow…?

Small experiment: How good an estimator are you?

1. Year Elvis Presley was born
2. Length of the Nile river
3. Latitude of Shanghai
4. Year Gutenberg press started to operate
5. Volume of water in the Great lakes
6. Distance Vancouver – St. John's (NFL)
7. Year Alexander the Great was born
8. Temperature of the surface of the sun

Small experiment: How good an estimator are you?

1. Year Elvis Presley was born (1935)
2. Length of the Nile river (6650 km)
3. Latitude of Shanghai (31° N)
4. Year Gutenberg started to print (1436)
5. Volume of the Great lakes (23000 km^3)
6. Distance Vancouver St. John's (5,010 or 7,400 km)
7. Year Alexander the Great was born (356 BC)
8. Temperature of the surface of the sun (6000°C)

Source Lines Of Code (SLOC)

- Used for many years as a poor substitute for Functional Size
- KSLOC = kilo-SLOC = 1000 lines of code
- KDSI = Kilo Delivered Source Instructions
- No comments, no test data
  - Line in Fortran and Cobol
  - Semi-colons in more recent languages: C, Java, etc.

SLOC: Pros and Cons

+ Easy to measure, unambiguous
+ Simple indicator to monitor progress
+ We have productivity estimates data
  – per language and type of application
- Accounting for rework is hard
- More difficult to use because of new technologies
  – reuse, COTS, model-driven design, patterns
  – Gui builders, application generators

Productivity

- SLOC / Staff-month
- Not coding speed !!!
- For the whole project and all activities together, including those not related to code, such as Project Management.
- Any guesses?

Productivity (SLOC/Staff-Month)

Automation	245 [120..440}
Command and control	225 [95..330]
Data Processing	330 [165 .. 500]
Software tools	260 [143 .. 610]
Military, spatial	100 [45 .. 300]
Scientific	195 [130 .. 360]
Telecommunications	250 [175 .. 440]
Simulations, trainers	224 [143.. 780]
Web	275 [190 .. 975]
Over 500 projects	[45 .. 975]

Source: Reifer 2002

Functional size

- A size of software derived from quantifying the functional user requirements (FUR)
- FUR examples:
  - Data movements
  - Data manipulations

ISO 14143:1997 – Software measurement - Functional Size – Definition of concepts

FURs

- Data in
- Storage
- Transformation
- Retrieval
- Data out

Function Points

- Started by Allan Albrecht (IBM) in 1979
- Consistent measure of software size
  - independent from technology used
- IFPUG = International Function Point Users' Group
  - www.ifpug.org
- FPA: Function Point Analysis
- Counting Practice Manual, v 4.1 US$75.00
- Certification Program

FP

- Weighted sum of Input, Output, Enquiries, Files…

		simple	medium	complex	
# input	X_1	3	4	6	$C_1 X_1$
# output	X_2	4	5	7	$C_2 X_2$
# enquiry	X_3	3	4	6	$C_3 X_3$
# file	X_4	7	10	15	$C_4 X_4$
# ext i/f	X_5	5	7	10	$C_5 X_5$

$\sum C_i X_i$

From FP to SLOC

- Cobol: 106 sloc/fp
- Ada: 71 sloc/fp
- C: 150 sloc/fp
- Assembler: 320 sloc/fp
- etc…

Full Function Points

- COSMIC: Common Software Measurement International Consortium
  - Alain Abran (ETS, UQAM), Charles Symons, etc.
- www.cosmicon.org
- Cosmic-FFP Measurement Manual, v 2, 1999
  - Free from UQAM
  - ISO standard 19761
- Open, simpler, more widely applicable
  - MIS *and* real-time

Functional Size Unit

- 1 data movement = 1 Cosmic Functional Size Unit (cfsu)
- 4 types of functional units:
  - Entry, Exit, Read, Write
  - No data manipulation
- Example: Create new employee
  - 1 entry
  - 1 write
  - 1 exit: confirmation or error

  Total: 3 cfsu

Functional Size Unit

- Example: Control temperature a regular interval
  - 1 entry (clock tick)
  - 1 entry (temperature reading)
  - 1 read (target temperature range)
  - 1 exit (on or off command to the heater)

  Total: 4 cfsu

Function Points: Summary

- Need rather detailed requirements specs, and somewhat an embryo of a design (architecture)
- Difficult to master, in spite of the apparent simplicity of the technique
- Less dependent on technology, language, etc.
- Not widely used
- Variants: Feature points, Use Case points

Other approaches to sizing

- Case-Based Reasoning
- In-house database of completed projects
- Expert opinion
- Wideband Delphi

- Build to size
  - Fix the effort, and build whatever fits inside this envelope

Wideband Delphi

- Delphi: Rand Corp. 1960 (??)
- Expert opinions, feedback, iteration, until consensus is reached.

- Difference
  - Delphi: no group discussion
  - Wideband Delphi: group discussion to speed up the consensus, to understand the variance

Steps in Wideband Delphi

1. Present a problem to panel of experts
2. Conduct group discussion
3. Collect expert opinions (anonymously)
4. If consensus, exit
5. Feedback summary of results to each expert
6. Go to step 2.

Estimate

Evaluation sheet: Example

Project: Operating System *Date :* 6/6/2000

Range of estimates after the 3rd round
Your estimate *25 PM*
Median estimate *45 PM*

Your estimate for the next round: 35 PM
Rationale for your estimate :
Looks like a standard process control operating system. Our people have had lots of experience with such systems. They should have no trouble with this one. I am increasing my estimate to account for the new DMA channel mentioned by one of the estimators.

Evolution of Estimates

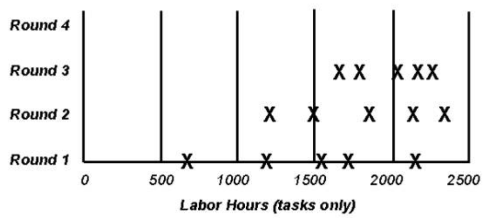


Figure 4. Estimation chart showing three rounds from a Wideband Delphi session.

Jan. 2014 34

Wideband Delphi: Pros and Cons

Pros
- Easy and inexpensive
- Educative value as a side effect

Cons
- May reach a consensus on an incorrect estimate
- Not very repeatable
- False sense of confidence

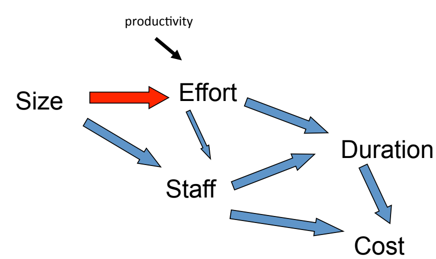
Jan. 2014 35

Agile Practice: Planning poker

- Grenning 2002
- Not very different than Wideband Delphi
- 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100
  – Why not other numbers?
- Process
  – Read description
  – Bid
  – Discuss discrepancies (2 min)
  – Re-bid

Jan. 2014 36

From Size to Effort



Jan. 2014 37

From Size to Effort

- E = f(S) ?
- Models
  – COCOMO
  – COCOMO II
  – SLIM
  – Price-S

Jan. 2014 38

Parametric tools: COCOMO

- COnstructive COst MOdel
- Barry Boehm, TRW, 1981
- A regression model
  – identify some variables
  – regression on a database of known projects
  – find useful relationships on these variables
  – use them to make predictions for other projects
  – grow the database and refine the model

Jan. 2014 39

COCOMO Modes = Types of Projects

1. Organic
   – payroll, inventory, scientific
   – small team, little innovations required
   – few constraints
2. Semidetached
   – Compilers, database systems, OS, etc…
   – medium size
3. Embedded
   – real-time constraints, innovation, complex interfaces

Jan. 2014 Copyright © 2005-14 by KESL 40

COCOMO: 3 levels of sophistications

- Basic COCOMO
- Intermediate COCOMO
  – Bring in some cost drivers
- Detailed COCOMO
  – Program decomposed into major components
  – Process decomposed in phases

Jan. 2014 Copyright © 2005-14 by KESL 41

Basic COCOMO Effort Formula

$$\text{Effort (E)} = a \times (\text{Size})^b$$

where

- *a* and *b* are constants derived from regression analysis
- Size in KSLOC
- E in staff-month (19 days/month)

Jan. 2014 Copyright © 2005-14 by KESL 42

Basic COCOMO Effort

- Organic:
  $E = 2.4 \times (\text{Size})^{1.05}$
- Semi-detached:
  $E = 3.0 \times (\text{Size})^{1.12}$
- Embedded:
  $E = 3.6 \times (\text{Size})^{1.20}$

Jan. 2014 Copyright © 2005-14 by KESL 43

Basic COCOMO Effort

- Project of 200 KLOC
  - That's a large project
- Organic:
  $E = 2.4 \times (200)^{1.05} = 626$ staff-months
- Semidetached:
  $E = 3.0 \times (200)^{1.12} = 1{,}133$ staff-months
- Embedded:
  $E = 3.6 \times (200)^{1.20} = 2{,}077$ staff-months

Jan. 2014 Copyright © 2005-14 by KESL 44

Basic COCOMO: Time to develop

- Organic:
  – $T_{dev} = 2.5\ (E)^{0.38}$ months
- Semidetached:
  – $T_{dev} = 2.5\ (E)^{0.35}$ months
- Embedded:
  – $T_{dev} = 2.5\ (E)^{0.32}$ months

Jan. 2014 Copyright © 2005-14 by KESL 45

Basic COCOMO: another example

- 7.5 KSLOC, simple (organic)

Effort: $E = 2.4 \times (7.5)^{1.05} = 20$ staff-months
Duration: $T_{dev} = 2.5(E)^{0.38} = 8$ months
Average staff: $N = E / T_{dev} = 2.5$ staff aver.
Productivity: $P = S / E = 375$ LOC/staff-month

Intermediate COCOMO: Cost Drivers

- Boehm analyzed the source of variance around the basic predictions of his model
- Found 15 variables that affect the effort
- New formula:

$$\text{Effort (E)} = a \times (\text{Size})^b \times C$$

- C = Effort Adjustment Factor (EAF)
  - C = C1 x C2 x… x Cn

Cost Drivers

- Product related
  - RELY – required reliability
  - DATA – size of database
  - CPLX – Complexity, constraints
- Computer related
  - TIME – execution time constraints
  - STOR – main storage constraint
  - VIRT – Virtual machine volatility
  - TURN – Computer turnaround time

Cost Drivers (cont.)

- Project related
  - ACAP – Analyst capability
  - AEXP – Application experience
  - PCAP – Programmer Capability
  - VEXP – Virtual Machine Experience
  - LEXP – Programming language experience
- Personnel related
  - MODP – use of modern techniques
  - TOOL – use of modern software tools
  - SCED – development schedule pressure

Cost Drivers Ratings

Cost Driver Rating

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	.75	.88	1.0	1.15	1.40	
DATA		.94	1.0	1.08	1.16	1.65
CPLX	.70	.85	1.0	1.15	1.30	1.65
TIME			1.0	1.11	1.30	1.66
Etc.	…	…	…	…	…	…

COCOMO Cost drivers

Cost Drivers	*Very Low*	*Low*	*Nominal*	*High*	*Very High*	*Extra High*
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility*		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience*	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
Project Attributes						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

Intermediate COCOMO Example

- 10 KSLOC, embedded software
- Cost drivers all nominal except:
  - DATA: not much data, Low -> 0.94
  - TIME: some constraints, High -> 1.11
  - ACAP : good analysts, High -> 0.86
  - PCAP: Good programmers, High -> 0.86
  - VEXP: low -> 1.10
  - etc…
- EAF = 1.17

Intermediate COCOMO Example (cont.)

- 10 KSLOC embedded software
  - $E = 2.8 (10)^{1.20} = 44$ staff-months
- EAF = 1.17
  - E = 44 x 1.17 = 51 staff-months

Intermediate COCOMO Example (2)

- With all factors at nominal, a project is estimated to be 44 staff-months.
- You hire more capable staff, both analysts and developers, and the ratings goes from 1.0 to 0.86, but staff cost increases from $5,000 to $6,000 per staff-month.
- Adjustments: 44 x 0.86 x 0.86 = 32.6 sm
- Cost differential:
  - $220K – $195K = $24.4 (improvement: 11%)

COCOMO Pros and Cons

Pros
- Versatile and Repeatable
- Can be backfitted from real programs
- Easy to use

Cons
- Somewhat outdated calibration
- Assume waterfall-like development
- Focuses only on development

Aside: Diseconomy of scale

- Effort = A x $Size^B$
- B > 1: diseconomies of scale
- B = 1: linear
- B < 1: economies of scale

COCOMO II: 1999

Main objectives of COCOMO II:
- To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's
- To develop software cost database and tool support capabilities for continuous model improvement

COCOMO II: 3 models

- The Application Composition Model
  - Good for projects built using rapid application development tools (GUI-builders etc)
- The Early Design Model
  - This model can get rough estimates before the entire architecture has been decided
- The Post-Architecture Model
  - Most detailed model, used after overall architecture has been decided on

COCOMO II differences

- The exponent value b in the effort equation is replaced with a variable value based on five scale factors rather then constants
- Size of project can be listed as object points, function points or source lines of code (SLOC).
- EAF is calculated from seventeen cost drivers better suited for today's methods, COCOMO81 had fifteen
- A breakage rating has been added to address volatility of system

COCOMO 2: Parameters & Potential Impacts

Complexity Parameters	Impact	Process Parameters	Impact
Size (new, reuse, integration)	Variable	Documentation match to needs	52%
Operational complexity	138%	Process maturity	43%
Timing constraints	63%	Schedule constraints	43%
Reliability	54%	Risk resolution (Architecture-first)	39%
Storage constraints	46%	Precedentedness	33%
Data base size/scope	42%	Development flexibility	26%
Reusability	31%	Team cohesion	29%

Team Parameters	Impact	Tools Parameters	Impact
Analyst capability	100%	Multi-site development	53%
Programmer capability	76%	Use of software tools	50%
Applications experience	51%	Platform volatility	49%
Personnel continuity	51%		
Language/Tool experience	43%		
Platform experience	40%		

"Impact" means the range of potential impact between the lowest and highest settings for this parameter

SLIM

- Peter Norden (IBM, 1960s)
- Defined the Norden-Rayleigh Function
- Lawrence Putnam (QSM, 1970) used regression to derive a predictor: Software Lifecycle Management (SLIM)

SLIM: Putnam Equation

$$S = C \times K^{1/3} \times T_d^{\ 4/3}$$

where

- S = size in SLOC
- C = an environmental factor
- K = total life-time effort
- T_d = delivery time constraint in years
- development effort: B = 0.39 K
- Compute your C from previous projects
  - 2000=poor, 8000=good, 11000=excellent
  - real-time: 1500

SLIM: example

- Project of 200 KLOC, with a C of 4000
- Total lifetime effort:
  $K = (1/T^4)(S/C)^3$
  $K = (1/T^4)(200000/4000)^3$
  $K = (1/T^4)(50)^3$
  K = 7812 staff-month (in 2 years)
- Development effort
  B = 0.39 K = 3100 sm; staff : 130 persons

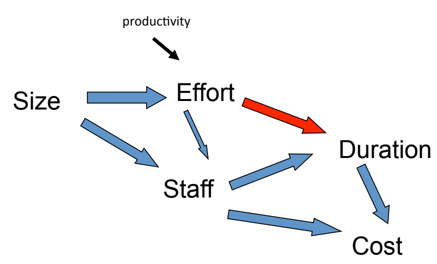
SLIM: Pros and Cons

- Pros
  - Complete tool by QSM is very sophisticated
  - Lots of know-how embedded in the tool
  - Allows calibration with local data
  - Covers complete project
- Cons
  - Works only on large systems
  - Size must be known
  - Estimates very sensitive to technology factor
  - Estimates very sensitive to time factor

McConnell's Table

	System Product		Business Product		Shrink Wrap Product	
KSLOC	Dur	Eff	Dur	Eff	Dur	Eff
10	10	48	6	9	7	15
25	15	140	9	27	10	44
50	20	360	11	71	14	115
100	26	820	15	160	18	270
200	35	1900	20	370	24	610
400	47	4200	27	840	32	1400

From Effort to Duration

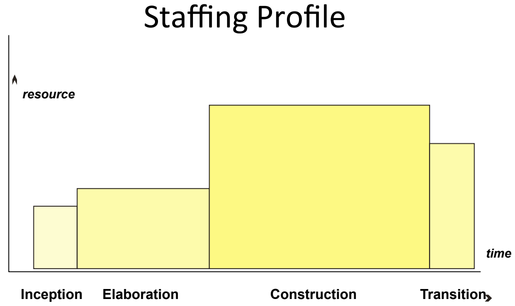


From Effort to Duration: Staffing

- Typical, greenfield development
- First cycle
- Some risks and no predefined architecture

	Inception	Elaboration	Construction	Transition
Effort	5%	20%	65%	10%
Duration	10%	30%	50%	10%

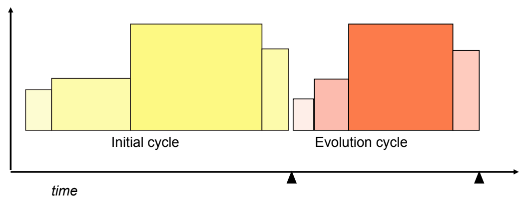
Staffing Profile



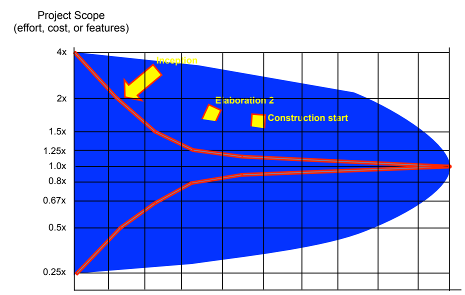
Variations

- Many risk, lost of unknowns
  - Longer elaboration
- No risks, architecture in place
  - Shorter elaboration
- Commercial product
  - Longer transition
- Certification issues
  - Very long transition
- Initial funding issue
  - Long inception

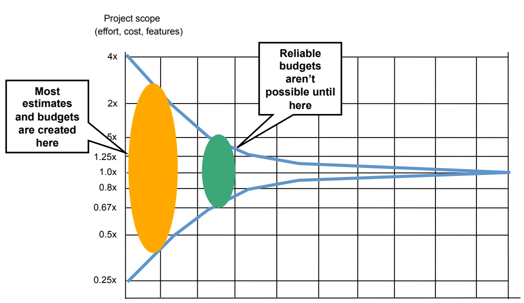
Evolution Cycle



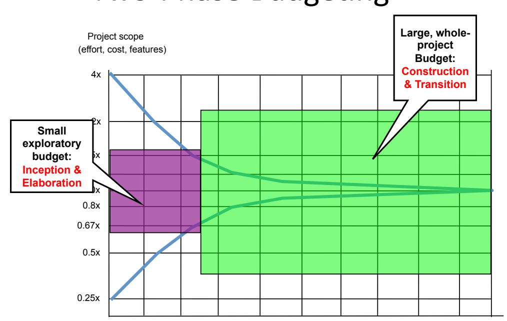
Cone of Uncertainty and Estimate Refinement



When to define budget?



Two-Phase Budgeting



Benefits of Two-Phase Budgeting

- Delays commitment until time when a commitment can be meaningful
- Forces activities that should occur upstream actually to occur upstream
  - Requirements, technical planning, quality planning, risk mitigation, architecture validation, etc.
- Helps set realistic expectations for all project stakeholders
- Improves coordination with non-software groups
- Improves execution by putting plans on more informed basis

Iteration Planning

- One iteration
  - Results in one executable version
  - Mitigates some risk
  - Implement ssome “stuff”
  - Builds up on previous iterations
  - Allow some feedback
    - Product
    - Process
    - People

Duration of an Iteration

- Duration driven by
  + size of organization
  + size of project
  - familiarity with the process, maturity
  - technical simplicity

For large projects, I use the rule of thumb:

$$W = \sqrt{(S/1000)}$$

- W in weeks
- S in SLOC

Examples

- 5 people, 5 weeks:
  - 1 week iterations
- 10 people, 6 to 12 weeks
  - 2 week iteration
- 130 people, 5 years
  - 3 to 6 month iterations

Iterations: How Many?

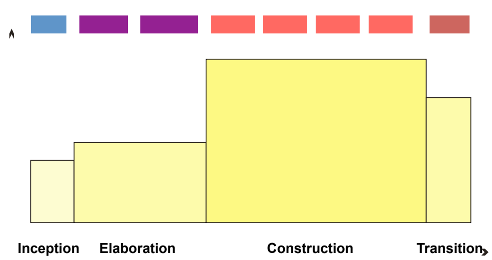
- 6 plus or minus 3

Inception : 0 .. 1
Elaboration : 1 .. 3
Construction : 1 .. 3
Transition : 1 .. 2

Iterations: How Many?

- Very efficient organizations:
  - Short iterations
  - as many as 15 in a single project
- Do not confuse “builds” with iterations
  - Nightly build, weekly build
  - Iteration involves more of the lifecycle activities

Phases and Iterations



[1, 2, 4, 1]

Capturing the Information

- Estimation hypothesis
- Estimations
- Duration =>
- Major milestones
- Iteration length =>
- Minor milestones

Capturing the Information

- Schedule
- Goes into Software Development Plan
  - 1.1.2 assumption and constraint
  - 1.1.4 Schedule and budget summary
  - 5.1.1 Estimation plan
  - 5.1.2 Staffing plan
  - 5.2 work plan
  - etc…

Worked out Estimation Example

- Assumptions:
  - Small Project,
  - Business support,
  - Few technical Risk,
  - Greenfield development,
  - Distributed system (internet based)
- Start from Use Cases
- Using Use Case Point

Unadjusted Use-Case Points UUCP

- 1 Complex Use Case
- 5 Medium Use Cases
- 6 Simple Use Cases
- Weight:
  - Complex :15
  - Medium: 10
  - Simple: 5
- UUCP = 1 x 15 + 5 x 10 + 6 x 5 = 95

Adjust for Technical Complexity

- Distributed 2 x 5
- Responsive 1 x 2
- Simple 1 x 0
- etc…
- TCF = 0.6 + (0.01 x TFactor)
- TCF = 0.6 + (0.01 x 32) = 0.92

Adjustment for Environment

- Familiar with internet 4 x 1.5
- Familiar with the application 4 x 0.5
- Stable requirements: 2 x 2
- etc…
- EF = 1.4 + (-0.03 x EFactor)
- EF = 1.4 - 0.03 x 25 = 0.65

Adjusted Use-Case Points

- UCP = UUCP x TCF x EF
- UCP = 95 x 0.92 x 0.65 = 57

Effort

- 20 to 28 staff-hour / UCP
  - Here some local data helps!
- 20 * 57 = 1140 staff-hour
- = 163 staff-day
- = 8.5 staff-month
- Average Staff
  - S = √8.5 = 3

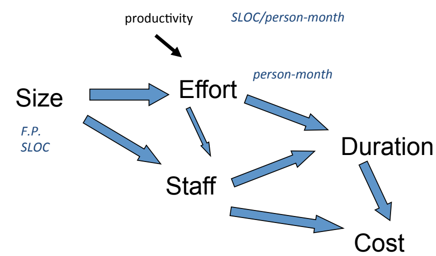
Schedule & Cost

- 3 + months…
- as 7 iterations of 2 weeks each:
  - [1, 2, 3, 1]
- Staff:
  - 2 for inception and elaboration,
  - 3 for construction and transition
- Cost:
  1 x 2 x 2 + 2 x 2 x 2 + 3 x 3 x 2 + 1 x 3 x 2
  = 4 + 8 + 18 + 6 = 36
  = 9+ staff month
  = 10 sm x $6,000 = $60,000

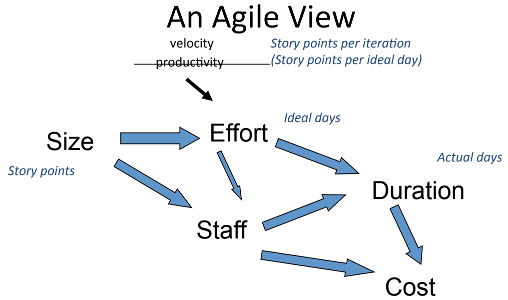
Ooops! Issue?

- Using McConnell's table
  - Business system
  - 9 staff-month goes with 6 months duration
  - and 10 KSLOC
- Risk?
  - Are we over-optimistic?
  - What make us think we can pull it in 3 months?
  - Is 10 KSLOC realistic, or totally out of this world?
  - Can we compare with another similar project?

The classical View



An Agile View



Summary

- Estimation of size is hard
  - SLOC, FP, FFP, UCP (etc.)
  - Wideband Delphi, and similar "wet finger" techniques
- From Size to Effort & Duration
  - Models: COCOMO 1, COCOMO 2, SLIM
  - Cost drivers: up and down, technical and non-technical
- Use multiple approaches when possible
- Estimate repetitively
  - as soon as more information is known
  - minimum, once per iteration
- Gather and keep data on past projects