Software Project Management

Daniel M. Berry
with material from James E. Tomayko

Nature of Software Production

SOFTWARE — program system product (PSP)

PROJECT — planned

MANAGEMENT — make sure that the PSP comes out as planned

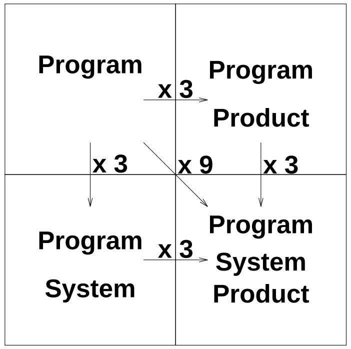
Software -1

We are not talking about *programs*, but about *program system products (PSP)*

We *all* know *all* about programs, but what about PSPs?

Fred Brooks explains the difference and shows the effort involved (multipliers may be bigger if the number of components is large):

Software -2



Program:

- program is complete by itself
- program is ready to run by author for planned inputs on system on which it was developed, and probably under *no* other circumstances

Program System:

- each program is a component in integrated collection (system)
- precisely defined interface to which all programs in system must comply
- each program must stick to reasonable resources
- each program is tested with other programs; number of combinations grows quadratically with each additional program

Program Product:

- product can be run, tested, repaired, extended by anyone, not just author
- product runs on multiple platforms
- product accommodates many sets of data
- range and form of input to product must be generalized
- product must test for validity of input and provide response to invalid inputs
- must be product documentation for maintainers and users

Program System Product:

- all attributes of program system *and*
- all attributes of program product

Programs vs. PSPs -1

Perhaps the key problem in SPM is that when we should be thinking about PSPs, we continue to think about programs, and *all* expectations,

- **ease vs. difficulties,**
- **time,**
- **costs,**
- **you name it,**

are off by an order of magnitude.

Programs vs. PSPs -2

We see only the program in the heart of the PSP and forget all the other *junk* that must be added to make it a PSP!

Project

The objective of the project to build a PSP is to make sure that all the necessary junk gets planned in.

Projects have plans:

- **Specific work to do**
- **Deliverables**
- **Resources**
  - **Multiple People**
  - **Schedule**
  - **Budget**

Management -1

Any project with more than one person must be managed by definition, just to keep the communication going between the folk in the project.

Note that the management does not need to be applied externally, the manager can be one of the managed folk!

Management -2

The job of management is to make sure that the planned junk does not get left behind in the zeal to release the PSP when only the program in its heart has been written!

Management -3

- **Control leads to quality.**
- **Deliver what you promise.**
- **Allocate resources properly.**
- **Communicate and facilitate communication.**

Truths about Management

Boehm says:

“Poor management can increase software costs more than any other factor”.

“Poor management can decrease software productivity more rapidly than any other factor”

“The single most important factor in the success of a (multi-person) software project is the talent of its project manager”

Production of Managers

Mark Kellner goes so far as to say:

“The software engineering profession has not produced a cadre of capable/competent managers.”

Promotion up the technical ladder requires skills different from those needed by a manager.

Basic Equation

Profit = Revenues – Expenses

Usually revenues are fixed, either by contract or by the market.

Therefore to maximize, or even to guarantee, profit, it is essential to reduce expenses or costs.

Cannot reduce anything unless you know what it is.

Required Knowledge

Therefore, we gotta know what the costs are.

More importantly, we gotta what they *will* be if we're using projected costs to determine what the price (i.e., revenues) is.

Multi-pronged attack

Actually, there is a mirror image to reducing costs that has the same net effect, when it is done right.

- **Reducing costs**
- **Increasing productivity**

Reducing Costs

To reduce cost, you have to know what you're spending and where you're spending it.

Increasing Productivity

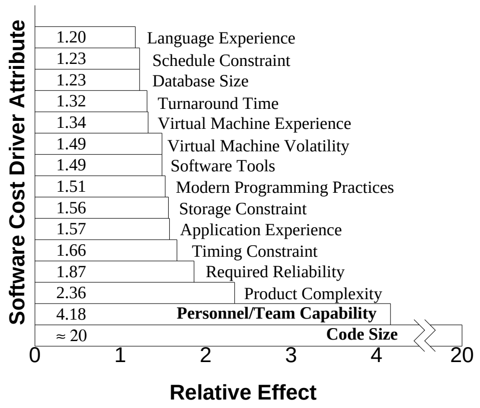
To increase productivity, you have to eliminate unnecessary work and make the work that's done more effective.

Recurring Theme -1

Recall the diagram presented earlier of Boehm's cost drivers, showing how much more important personnel and team capability are than any technical factor.

Well, there is another line showing an even more important cost driver for software, namely the size of the code itself, and that's about 5 times more important than personnel and team capability.

Recurring Theme -2



Recurring Theme -3

So a simple way to reduce costs is to write less code! Nu?!

- **Beg it.**
- **Borrow it.**
- **Buy it.**
- **(No "steal it"!)**

Outline

- **Lifecycles**
- **Cost Estimation**
- **Risk Management**
- **Planning General Issues**
- **Process Management**
- **Planning Details**
- **Notations**

Each major topic will be preceded with its own outline!

Lifecycles

Outline:

- **Build-and-fix**
- **Waterfall**
- **Spiral**
- **One Sweep of Spiral**
- **Requirements Engineering**

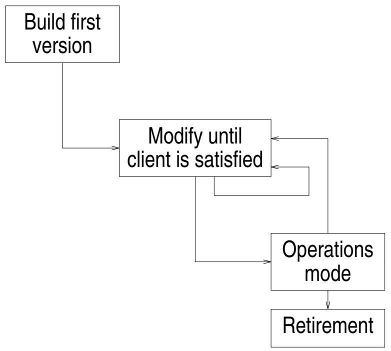
Reasons for Considering Topic

Lifecycle models are generally considered too constraining and too ideal.

However, it is useful to understand what lifecycles were envisioned when documentation standards were developed.

Later we will learn how and why to fake the lifecycles to make the documents.

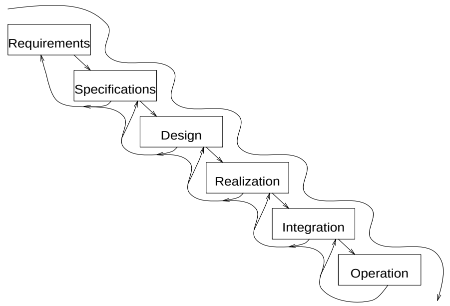
Build-and-Fix



All too common!

Waterfall -1

Win Royce described what is now called the traditional waterfall lifecycle.



Waterfall -2

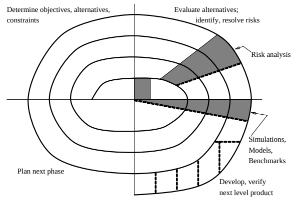
The waterfall model is descriptive, and not too good at that.

It is by *no* means proscriptive; it simply does not work!

Lowering fever is a good model of getting over an illness, but refrigerating a sick person will not make him or her better.

Spiral Model

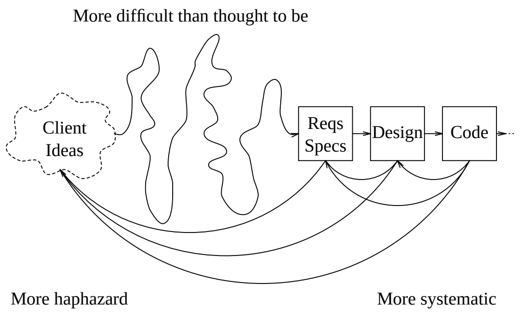
Barry Boehm introduced the more realistic spiral model.



The waterfall can be considered one 360° sweep of the spiral.

One Sweep of Spiral

But here is the *real* lifecycle for one sweep.

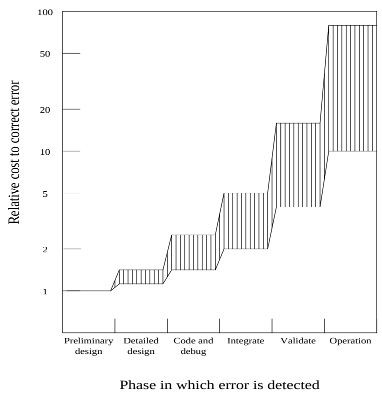


Requirements Engineering -1

It is important to try to straighten the tortuous line from conception to requirements specifications.

Recall that the cost to correct an error skyrockets as a function of lifecycle stage.

Requirements Engineering -2



Requirements Engineering -3

Meir Lehman identified E-type software.

- **A system that solves a problem or implements an application in some *real world* domain.**
- **Once installed, an E-type system becomes inextricably part of the application domain, so that it ends up altering its own requirements.**

Requirements Engineering -4

Martin & Tsai did experiment to identify lifecycle stages in which requirement errors are found

- **Used polished 10-page requirements for centralized railroad traffic controller.**
- **Ten 4-person teams of software engineers looked for errors.**
- **Requirements author believed that teams would find only 1 or 2 errors.**

Requirements Engineering -5

- **92 errors, some very serious, were found!**
- **Average team found only 35.5 errors, i.e., it missed 56.5 to be found downstream!**
- **Many errors found by *only* one team!**
- **Errors of greatest severity found by fewest teams!**

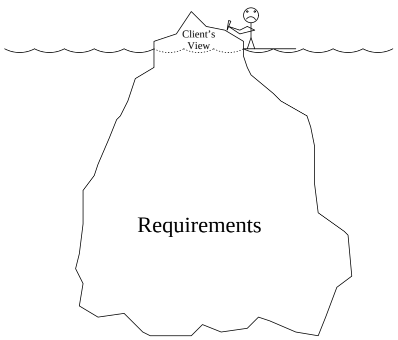
Requirements Engineering -6

Most errors are introduced during requirements specification.

Boehm: At TRW 54% of all errors were detected after coding and unit test; 85% of these errors were allocatable to the requirements and design stages rather than the coding stage, which accounted for only 17% of the errors.

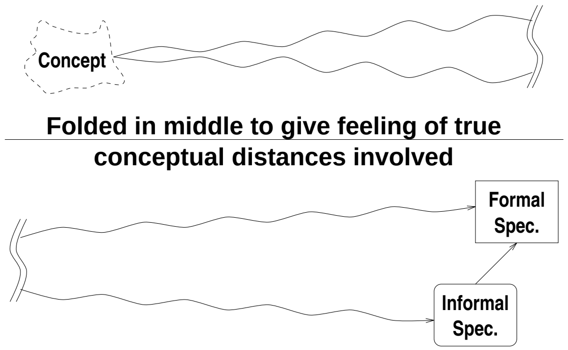
Requirements Engineering -7

The requirements iceberg and various icepicks chipping at it:



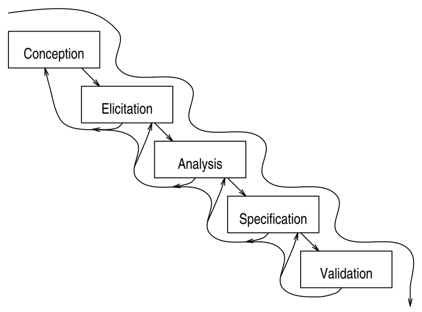
Requirements Engineering -8

The problem is the conceptual distance from the client's ideas to the specifications.



Requirements Engineering -9

Requirements engineering has its own lifecycle:



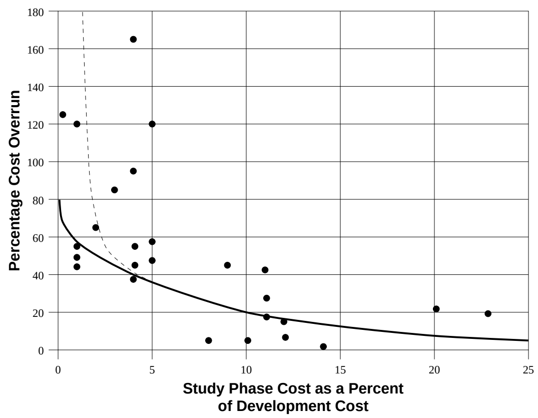
Requirements Engineering -10

The next slide shows the benefits of spending a significant percentage of development costs on studying the requirements.

It is a graph from “System Engineering Overview” by Kevin Forsberg and Harold Mooz, 1996.

It relates percentage cost overrun to study phase cost as a percentage of development cost in 25 NASA projects.

Requirements Engineering -11



Requirements Engineering -12

The study, performed by W. Gruhl at NASA HQ includes such projects as

- **Hubble Space Telescope**
- **TDRSS**
- **Gamma Ray Obs 1978**
- **Gamma Ray Obs 1982**
- **SeaSat**
- **Pioneer Venus**
- **Voyager**

Cost Estimation

Outline:

- **What needs to be estimated**
- **Understanding software costs**
- **Estimation models**

What Needs to be Estimated?

- **Size**
- **Time**
- **Complexity**
- **Effort**
- **Resources**

That is, all sorts of *costs*!

Understanding Software Costs

What affects software costs?

- **Why estimation so hard for SW?**
- **Personnel & team capabilities**
- **Individual differences**
- **Team size**
- **Product complexity**
- **Fun vs. duty**
- **Main reason for poor estimates**
- **Accuracy of cost estimation**
- **Two-step cost estimation**
- **Risk of software cost estimation**

Why Estimation SO Hard for SW?

- **Often entirely new**
- **Start with incomplete specifications**
- **Moving target**
- **People factors**
- **Difficult to relate size and complexity**

Personnel & Team Capabilities -1

Effects of personnel and team capability on productivity:

- **factors that managers cannot affect**
- **factors that managers can affect**

Personnel & Team Capabilities -2

There is nothing a manager can do to change the talent of the individuals.

However, there are things that can be done to bring the talent of the indviduals out.

Personnel & Team Capabilities -2

Some things that management can do to help improve personnel productivity:

- **make a better programming environment**
- **provide education and training**
- **improve the software process**

Individual Differences -1

- **Individuals can be virtuosos, but there is a limit to what a virtuoso can do.**

Individual Differences -2

- **An experiment to show that programmers using interactive system are more productive than those using batch system failed because the so-called independent variable of programmer quality dominated; they found a 26 to 1 productivity ratio among *experienced* programmers.**
- **Other studies have shown 17 to 1 productivity ratio among individual programmers.**

Individual Differences -3

Not many human endeavors have this wide of a gap.

For example, in the best sprinters are not 17 times faster than the worst.

On the other hand, Michael Jordan is around 17 times better than the worst basket shooter and Pat Riley is around 17 times better than the worst coach.

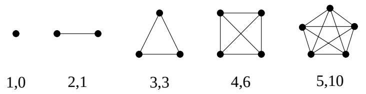
Individual Differences -4

I guess that the key here is that when *skill* rather than *capacity* is involved, individual differences are large.

Team Size

Smaller teams are generally more effective than larger teams.

- **There is less communication between members.**



number of persons, lines of communication

- **Stars of group can shine through easier.**

Product Complexity

The more complex the project, the lower the productivity.

There are definite harbingers of complex products:
- **newness for the sake of newness**
- **embedded in a larger system**
- **filled with features**

Fun vs. Duty

The fun of software development is outweighed by the drudgery.

- **need for perfection**
- **taking responsibility for quality of work products**
- **debugging**
- **documentation**

That everyone would prefer to forget the drudgery leads to...

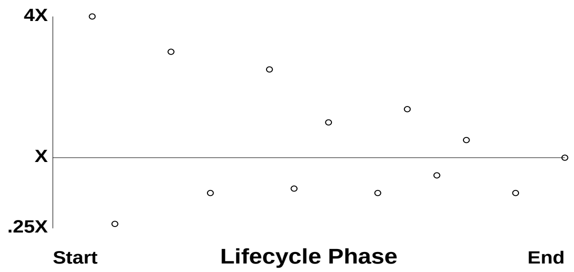
Main Reason for Poor Estimates

Estimating fails due to lack of attention to distracting factors.

That is, everyone forgets the junk that makes a program system product out of a program!

Accuracy of Cost Estimation

Basic problem with *all* estimates!



Lousy foresight; perfect hindsight!

Two-Step Cost Estimation

Must map from requirements to personnel.

1. **Internal influences step, e.g., function-point analysis that estimates program size from details of required functions**
2. **External influences step, e.g., COCOMO that maps from estimated size to people needed**

Code Size

Code size can

- **be affected by the desired functionality.**
- **affect the desired functionality.**
- **be affected by the implementation.**
- **be affected by external factors, e.g., memory size limitations or coding standards.**

Code Size Estimation Models

- **Sizing by analogy**
- **Wideband Delphi technique**
- **Clark's model**
- **Function points**

Note how all of these depend on detailed knowledge of expected internals of the program or of its requirements.

Sizing by Analogy

Find another program like the one you're going to build and base your estimate on its size.

Wideband Delphi Technique

1. Individuals examine the requirements.
2. Discuss the requirements in a group.
3. Individuals estimate anonymously.
4. Compare individual estimates to mean.
5. Discuss results in group.
6. Loop back to 3.

Clark's Model

$E = \frac{(L + 4M + H)}{6}$ for each module

M = what you think is about right size for module

H = the biggest you think it can ever be

L = the smallest you think it can ever be

E = final estimate for the module

Captures standard bell-shaped distribution.

Function Points -1

Function points were invented by Allan Albrecht at IBM in 1979.

Function points were originally intended for commercial applications, and as we look at the factors involved, this will be clear.

However, the idea of finding key factors can be applied to any application domain; you just have to find a different set of functional factors.

Function Points -2

Objective: to be able to estimate code size relatively easy *early* in the lifecycle from knowledge of only the requirements.

This is all you know when the customer says, "Here's what I want. How much will it cost?"

If you over estimate, you lose the job. If you underestimate, you lose the profit!

Function Points -3

Function Points (FP) =

Sum of Functional Factors $\times$

[0.65 + 0.01 $\times$ Sum of Weighting Factors]

Functional Factors

Five Functional Factors (Simple, Average, Complex)

Functional Factor	Weights for S	A	C
Number of user inputs	$\times$ 3	4	6
Number of user outputs	$\times$ 4	5	7
Number of user inquiries	$\times$ 3	4	6
Number of files	$\times$ 7	10	15
Number of external interfaces	$\times$ 5	7	10

Weights of Functional Factors

Weights can be adjusted, according to experience.

Nowadays, because of libraries, the weights tend to be the same.

Weighting Factors -1

1. **Does system require reliable backup and recovery?**
2. **Are data communications required?**
3. **Are there distributed processing functions?**
4. **Is performance critical?**

Weighting Factors -2

5 Will system run in an existing, heavily used operational environment?
6 Does system require on-line data entry?
7 Does the on-line data entry require the input transaction to be built over multiple screens or operations?

Weighting Factors -3

8 Are master files updated on-line?
9 Are input, output, files, or inquiries complex?
10 Is the internal processing complex?
11 Is the code designed to be reusable?

Weighting Factors -4

12 Are conversion and installation included in the design?
13 Is system designed for multiple installations in different organizations?
14 Is the application designed to facilitate change and ease of use by the users?

Alternate Weighting Factors -1

1 Data communications
2 Distributed data processing
3 Performance criteria
4 Heavily utilized hardware
5 High transaction rates

Alternate Weighting Factors -2

6 On-line data entry
7 End-user efficiency
8 On-line updating
9 Complex computations
10 Reusability

Alternate Weighting Factors -3

11 Ease of installation
12 Ease of operation
13 Portability
14 Maintainability

Modern Weighting Factors

Nowadays, the following are considered:

- interfaces with other applications
- special security features
- direct access by third party software
- documentation requirements
- training and help subsystems

Modern Formula -1

$$\begin{aligned}\text{FP} = \; & 0.44 \times \text{No. of Input Element Types} + \\ & 1.67 \times \text{No. of Entity Type References} + \\ & 0.38 \times \text{No. of Output Element Types}\end{aligned}$$

That is count number of types used rather than objects.

Presumably, all accesses to objects of the same types will reuse the same operators.

Modern Formula -2

The new formula reflects current modular style of programming.

In this style, the intellectual effort is in designing classes not variables.

Key Observations about FPs

These measures are based on user's external view and are technology independent.

They can be developed early in the lifecycle, enabling their use for early cost estimation in planning stages.

They can be understood and evaluated by nontechnical users!

Additional FP Metrics

- **Productivity = FP/PM**
- **Quality = Errors/FP**
- **Cost = Dollars/FP**
- **Documentation = Pages/FP**

FP vs. DSI

Basic Assembler	**360**
Macro Assembler	**213**
C	**128**
COBOL	**105**
FORTRAN	**105**
Pascal	**80**
Ada	**72**
Basic	**64**
4GL	**25**

So now we have the data to plug into, e.g., COCOMO to estimate personnel!

Advantages of Function Points

- Deeper understanding of complexity than other methods
- More than one variable
- Can and must look at requirements
- Can and must involve user

Disadvantages of Function Points

- More complex
- Subjective
- DP-biased

COCOMO

- History
- Empiricism
- Three levels of detail
- Three development environments
- Assumptions of COCOMO model
- DSI
- Person-month formulae
- Time of development formulae
- Full-time software personnel
- COCOMO caveat

History

Developed by Barry Boehm in 1970s base on his experience as head of software development at TRW.

He was in position of having to make estimates, and he got to be good at it.

COCOMO is a formalization of his experience in 62+ projects at TRW.

Described in Boehm’s 1981 book *Software Engineering Economics*

Empiricism

It is entirely empirical, based on 62+ projects at TRW.

Although, the shape of the formulae makes sense, that is, factors that are adversely affected by growth in inter-module communication contribute to quadratic growth in needed personnel.

Each place will have to adjust the constants to reflect what is true there.

Three Levels of Detail -1

- **Basic**
- **Intermediate**
- **Detailed**

Three Levels of Detail -2

As the detail increases, the number of factors considered increases.

“Detailed” considered most accurate, but may not be.

In other words, by calibrating constants in basic model, might get better results faster!

Three Levels of Detail -3

We cover only basic here.

There are cost-estimation tools that implement the model.

Use of such tools can help insure company-wide input in calibration and company-wide consistency in resulting estimates.

Three Development Environments

1. **Organic (Informal group structure)**

2. **Semi-detached**

3. **Embedded (Very formal group hierarchy and process)**

Organic Development Mode

- **Thorough understanding of project objectives**
- **Extensive experience with related systems**
- **Minimal need to meet predefined requirements**
- **Minimal need to conform to external interfaces**
- **Hardware already exists**
- **Minimal need for innovation**
- **Loose deadline**

Semi-detached Development Mode

- **Much understanding of project objectives**
- **Much experience with related systems**
- **Much need to meet predefined requirements**
- **Much need to conform to external interfaces**
- **Some concurrently developed new hardware**
- **Some need for innovation**
- **Moderately tight deadline**

Embedded Development Mode

- **Only general understanding of project objectives**
- **Moderate experience with related systems**
- **Must meet predefined requirements**
- **Must conform to external interfaces**
- **Much concurrently developed new hardware**
- **Much need for innovation**
- **Very tight deadline**

Assumptions of COCOMO Model

- **Low volatility of requirements**
- **Good SE practice**
- **Good management**

DSI -1

The Concept of DSI is an attempt to capture intellectual effort.

Delivered (not thrown out code)
Source (that which humans operate on)
Instructions (and not comments)

However, if test suites are to be delivered, they must be counted too

Also use KDSI, thousand DSI

DSI -2

Counting source instructions is hard

What do we count?
- **line-feeds**
- **semicolons**

What do we do about control constructs?
- **if (...) - - - ;**
- **if (...) {**

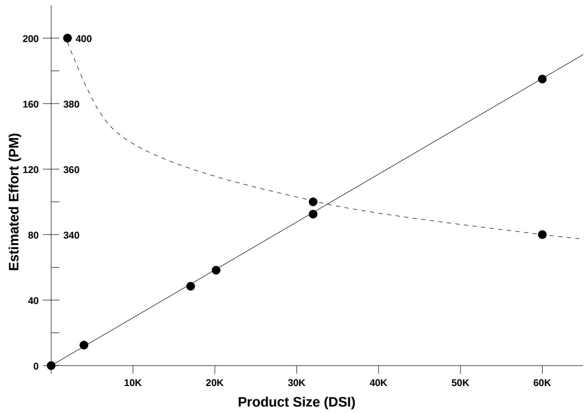
Person-Month Formulae -1

Organic: $PM = 2.4 \times KDSI^{1.05}$

Semi-detached: $PM = 3.0 \times KDSI^{1.12}$

Embedded: $PM = 3.6 \times KDSI^{1.20}$

Person-Month Formulae -2



Time of Development Formulae -1

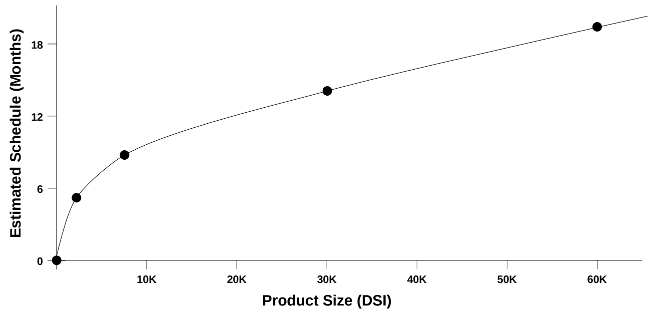
Organic: $TDEV = 2.5 \times PM^{0.38}$

Semi-detached: $TDEV = 2.5 \times PM^{0.35}$

Embedded: $TDEV = 2.5 \times PM^{0.32}$

This is consistent with people and time not being exchangeable!

Time of Development Formulae -2



Full-Time Software Personnel -1

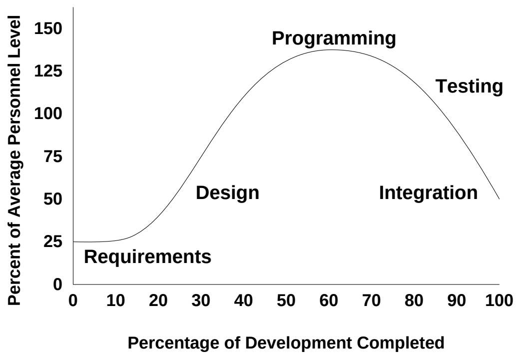
$$FSP = \frac{PM}{TDEV}$$

This assumes uniform personnel level over whole project.

Usually want to start off light and hire more as project evolves.

Use Rayleigh curve.

Full-Time Software Personnel -2



Full-Time Software Personnel -3

$$FSP = PM \times \left(\frac{t}{p^2}\right) \times e^{-\left(\frac{t^2}{2p^2}\right)}$$

Here p is percentage of development schedule completed at peak.

Ada Modifications to Basic Model -

- Assume smaller teams initially, but allow longer design period.
- DSI = carriage returns in specification parts and bodies.

Ada Modifications to Basic Model -2

$PM = 2.4 \times KDSI^{1.05}$

$TDEV = 3 \times PM^{0.32}$

PM for PD, DD, CUT, IT $=$
PM $\times$ (.23, .29, .22, .26)

TDEV for PD, DD, CUT, IT $=$
TDEV $\times$ (.39, .25, .15, .21)

PD = preliminary design, DD = detailed design, CUT = code unit testing, IT = integration testing

Intermediate Model -1

Sample multipliers for cost drivers:

Cost Drivers	Very Low	Low	Nom-inal	High	Very High	Extra High
Product Attributes						
Required Software Reliability	0.75	0.88	1.00	1.15	1.40	
Database Size		0.94	1.00	1.08	1.16	
Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Personnel Attributes						
Analyst Capability	1.46	1.19	1.00	0.86	0.71	
Applications Experience	1.29	1.13	1.00	0.91	0.82	
Programmer Capability	1.42	1.17	1.00	0.86	0.70	
Virtual Machine Experience	1.21	1.10	1.00	0.90		
Programming Language Experience	1.14	1.07	1.00	0.95		

Intermediate Model -2

Sample rating determination:

Cost Drivers	Situation	Rating
Required Software Reliability	Serious financial consequences of software fault	High
Database Size	20,000 bytes	Low
Product Complexity	Communications processing	Very High
Analyst Capability	Good senior analyst	High
Applications Experience	Three years	Nominal
Programmer Capability	Good senior programmer	High
Virtual Machine Experience	Six months	Low
Programming Language Experience	Twelve months	Nominal

COCOMO Caveat

All depends on estimate of code size (DSI), which can be

- **a result of solving a problem**
- **driven by external factors, e.g., memory bound by system design or standards**
- **directly affected by desired functionality**

and it

- **directly affects desired functionality**

Advantages of COCOMO

- **Reasonable**
- **Simple**
- **Clear**

Disadvantages of COCOMO

- **Too believable**
- **Too dependent on one variable**
- **Subjective**

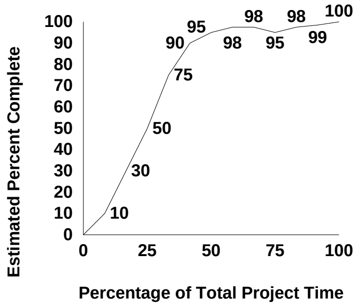
Implications of Cost Drivers

The biggest cost driver is the number of instructions to develop.

Therefore, it pays to try to eliminate the necessity of developing new instructions.

- **Off-The-Shelf Software (OTS)**
- **Reuse or adapt existing software**
- **Application generators**

Risks of Software Cost Estimation



Why don't I believe "It's 90% done!"

Typical Estimation Results

Contractor	***LOC***	***Dollars***
A	**153K**	**2.8M**
B	**282K**	**2.5M**
C	**400K**	**4.6M**
D	**735K**	**4.5M**
E	**766K**	**2.1M**

And the Winna is

E!

Actual code size 900K Actual cost 5.8M

Nu?

Why Estimates Fail?

- **Optimism**
- **Confusion between effort and progress**
- **Gutless estimating**
- **Poor progress monitoring**
- **Requirements Creep**
- **Adding peoplepower**

Optimism -1

It comes primarily from forgetting that we are dealing with a program system product rather than a program.

It happens to the best of us!

Optimism -2

Meir Burstin was a very successful software manager.

He could estimate project durations just like that, right off the seat of his pants.

His company's success was testimony to his estimation ability.

Optimism -3

Soon after he sold the company, he went to get a Ph.D. and had to develop a requirements management system for his Ph.D. research.

His estimate was suddenly low, by an order of magnitude.

Optimism -4

When asked about this, he said that since he was doing by himself, and he *is* a good programmer, he forgot to apply all his normal fudge factors.

That is, he thought he could do it as a program and forgot that it was a PSP!

Optimism -5

Donald Knuth is a good programmer, a programmer's programmer.

I remember hearing a talk he gave in 1980 when he was about .75 year into the TEX-and-METAFONT project.

He expected to be finished soon, that it would be a 1-year project.

He wanted to get back to writing his 7-volume encyclopedia of programming!

Optimism -6

It took 10 years to get to the stage that it was a satisfactory PSP.

I'll bet that he was thinking program, not PSP, when he gave his 1-year estimate!

Rules of Thumb -1

Here are some rules of thumb that you can use.

If you think program when you're supposed to be thinking PSP.

Identify which rule of thumb below applies, and take your "program" estimate and let it be just the coding stage.

Rules of Thumb -2

From that, you can get a rough estimate of the full time needed.

Also when you do get a detailed estimate, if it doesn't jibe with one of these rules of thumb, something is wrong.

Brooks's Rule of Thumb

For distribution of effort in functionally decomposed programs:

1⁄3	**planning**
1⁄6	**coding**
1⁄4	**unit test and early integration test**
1⁄4	**integration test**

Tomayko's Rule of Thumb:

For distribution of effort for developing modular, data hiding software:

1⁄2	**planning**
1⁄12	**coding**
1⁄4	**unit test and early integration test**
1⁄6	**integration test**

Adding Peoplepower -1

Sometimes it seems unavoidable.

Sometimes it is done to protect against future problems.

In either case it's a disaster!

"Adding manpower to a late software project makes it later" — F.P. Brooks, Jr.

Adding Peoplepower -2

The problems are bring the new people up to date and adding them to the communication loop.

Both catch-up and additional necessary communication cost more time than a new person adds.

You can add more bricklayers to a wall building project; no catch up and no additional necessary communication!

Adding Peoplepower -3

If it *is* necessary, i.e., it simply requires too much work for the available people to deliver.

Then add the people!

But, then the whole project must be re-planned with a *later* deadline!

No two ways about it!

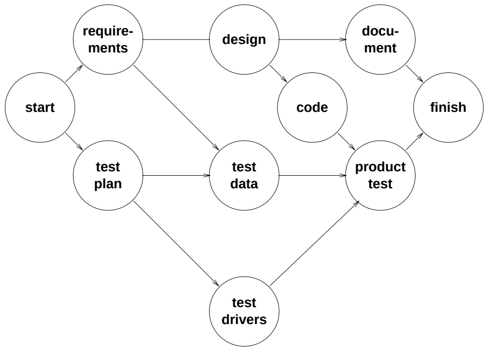
Cost Increases

Causes:

- **Requirements changes: 85%**
- **Poor estimation: 15%**

Danger Lurks -1

Quick, look at a typical project's Pert Chart:



Danger Lurks -2

The duration of which development phase is the most dangerous to underestimate?

Danger Lurks -3

The answer:

Testing!

You are too close to delivery for recovery!

Self-Fulfilling Prophecies

Different estimates make different projects!

Tarek Abdel-Hamid found that estimates tend to influence people's work habits.

Parkinson's law: Work tends to expand to fill the time available!

Validity of Estimation Models

Chris Kemerer did an Empirical Evaluation of Software Cost Estimation Models.

Used COCOMO, FPs, and other methods on data of *finished* projects.

Data on 15 large COBOL projects were collected to test accuracy of the models *ex post facto* using actual DSI data and not estimates.

Kemerer's COCOMO Data

A mean over 15 projects:

Actual/Estimate	***PM***	***Error***
Actual	**219**	
Basic Estimate	**1226**	**610%**
Intermediate Estimate	**1280**	**583%**
Detailed	**1291**	**607%**

Kemerer's FP Data

A mean over 14 projects:

Actual KLOC:	**221**
Estimate:	**128**
Error:	**38% low**

What can be inferred?

Validity of Models

Kemerer concludes that the models are generally valid, that the shape of the formulae are OK.

But all the models *need* calibration of their multipliers, powers, and constants.

So this means that before you can start believing your estimates you have to have been applying them over a number of projects.

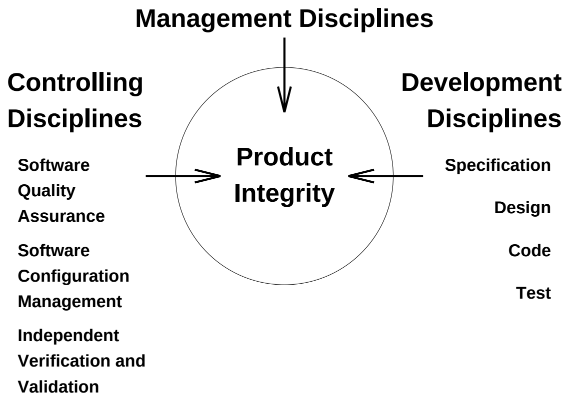
What’s a Project Manager to Do?

- **Reject the models.**
- **Use fudge factors in the models.**
- **Adapt and calibrate the models.**
- **Make new models.**

Configuration Management (CM)

- **Role of CM**
- **Functions of CM**
- **Commitment is necessary**
- **Typical configured items**
- **CM library functions**
- **Variations vs. revision**
- **Types of changes**
- **Implementing CM**

Role of CM



Functions of CM

- **Maintain integrity of configured items**
- **Evaluate and control changes**
- **Make the product visible**

Commitment is Necessary

No matter how good are the CM tools, it is always possible to by-pass them, rendering them useless.

Therefore:

Commitment to configuration management by the entire organization is the key to success of configuration management.

Typical Configured Items -1

- **Requirements**
- **Specifications**
- **Design Documents**
- **Source Code**
- **Object Code**
- **Load Modules**
- **Tools**

Typical Configured Items -2

- **System Description**
- **Test Plans**
- **Test Suites**
- **User Manuals**
- **Maintenance Manuals**
- **Interface Control Documents**
- **Memory Maps**

In other words, all deliverables!

CM Library Functions

- **Software Part Naming**
- **Configured Item Maintenance/Archiving**
- **Version Control**
- **Revision Control**
- **Preparation for Product Release**

Variations vs. Revision

Variations can co-exist, e.g., different versions of same program for two different CPUs or OSs

A new revision replaces an older one.

Types of Changes -1

- **Disrepancies**
  - **Requirements Errors**
  - **Development Errors**
  - **Violations of Standards**

- **Requested Changes**
  - **Unimplementable Requirements**
  - **Enhancements**

Types of Changes -2

- **Disrepancies — absolutely critical, cannot go on without fixing**
- **Requested Changes — more voluntary, can go on without fixing**

  Requested changes tend to be more difficult than fixing disrepancies and tend to have major impacts on other features *if* carried out.

  Disrepancies have major impacts if they are *not* fixed.

Implementing CM

- **Fundamental principles**
- **CCB characteristics**
- **Hierarchies of CCBs**
- **Evaluating changes**
- **Disrepancy report (DR) evaluation**
- **Change request (CR) evaluation**
- **Simultaneous update problem**
- **Variations & revisions tree**
- **Tools for version control**
- **Tools for system building**
- **Trends of reporting**
- **Standards for CM plans**

Fundamental Principles

Fundamental principles to guide configuration control boards (CCBs):

- **Principle of authority**
- **Principle of solitary responsibility**
- **Principle of specificity (assigning the DR or CR to the right CCB)**

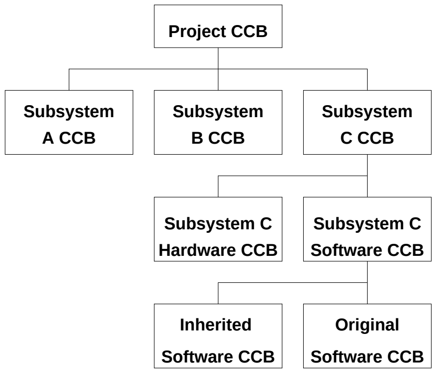
As Jim Tomayko says, the CCB has to have teeth (grrr!) or it will not work!

CCB Characteristics

Factors determining CCB characteristics:

- **Hierarchies**
- **Scope**
- **Composition**

Hierarchies of CCBs



Evaluating Changes -1

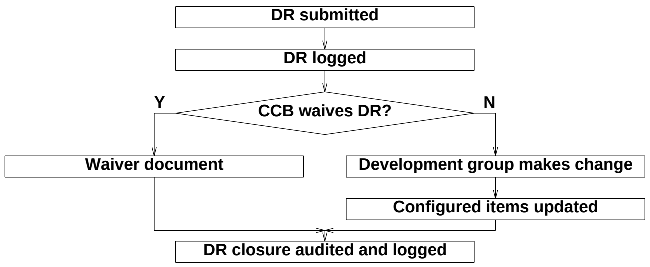
Key factors in evaluating proposed changes:

- **Size**
- **Complexity**
- **Date needed**
- **CPU and memory impact**
- **Cost**
- **Test requirements**

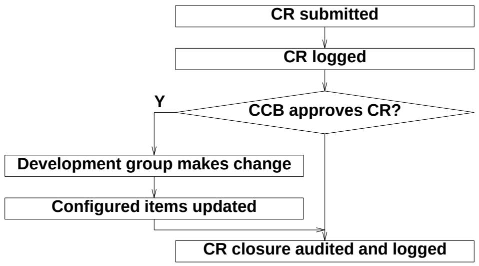
Evaluating Changes -2

- **Criticality of area involved**
- **Politics (customer/marketing desires)**
- **Approved changes already in progress**
- **Resources (skills/hardware/system)**
- **Impact and current and future work**
- **Is there an alternative?**

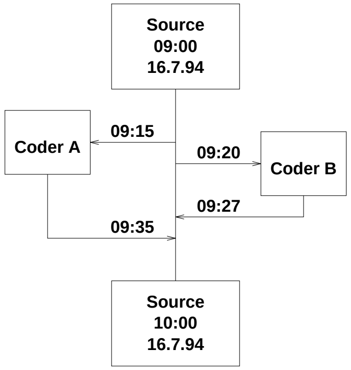
Disrepancy Report (DR) Evaluation



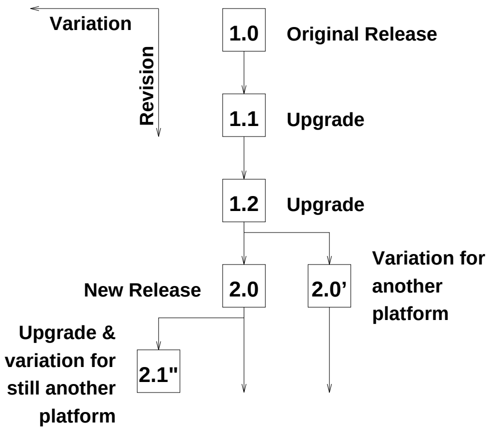
Change Request (CR) Evaluation



Simultaneous Update Problem



Variations & Revisions Tree



Tools for Version Control

Keeping track of versions and revisions tree

- **SCCS**
- **RCS**
- **Domain**

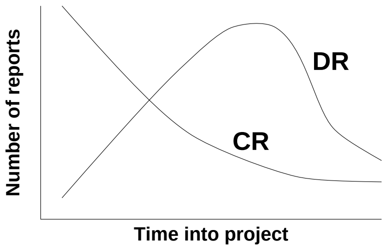
Tools for System Building

Describe the system and let the tool build it according to description.

Description shows module dependencies: if A includes B then recompilation of B forces recompilation of A

- **Shell scripts/batch files**
- **Make**
- **Domain**

Trends of Reporting



Standards for CM Plans

- IEEE (ANSI/IEEE 828-1983)
- MIL-STD-483A (Air Force)
- DoD-STD-2167

Legal Issues

- Intellectual Property Protection
- Liability and Warranty

Note that in all the following, laws exist, agencies exist to administer various protections, and courts have final say in disputes over whether protections are legitimate in any case.

Intellectual Property

What is intellectual property?

Why is it important?

Pirating costs money and future advancement!

Three main ways to protect intellectual property:
- Patent
- Copyright
- Trade Secret

Patent -1

Exclusive right to manufacture and sell product meeting description of patent for fixed number of years (about 50).

Get it by applying to national patent office and showing that your invention meets requirements and you were first to think of it.

Must disclose the full details of the invention, but this is what you get the exclusive right to produce and sell.

Patent -2

Requirements for patentability:

- **utility**
- **novelty**
- **non-obviousness**

Patent -3

Excluded from patents:

- **business systems**
- **printed matter**
- **mental steps**
- **algorithms**

Patent -4

But Gottschalk vs. Benson 1972 granted patent to an industrial process that included a computer program to do process control that is beyond human and mechanical capabilities.

So now there is a big flurry to patent programs.

Patent -5

Definition of algorithm:

- **finite**
- **definite**
- **input**
- **output**
- **effective**

Patent -6

Routines that are part of programmer’s normal bag of tricks are being patented.

Patents can be challenged in courts and that is the only defense against an improperly granted patent.

Copyright -1

Copyright is the exclusive right to publish something written or performable for a fixed number of years.

Protection is on expression and not idea, but you cannot copyright something for which there is only one expression, e.g., a formula.

Protection extends to derived works, e.g., obtained by translation.

Copyright -3

Recently copyright has been extended to software in any form, source or object.

The copy needed to run and a back up copy are permitted when you license software; copyright cannot prevent someone from using object for its intended purpose or for personal use.

Copyright -3

Licensing vs. buying software.

Shrinkwrap licensing.

License usually prohibits reverse engineering which is allowed by copyright law.

Big issue in courts now: is output generated by programs copyrightable, e.g., look and feel?

Trade Secret -1

Problem with both patent and copyright: you have to disclose secrets.

Can go trade secret route.

Laws exist to protect secrets obtained unfairly or fraudulently *if* you take active steps to protect them.

Trade Secret -2

Active steps:

- **Security at plant**
- **Non-disclosure agreements for employees**
- **Non-disclosure agreements with potential customers**
- **Licensing agreements that specify no reverse engineering and non-disclosure of secrets discovered while using**

Trade Secret -3

Caveat: You lose the protection of the law if the usurper of the secret can show that you have not been diligent in protecting the secret or have given it out without limits.

Liability and Warranty -1

Liability: You are responsible for the effects of your actions and products and can be made to pay for damages that result, and sometimes punitive damages if it can be shown that you were willfully negligent

Warranty: Claim, backed by right to return a product at no cost to consumer, of capability, suitability, or fitness of the product for its stated or intended purpose.

Liability and Warranty -2

Protection from liability:

- **Limited warranty**
- **Disclaimers**
- **Limit of remedy**

Liability and Warranty -3

Definition of fraud:

- **Misrepresentation of a capability**
- **Using the user as a beta site**
- **Misrepresentation of suitability or fitness**
- **Misrepresentation of time or management or money savings**

Liability and Warranty -4

- **Express warranty**
- **Implied warranty**

Liability and Warranty -5

Concept of unconscionability

Liability and Warranty -6

Differentiate between

- **goods**
- **services**

Liability and Warranty -7

- **Professional liability**
- **Product liability**

Liability and Warranty -8

Warranty protection for computer systems:

- **hardware**
- **off-the-shelf software**
- **custom software**