

MSWE 607: Software Life Cycle methods and Techniques

- Instructor: Professor Marvin V. Zelkowitz
- Office: 4121 AV Williams
- Phone: 405-2690 or 403-8935 (Fraunhofer Center)
- Email (Best way to contact) mvz@cs.umd.edu
- Hours available: Wed and Thurs 9-10am or by Appointment
- Web page:
<http://www.cs.umd.edu/~mvz/mswe607-f00/>
- MSWE 607 is essentially a continuation of MSWE 601. It is part 2 of technology for software engineering. Some material may seem duplicated, but emphasis guaranteed to be different.
- MSWE 609 not being taught again. Some material to be included in MSWE 607

Goals of course

- **1. Many techniques are not that effective in improving software development practices**
- **2. But some techniques are effective**
- **3. Be skeptical when told that a new technique works**

- **Themes:**
What can you do to evaluate a new technique?
How do you measure what you are doing?

What is software engineering?

- **Software engineering: the application of a systematic disciplined quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.**
- **That is:**
 - **INPUT** is people, equipment, tools, methods, schedule, requirements.
 - **OUTPUT:** product and documentation
 - **PROCESS:** Manage risk to be successful.
- **But we don't really understand risk: "possibility of loss or injury." So there has to be a possibility of an event occurring and there has to be a loss if the event occurs.**

Risk

- **Most managers don't understand risk or are not willing to defend their decision if a risk analysis determines a project will fail. E.g., in an engineering discipline, an engineer may not undertake a project if professional judgment says it will fail. Software managers rarely make this decision.**
- **Some risk analysis is subjective, so there may not be a "right" answer. But there should at least be an indication of what the risk is.**

One measure of risk – Utility functions

Which of the following would you choose:

- **1. You can win between 0 and \$1000**
 - Spin a dial containing the numbers 1 through 10. If 1—9 show up, you get nothing; if 10 shows up you get \$1000.
 - Win \$100 for doing nothing
- **2. You can lose between 0 and \$1000**
 - Spin a dial containing the numbers 1 through 10. If 1—9 show up, you pay nothing; if 10 shows up you pay \$1000.
 - Pay \$100 for doing nothing

Utility function experiment

- For 1, most people choose the \$100 as “found money.” (E.g., in one class 31 out of 46 chose this option.) It didn’t matter that they had the potential for winning \$1000.
- But for 2, most people chose spinning the dial, even though they risked losing \$1000 (e.g., 16 out of 46 chose paying \$100).

But both experiments were the same (minimal gain of +/- \$100 and average gain or loss of \$100), yet on the chances of losing money, people were more willing to gamble than if they were guaranteed a win.

Understanding risk tolerance important for any project manager.

10 most important risk factors:

From capers Jones:	From Barry Boehm:
1. Inaccurate metrics	Personnel shortfalls
2. Inadequate measurement	Unrealistic schedules and budgets
3. Excessive schedule pressure	Wrong functionality
4. Inaccurate cost estimating	Gold plating
5. Management malpractice (e.g., knowledge)	Wrong user interface
6. Silver bullet syndrome (will save the project)	Continuous requirements changes
7. Creeping user requirements	Shortfalls in externally supplied components
8. Low quality	Shortfalls in externally provided tasks
9. Low productivity	Real time performance constraints
10. Cancelled projects	Straining computer science technology

Note: Boehm's experiences working for a DoD contractor (TRW) show up in his list – depending upon subcontracts (#7 #8) – and 1970s computer technology – performance (#9)

Risk Factors (Reifer)

Risk item	Risk reduction	Cost impact
Late delivery of hardware	Acquire time on another system	Computer time costs
New operating system	Benchmark early and perform acceptance test before use	Staffing to prepare benchmark
Feasibility of requirements	Feasibility analysis and simulation	Staffing to conduct analysis
Staffing up	Start recruiting and training early	Recruiting and training costs
Feasibility of design	Peer reviews	Added effort for review preparation
Lack of management visibility	Use detailed work packaging and weekly reporting	Added effort to prepare inputs for reports
Configuration integrity of software products	Use formal change control system	Purchase price of library plus administrative costs
Lack of a test discipline	Use independent test group and get them involved early	Cost of using test group

MANAGING RISK- Risk Assessment

Risk identification – What can go wrong? Which of 10 Jones or Boehm risk factors are possible to occur?

Risk analysis – For each risk, what is the cost if it occurs?

Risk prioritization – What is the probability that the risk may occur? What is the expected value of the loss (cost of risk times its probability of occurrence)?

MANAGING RISK- Risk Control

- Risk management planning – Identify high cost items and develop alternative strategies to handle occurrence of risk
- Risk resolution – Prototype, simulation, benchmarks, ... Evaluate alternative strategies before they are needed
- Risk monitoring – Continually evaluate risks and determine new risks during project execution.

SOFTWARE COST ESTIMATION

Basic method:

1. break system down into recognizable pieces
2. estimate each piece
3. sum pieces together

This works when we understand each decomposition and can estimate the pieces.

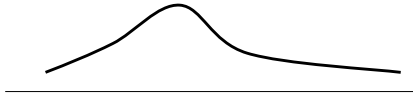
For other engineering fields (e.g., bridge and tunnel building), thousands are built each year so the basic understanding is there. Not so with software. We don't fully understand the decomposition principle to make well understood pieces.

SOFTWARE COST ESTIMATION - 2

- However, even in other fields, estimation is not good if the product is new (e.g., although the McHenry tunnel in Baltimore was built well under budget, there were billions in overruns on the English-French “Chunnel”; also NASA and space station).
- There is still some belief in the “exchanging” of staff and time (e.g., Brook’s “Mythical Man Month”). Research in 1970s, based upon hardware reliability theory, indicates that the “natural” growth of staff follows the Rayleigh curve.

Rayleigh estimator

$$\text{Effort} = 2 k a t e^{-at^2}$$



However, there is some empirical evidence that this schedule cannot be compressed more than about 25% without dire consequences.

MEASUREMENT MODELS

Major data needed to monitor a project:

- Effort (or its related cost)
- Defects
- Schedule
- Functionality

We don't really know what "functionality" means so we use alternatives:

- Size (e.g., Lines of code)
- Data complexity (e.g., function points)
- Modules
- Other measures of effort-to-build

1965 SDC model

- Effort (months) = -33.63
 - + 9.15 (lack of requirements)(0-2)
 - + 10.73 (Stability of design (0-3)
 - + 0.51 (Percent math)
 - + 0.46 (Percent storage/retrieval instructions)
 - + 0.40 (Number of subprograms)
 - + 7.28 (Programming language (0-1)
 - 21.45 (Business applications) (0-1)
 - + 13.53 (Stand alone program (0-1)
 - + 12.35 (First program on computer (0-1)
 - + 58.82 (Concurrent hardware development (0-1)
 - + 30.61 (Random access device used (0-1)
 - + 29.55 (Different host target (0-1)
 - + 0.54 (Number of personnel trips)
 - 25.20 (Developed by military organization (0-1)

COCOMO –COst Containment MOdel

Boehm – 1981. The assumption is that effort is related to size as follows: $\text{Effort} = k(\text{LOC})^p$ where k and p depend upon application domain and environment and $p > 1$.

Much of COCOMO is based upon 1970s technology which underlies many of the assumptions in the model (e.g., batch environments, DoD contracting, FORTRAN and similar languages).

One of the weaknesses of COCOMO is that other environments may not have the same underlying assumptions used in setting the parameters.

Nominal effort

- Basic effort equations:
- Organic development (well understood)
MM = 3.2 KLOC^{1.05}
- Semidetached (More complex)
MM = 3.0 KLOC^{1.12}
- Embedded (highly complex)
MM = 2.8 KLOC^{1.20}
- Then multiply MM by 15 COCOMO factors

COCOMO factors

Cost Driver	Low value	High value
Product attributes		
Required reliability	.75	1.40
Data base size	.94	1.16
Product complexity	.70	1.65
Computer attributes		
Execution constraints	1.00	1.66
Main storage constraints	1.00	1.58
Virtual machine volatility	.87	1.30
Computer turnaround time	.87	1.15
Personnel attributes		
Analyst capabilities	1.46	.71
Application experience	1.29	.82
Programmer capability	1.42	.70
Virtual machine experience	1.21	.90
Programming language experience	1.14	.95
Project attributes		
Methodology	1.24	.82
Software tools	1.24	.83
Required development schedule	1.23	1.10

COCOMO II

- COCOMO/SCM 14 -- COCOMO development is continuing*
- **Theme:** Software Cost, Schedule, and Process, October 26-28, 1999
- Last year's Forum introduced a number of emerging extensions to the COCOMO II estimation model. These included COCOTS for COTS integration, CORADMO for Rapid Application Development, COQUALMO for quality estimation, and COSSEMO for stagewise schedule and effort estimation. Discussions at the Forum indicated that these and other cost and schedule estimation models need to address the effects of process selection on cost, schedule, and quality. The choice of process objectives (optimize cost, schedule, or quality) and process architectures (waterfall, incremental, evolutionary, spiral) can have significant effects on a product's overall cost, schedule, and quality, and on the distribution of cost and schedule by stage.
- This year's Forum particularly solicits presentations addressing these cost/schedule/process issues, either via experience reports or modeling and data analysis. Besides these topics, the Forum also solicits papers on other software cost estimation-related topics such as cost models for new software approaches (object-oriented, COTS integration, software product lines); software sizing; and relations between software cost estimation and software risk assessment, requirements, architecture, value assessment, and quality assessment.
- * - Email announcement, September 13, 1999

RECENT COST MODELS

Around 1995 Boehm started to work on a revision to COCOMO called COCOMO-2. Initially, it did not do as well as COCOMO in average error in estimating effort.

Now working on: (<http://sunset.usc.edu>)

- COCOTS – Component based cost model
- COQUALMO – Quality cost model

USC (Boehm), University and Fraunhofer Center about to start a new NSF project to work on these and related models.

More later this semester.

FUNCTION POINTS

An alternative estimating model is based upon function points. The idea is computation issues are not critical, so effort depends upon the data that is processed.

Most useful in data processing environments.

Function Point algorithms are managed by IFPUG (International Function Point Users Group). Function points are viewed as ranging from an important technology to a religion.

COUNTING FUNCTION POINTS

Basic process is:

- **Determine function point applicability (new application, enhanced application, installed application)**
- **Identify the application boundary**
- **Identify all data functions and complexity**
- **Identify all transactions and complexity**
- **Determine unadjusted function point count**
- **Determine value added factor**
- **Calculate adjusted function point count**

ADJUSTING VALUE

For unadjusted FP count, compute 5 sets of items:

- Logical input files
- External output files
- External inquiry files
- Internal data (data types)
- External data (data types)

FINAL FUNCTION POINT VALUE

For each data item determine whether the item is low, medium or heavy and multiply by an appropriate factor:

	Low	Medium	Heavy
Logical input	7	10	15
External output	5	7	10
External inquiry	3	4	6
Data in	3	4	6
Data out	4	5	7

From this unadjusted count, determine the value factor using 14 attributes, each rated from 0 to 5. Some of the attributes are: data communications, distributed processing, heavily used configuration, transaction rate, online data entry, end user efficiency, complex processing, ...

Function Points Calculation

Application Characteristics

DATA COMMUNICATIONS	ON-LINE UPDATE
DISTRIBUTED DATA OR PROCESSING	COMPLEX PROCESSING
PERFORMANCE OBJECTIVES	REUSABILITY
HEAVILY-USED CONFIGURATION	CONVERSION & INSTALLATION EASE
TRANSACTION RATE	OPERATIONAL EASE
ON-LINE DATA ENTRY	MULTIPLE-SITE
END USER EFFICIENCY	FACILITATE CHANGE



INFLUENCE SCALE

0	NONE
1	INSIGNIFICANT, MINOR
2	MODERATE
3	AVERAGE
4	SIGNIFICANT
5	STRONG THROUGHOUT

Function Points Calculation

FUNCTION POINTS =

$(\Sigma \text{ INPUTS} * \text{WEIGHTS} + _ \text{ OUTPUTS} * \text{WEIGHTS}$

$\Sigma \text{ QUERIES} * \text{WEIGHTS} + _ \text{ FILES} * \text{WEIGHTS} +$

$\Sigma \text{ INTERFACES} * \text{WEIGHTS}) * (0.65 + 1\% \text{ TOTAL INFLUENCE})$

FUNCTION POINTS - SUMMARY

This gives a typical value of around 15 for a simple system and from 30 to 60 for a complex system.

The adjusted FP count is $\text{Value-factor} \times .01 + .65$ which gives an adjusted function point rate from about 90% to 125% of the unadjusted count.

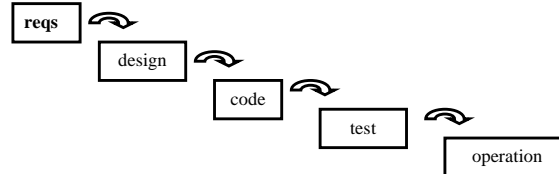
Using a figure of 20-30 function points per month, the effort can be computed.

Note that this is a gross simplification of the process, but it is the basic model that is applied.

LIFE CYCLE MODELS

Waterfall – 1970 – Royce

No one really uses it, but it is a simple abstraction of the process.

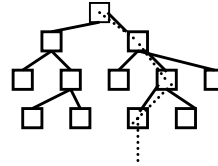


Problems with it are:

- Feedback – need to fix previous stages as errors found in later stages
- Need to understand final architecture and resources to build specifications and design
- Often need to build subpieces first before entire system is built

THREADS

Threads (build one path through the design tree) first, Builds (build part of functionality first) are both variations to this model where each part sort of follows waterfall design.



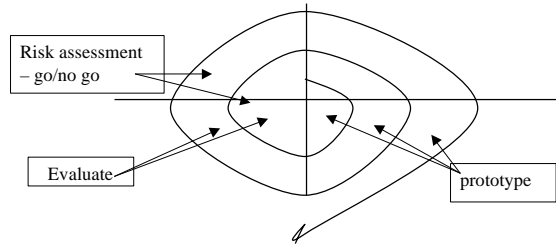
SPIRAL MODEL

Activities are similar to waterfall, but emphasis is on risk reduction. At each stage, build prototype, evaluate prototype, and make go-no go decision as to continuing with project.

Note that emphasis of waterfall model is on products – a specification milestone with a specification document, a design review with a design document, a code review with source code listings. There is little emphasis on feasibility of continuing with each review.

SPIRAL MODEL FIGURE

- Risk assessment drives decision making



“MICROSOFT” MODEL

“Microsoft” model (for lack of a better name)

- Build product
- Write requirements

Emphasis here is critical time to market issues. Builds done daily, so a complete product always ready. New features added when ready. If a build fails, programmer must come in and fix. In this market, time to leisurely do requirements review, develop specifications, then build just doesn't work.

ISSUES WITH THESE MODELS

Don't handle current interests –

Reuse – use existing code for new system

COTS – Commercial off the shelf software – e.g., Don't need to build a user interface, can just use HTML and existing web browsers for many applications.

In both cases, design effort affected by existing components.

REUSE

Reuse is not free (which many managers assume). You have to integrate the new code into the existing components. What if COTS does not do what you need? You need to write “glue code” to make it work. What if you update system and COTS vendor doesn't? What if COTS vendor goes out of business?

How to estimate effort and manage people when you have existing code to use?

How do you test a system with COTS you never wrote?