Black art of software estimation

You cannot predict nor control what you cannot measure
Fenton and Pfleeger, „Software Metrics“, 2nd Edition

Josip Šaban, M. Sc. computing, MBA, PMP
Erste Group IT, Vienna
Meridian Data, Zagreb

www.linkedin.com/in/josipsaban
josip.saban@erstegroup.com
meridiandatasoftware@gmail.com

About me

- FER Zagreb (2004), Cotrugli MBA (2011)
- MC* (SQL Server, Biztalk, Project Server, .NET), MCT from 2012 until 2014
- CompTIA Project+, PMP, Prince2 Foundation, ITIL Foundation
- Coursera - finished 9 courses and 3 specializations (big fan☺)
- Worked on vendor and client side, from startups to corporate sector, from operational to management roles
- Currently working in IT department in Erste Group IT in Vienna
- Also owner of sole proprietorship Meridian Data
- Private interests include soccer, judging tennis and travelling

Lecture name origin

- Lecture name taken from one of the best books in the field
  - MS Press - Software Estimation - Demystifying The Black Art
- Other key books (not many on this topic)
  - Wiley & Sons - Software Measurement And Estimation - A Practical Approach
  - J. Ross Publishing - Softwa[illegible]imation Best Practices Tools Techniques - A Complete Guide For Software Project Estimators
  - Springer - Software Cost Estimation Benchmarking and Risk Assessment
  - Auberback - Software Sizing, Estimation, and Risk Management - When Performance Is Measured Performance Improves
  - ISBSG - Practical Software Project Estimation - A Toolkit For Estimating Software Development

About this lecture

- Scenario one
  - You are a software manager responsible for building a new system and you need to tell the sales team how much effort it is going to take and how soon it can be ready
- Scenario two
  - You are responsible for making a go/no-go decision on releasing either a new system or an upgrade to an existing system
- How do you answer these questions and are you confident in your decisions?

Lecture motivation

- This is a senior level lecture, technology agnostic
- All too often, we are faced with situations where we could rely only on our intuition and gut feelings rather than managing by the numbers
- We will discuss required tools, data, and quantative measurements to make these kinds of decisions in software projects

Lecture motivation

- Managing your software by the numbers enables you to have repeatable results and continuous improvement
- Software engineers need to be skilled in estimation and measurement
  - Understand the activities and risks involved in software development
  - Predicting and controlling the activities
  - Managing the risks
  - Delivering reliably
  - Managing proactively to avoid crises

SO LET'S BEGIN…FROM BASICS

What you measure is what you get.

R.S. Kaplan and D.P. Norton, "The balanced scorecard - measures that drive performance," The Harvard Business Review, 1992.

Introduction

- To define an appropriate measurements program you need to answer the following questions
  - Who is the customer for the metrics
  - What are their goals with respect to the product, process, or resource under measurement
  - What metrics, when collected, will demonstrate whether or not the goal has been or is being met

Importance of estimations

- During the planning phase of a project, it is necessary to make „first guess” about cost and time
- Estimations are often the basis for the decision to start a project
- Estimations are the foundation for project planning and for further actions
  - Estimating is one of the core tasks of project management, but still considered as black magic!

What do we get out of the estimate

- Components
- Size
- Effort needed
- Cost
- Time
- Risk
- Data for future improvements

Who measures

- Recommended: Project Lead
  - Person responsible for completion of estimated task
- Often: You!

Why do we measure

- Estimation gives the Customer:
  - Budget
  - “Quote” to compare with
- Estimation gives the Supplier:
  - Budget
  - Chance to evaluate risks/rewards of a project
  - Way to keep track of progress
- Why is this Important to You?
  - It will be a part of your job as Software Engineers!

When do we measure

- Before Development
  - When project is acquired from customer
  - Following project Approval for internal projects
- During Development
  - Decreases Uncertainty (we discuss this later in more detail)
- Estimating… After Development?
  - Not as common - but very beneficial
  - Compare actual cost size time to estimated values
  - Improves knowledge for future estimations

The Challenges

- Many variables
  - Sizing
  - Development Environment
  - Support Environment
  - Staff (how much/how talented)
  - Project „boredom”

Components of an Estimation

- Cost
  - Personnel (in person days or valued in personnel cost)
    - Person day: Effort of one person per working day
  - Material (PCs, software, tools etc.)
  - Extra costs (travel expenses etc.)
- Development Time
  - Project duration
  - Dependencies
- Infrastructure
  - Rooms, technical infrastructure, especially in offshore scenarios

Estimating Development Time

- Development time often estimated by formula
  - Duration = Effort / People
- Problem with formula, because:
  - A larger project team increases communication complexity which usually reduces productivity
- Therefore it is not possible to reduce duration arbitrarily by adding more people to a project
  - Always valid: „Nine woman will not give birth in one month”

Estimating Personnel Cost

- Personnel type: Team leader, application domain expert, analyst, designer, programmer, tester…
  - Cost rate: Cost per person per day
- 2 alternatives for cost rate:
  - Single cost rate for all types (no differentiation necessary)
  - Assign different cost rates to different personnel types based on experience, qualification and skills
- Personnel cost: person days x cost rate.

Challenge - size estimation

- Not done as often as effort estimation
- Why do we do it?
  - Drives Scheduling and Cost Estimations
    - Size -> Effort -> Cost
- How do we size Software?

Sizing – „the old way” – lines of code

- How many lines are in STRNCAT function

```
/* Strncat() appends up to count characters from string
src to string dest, and then appends a terminating null
character. If copying takes place between objects that
overlap, the behavior is undefined. */
char *strncat (char *dest, const char *src, size_t count)
  {
      char *temp=dest;
      if (count) {
              while (*dest)
                      dest++;
              while ((*dest++=*src++)) {
                      if (--count == 0) {
                              *dest='\0';
                              break;
                              }
              }
              } return temp;
  }
```

- Depends ☺

Sizing – „the old way” – lines of code

- Original method of estimating size
- Program Size = LOC needed for functionality
- How many LOC from the experiment?
  - The world may never know…

Calculating the LOC Estimate

$$Est.LOC = \frac{4(likely\ LOC) + minimum\ LOC + maximum\ LOC}{6}$$

Module 1 Estimated LOC				
Functionality	**Min LOC**	**Max. LOC**	**Likely LOC**	**Estimated LOC**
F1	100	150	125	140
F2	95	120	105	105
TOTAL				**245**

- Alternative - Average between Min. and Max. LOC

LOC - Pros/Cons

- Pros
  - Beneficial for Real Time/Embedded Applications
  - Can be used to create Historical data for future estimates
  - Easy to count
- Cons
  - No defined “Line”
  - Doesn’t promote code optimizations

Estimating the effort

- Person Hours/Months/[Unit of time]
- Derives Schedule, Cost…

Method - Analogy

- How it’s done:
  - Use knowledge of previous projects to come up with estimates
- What is needed for success:
  - Company must keep records
  - Similar Projects must have been made

Method - Analogy

- Parameters for Analyzing
  - Project Type
    - Full life cycle
    - Implementation
    - Conversion
    - Port
    - Migration
  - Client Information
    - Application Domain
    - Size of client's organization
    - Number of client locations
  - Development Platform

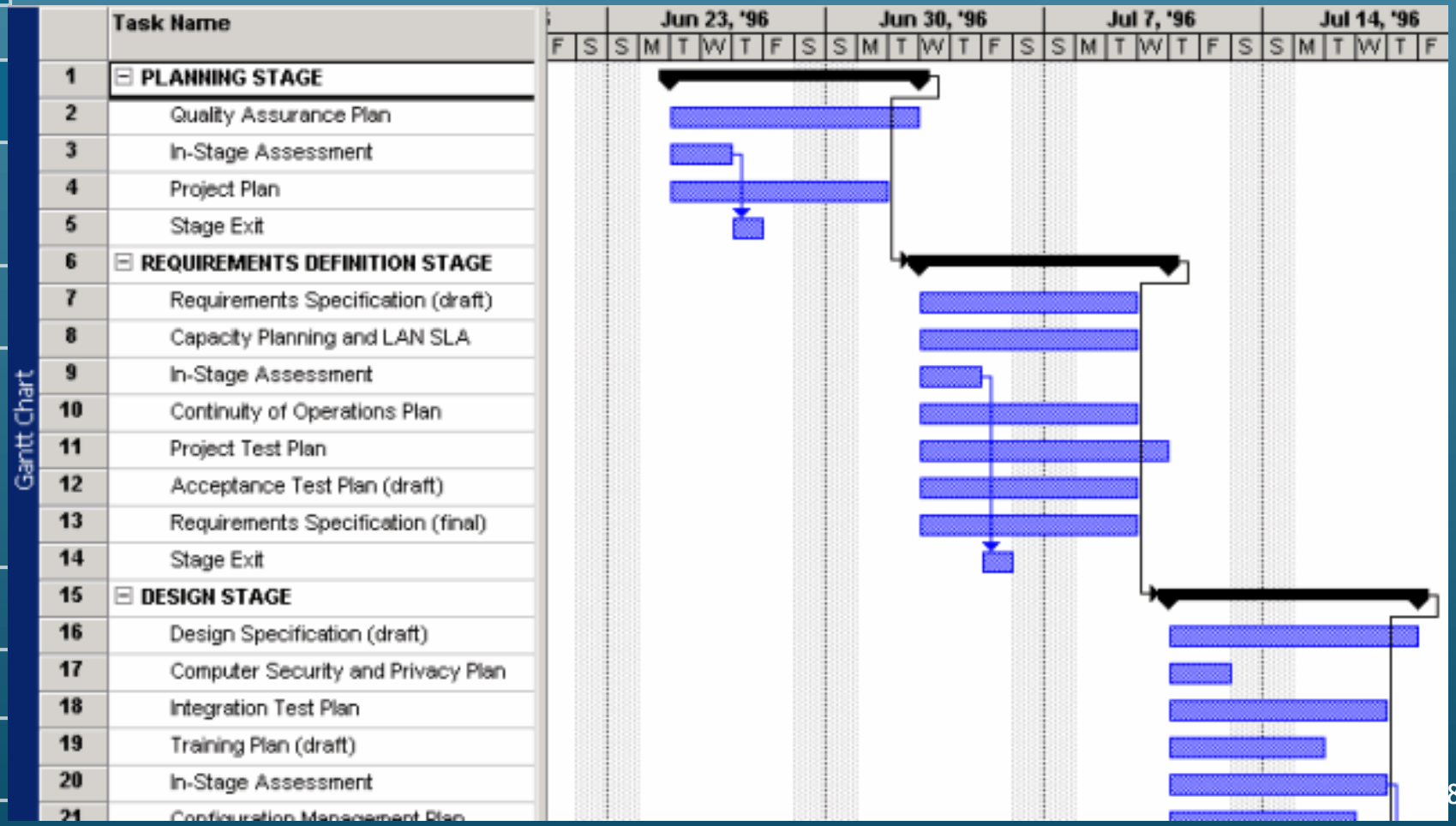
Analogy - Strengths/Weaknesses

- Strengths:
  - Reliable
  - Intuitive
  - Estimation Data can be purchased
- Weaknesses
  - Poor Record Keeping = bad estimates
  - Not the best for new organizations
  - Time consuming

Method - Expert Judgement (Delphi)

- How it's done:
  - Work with Subject Matter Experts (SMEs) to formulate estimates
  - If a small team of SMEs is used, a common estimate is derived
- Works well because:
  - No historical data needed
  - If proper SME is chosen - Extremely accurate
  - Quick
- Inaccurate because:
  - SME can be biased / inconsistent / overconfident

Methods: Activity and Work Decomposition



Methods: Activity and Work Decomposition

- How it's done:
  - Outline the needed tasks involved for the project
  - Should be done by person who ensures the tasks are accomplished.
  - Use knowledge of needed work and skill set of the implementers to derive time estimates.
- Pros:
  - Estimator is knowledgeable of the work at hand
  - No explicit sizing such as Lines of Code count is needed
- Limitations:
  - Subjective - accuracy is up to the estimator's opinion
  - No sizing

Methods: Top-Down

- How it's done:
  - Decomposition of system into smaller components
- Works well because:
  - Estimates are linked to requirements
- Inaccurate when:
  - Good requirements aren't available
  - Bias may lead to underestimation

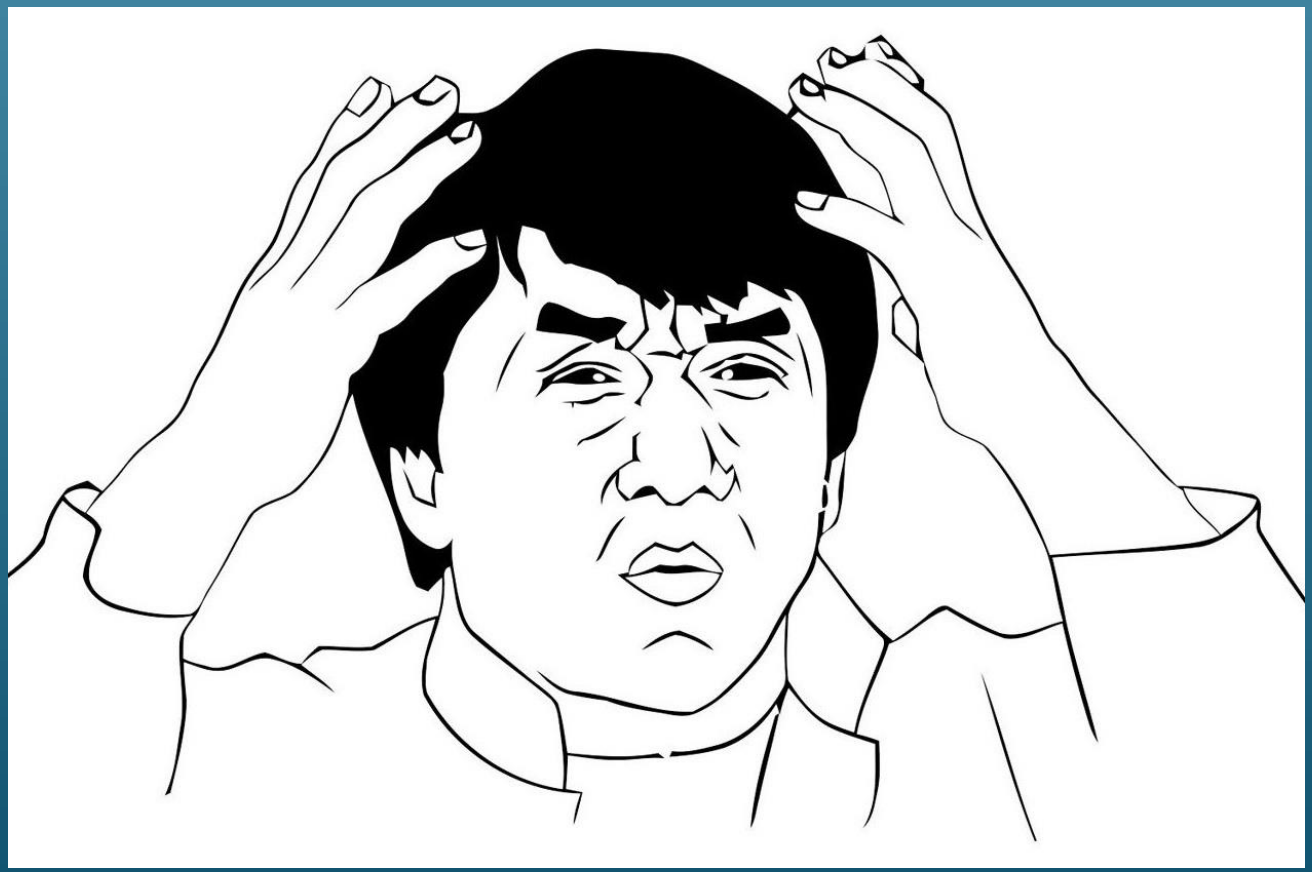
Methods: Bottom-Up

- How it's done:
  - Individuals evaluate each component of the entire system.
  - Sum to formulate project estimate
- Works well because:
  - Can be very accurate – detailed
  - Builds responsibility
- Issues:
  - Time consuming process
  - Details may not be available
  - Bias can lead to underestimation
  - Can miss integration costs

Methods: Design To Cost

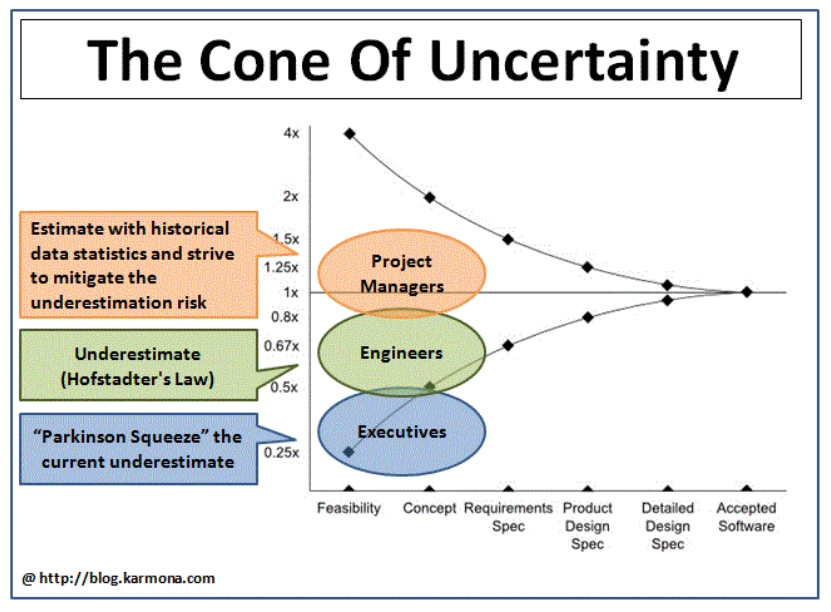
- How it's done:
  - Work with SMEs to find out how much we can give the customer for given budget
- Works well because:
  - The price is right!
- Issues:
  - Need knowledge of functionality cost

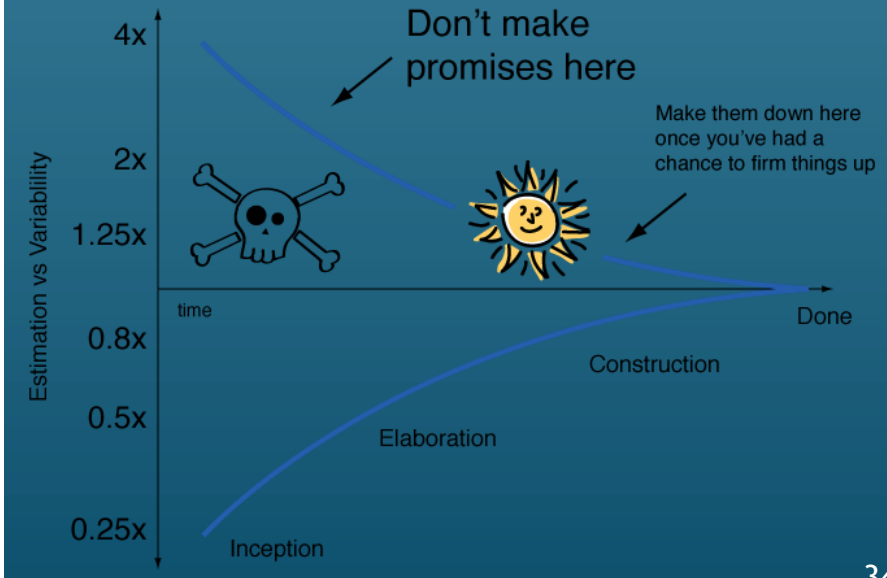
Estimating…Uncertainty?



The Cone of Uncertainty

- 400% to 25% !?
  - Current studies show closer to +100% / -50% at feasibility stage
  - Strong tendency to underestimate





Acknowledging Uncertainty

- Alternatives to giving a single number
  - Between X and Y person months
  - X% chance it will be under Y person months
  - Use the “pX” approach
    - “X” = X% chance of NOT exceeding the estimate
    - Example – You’re on a project and your estimate is 8 person months. You’ve researched that 10% of similar projects have been under budget.
      - p10 estimate = 7 staff months – 10% chance of being under, 90% being over

Dealing with Uncertainties

- Communicate in a way that makes them known, such as the pX method.
- Methodology > Gut Feelings
  - Work pressure drives over confidence
- Re-Estimate
  - Feasibility Stage - Ball Park
  - Requirements Stage – More Detailed
  - Design Stage - Refined
- Agile Environments:
  - Expect higher uncertainty

AND NOW…LET'S GET SERIOUS

The true measure of a man is what he would do if he knew he would never be caught

Lord Kelvin

Methods: Parametric Models

- How it's done:
  - Estimate by use of design parameters and mathematic formulas
- Works well because:
  - Fast
  - Ease of use
  - Can be re-useable
- Issues:
  - Models need to "fit the bill"
  - Can end up being very inaccurate through poor choice of parameters

COCOMO (COnstructive COst MOdel)

- Developed by Barry Boehm in 1981
- Also called COCOMO I or Basic COCOMO
- Top-down approach to estimate cost, effort and schedule of software projects, based on size and complexity of projects
- Assumptions:
  - Derivability of effort by comparing finished projects ("COCOMO database")
  - System requirements do not change during development
  - Exclusion of some efforts (for example administration, training, rollout, integration).

Calculation of Effort

- Estimate number of instructions
  - KDSI = “Kilo Delivered Source Instructions”
- Determine project complexity parameters: A, B
  - Regression analysis, matching project data to equation
- 3 levels of difficulty that characterize projects – templates for companies that don’t have resources for their A/B calculations
  - Simple project (“organic mode”)
  - Semi-complex project (“semidetached mode”)
  - Complex project (“embedded mode”)
- Calculate effort
  - Effort = $A * KDSI^B$
- Also called Basic COCOMO

Calculation of Effort in Basic COCOMO

- Formula: Effort = A * $KDSI^B$
  - Effort is counted in person months: 152 productive hours (8 hours per day, 19 days/month, less weekends, holidays, etc.)
  - A, B are constants based on the complexity of the project

Project Complexity	A	B
Simple	2.4	1.05
Semi-Complex	3.0	1.12
Complex	3.6	1.20

Calculation of Development Time

- Basic formula: $T = C * Effort^D$
  - T = Time to develop in months
  - C, D = constants based on the complexity of the project
  - Effort = Effort in person months (see slide before)

Project Complexity	C	D
Simple	2.5	0.38
Semi-Complex	2.5	0.35
Complex	2.5	0.32

Basic COCOMO Example

- Volume = 30000 LOC = 30KLOC
- Project type = Simple
- Effort = $2.4 * (30)^{1.05}$ = 85 PM
- Development Time = $2.5 * (85)^{0.38}$ = 13.5 months

=> Avg. staffing: 85/13.5 = 6.3 persons

=> Avg. productivity: 30000/85 = 353 LOC/PM

Other COCOMO Models

- Intermediate COCOMO
  - 15 additional cost drivers (parameters) yielding a multiplicative correction factor
  - Basic COCOMO is based on value of 1.00 for each of the cost drivers
- Detailed COCOMO
  - Multipliers depend on phase: Requirements; System Design; Detailed Design; Code and Unit Test; Integrate & Test; Maintenance

Steps in Intermediate COCOMO

- Basic COCOMO steps
  - Estimate number of instructions
  - Determine project complexity parameters: A, B
  - Determine level of difficulty that characterizes the project
- New step
  - Determine cost drivers
    - 15 cost drivers c1 , c1 …. c15
- Calculate effort
  - Effort = $A * KDSI^{B} * c1 * c1 …. * c15$

Calculation of Effort in Intermediate COCOMO

- Basic formula:
  - Effort = $A * KDSI^B * c_1 * c_1 \ldots * c_{15}$
  - Effort is measured in PM (person months, 152 productive hours (8 hours per day, 19 days/month, less weekends, holidays, etc.)
    - A, B are constants based on the complexity of the project

Project Complexity	C	D
Simple	2.4	1.05
Semi-Complex	3.0	1.12
Complex	3.6	1.20

Intermediate COCOMO: 15 Cost drivers

- Product Attributes
  - Required reliability
  - Database size
  - Product complexity
- Computer Attributes
  - Execution Time constraint
  - Main storage constraint
  - Virtual Storage volatility
  - Turnaround time
- Personnel Attributes
  - Analyst capability
  - Applications experience
  - Programmer capability
  - Virtual machine experience
  - Language experience

- Project Attributes
  - Use of modern programming practices
  - Use of software tools
  - Required development schedule

- Rated on a qualitative scale between "very low" and "extra high„
- Associated values are multiplied with each other

COCOMO II

- Revision of COCOMO I in 1997
- Provides three models of increasing detail
  - Application Composition Model
    - Estimates for prototypes based on GUI builder tools and existing components
  - Early Design Model
    - Estimates before software architecture is defined
    - For system design phase, closest to original COCOMO, uses function points as size estimation
  - Post Architecture Model
    - Estimates once architecture is defined
    - For actual development phase and maintenance; Uses FPs or SLOC as size measure
- Estimator selects one of the three models based on current state of the project

COCOMO II

- Targeted for iterative software lifecycle models
  - Boehm's spiral model
  - COCOMO I assumed a waterfall model
    - 30% design; 30% coding; 40% integration and test
- COCOMO II includes new costs drivers to deal with
  - Team experience
  - Developer skills
  - Distributed development
- COCOMO II includes new equations for reuse
  - Enables build vs. buy trade-offs

COCOMO II: Added Cost drivers

- Development flexibility
- Team cohesion
- Developed for reuse
- Precedent
- Architecture & risk resolution
- Personnel continuity
- Documentation match life cycle needs
- Multi-Site development

Advantages of COCOMO

- Appropriate for a quick, high-level estimation of project costs
- Fair results with smaller projects in a well known development environment
  - Assumes comparison with past projects is possible
- Covers all development activities (from analysis to testing)
- Intermediate COCOMO yields good results for projects on which the model is based

Problems with COCOMO

- Judgment requirement to determine the influencing factors and their values
- Experience shows that estimation results can deviate from actual effort by a factor of 4
- Some important factors are not considered:
  - Skills of team members, travel, environmental factors, user interface quality, overhead cost

Problems with COCOMO

- Judgment requirement to determine the influencing factors and their values
- Experience shows that estimation results can deviate from actual effort by a factor of 4
- Some important factors are not considered:
  - Skills of team members, travel, environmental factors, user interface quality, overhead cost

Function Point Analysis

- Developed by Allen Albrecht, IBM Research, 1979
- Technique to determine size of software projects
  - Size is measured from a functional point of view
  - Estimates are based on functional requirements
- Albrecht originally used the technique to predict effort
  - Size is usually the primary driver of development effort

Function Point Analysis

- Independent of
  - Implementation language and technology
  - Development methodology
  - Capability of the project team
- A top-down approach based on function types
  - Three steps: Plan the count, perform the count, estimate the effort.

Steps in Function Point Analysis

- Plan the count
  - Type of count: development, enhancement, application
  - Identify the counting boundary
  - Identify sources for counting information: software, documentation and/or expert
- Perform the count
  - Count data access functions
  - Count transaction functions

Steps in Function Point Analysis

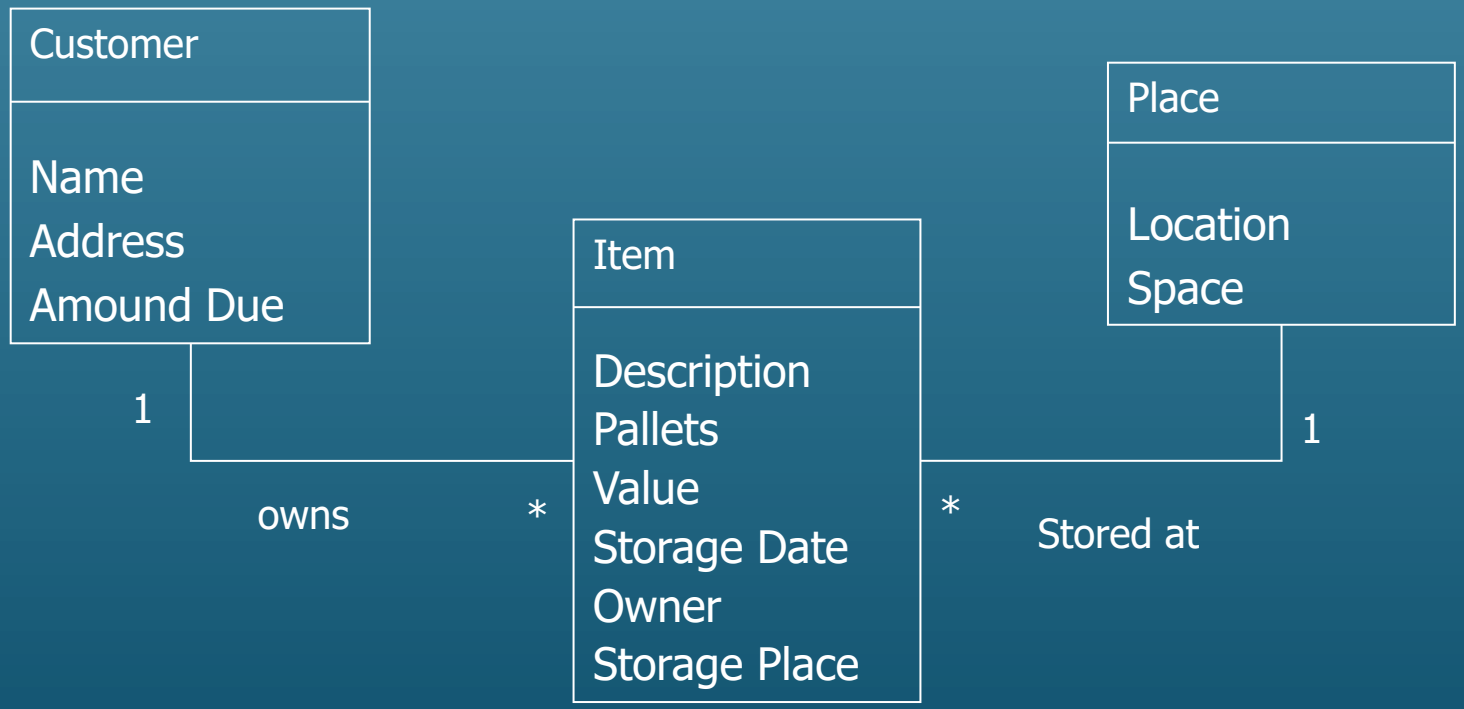
- Estimate the effort
  - Compute the unadjusted function points (UFP)
  - Compute the Value Added Factor (VAF)
  - Compute the adjusted Function Points (FA)
  - Compute the performance factor
  - Calculate the effort in person days

Function Types

- Data function types
  - # of internal logical files (ILF)
  - # of external interface files (EIF)
- Transaction function types
  - # of external input (EI)
  - # of external output (EO)
  - # of external queries (EQ)
- Calculate the UFP (unadjusted function points):
  - $UFP = a \cdot EI + b \cdot EO + c \cdot EQ + d \cdot ILF + e \cdot EIF$

a-f are unadjusted weight factors (see next slide)

Object Model Example



Calculate the Unadjusted Function Points

Function Type	Number		Weight Factors: simple	average	complex		
External Input (EI)		x	3	4	6	=	
External Output (EO)		x	4	5	7	=	
External Queries (EQ)		x	3	4	6	=	
Internal Datasets (ILF)		x	7	10	15	=	
Interfaces (EIF)		x	5	7	10	=	

Unadjusted Function Points (UFP) =

Mapping Functions to Transaction Types

Add Customer
Change Customer
Delete Customer
Receive payment
Deposit Item
Retrieve Item
Add Place
Change Place Data
Delete Place

External Inputs

Print Customer item list
Print Bill
Print Item List

External Outputs

Query Customer
Query Customer's items
Query Places
Query Stored Items

External Inquiries

Advantages of Function Point Analysis

- Independent of implementation language and technology
- Estimates are based on design specification
  - Usually known before implementation tasks are known
- Users without technical knowledge can be integrated into the estimation process
  - Incorporation of experiences from different organizations
- Easy to learn
  - Limited time effort

Disadvantages of Function Point Analysis

- Complete description of functions necessary
  - Often not the case in early project stages -> especially in iterative software processes
- Only complexity of specification is estimated
  - Implementation is often more relevant for estimation
- High uncertainty in calculating function points:
  - Weight factors are usually deducted from past experiences (environment, used technology and tools may be out-of-date in the current project)
- Does not measure the performance of people

Online Availability of Estimation Tools

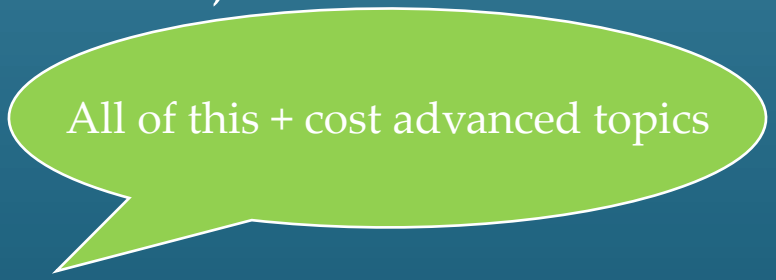
- Basic and Intermediate COCOMO I (JavaScript)
  - http://www1.jsc.nasa.gov/bu2/COCOMO.html
  - http://ivs.cs.uni-magdeburg.de/sw-eng/us/java/COCOMO/index.shtml
- COCOMO II (Unix, Windows and Java)
  - http://sunset.usc.edu/available_tools/index.html
- Function Point Calculator (Java)
  - http://ivs.cs.uni-magdeburg.de/sw-eng/us/java/fp/

Why Estimates Fail – A recap

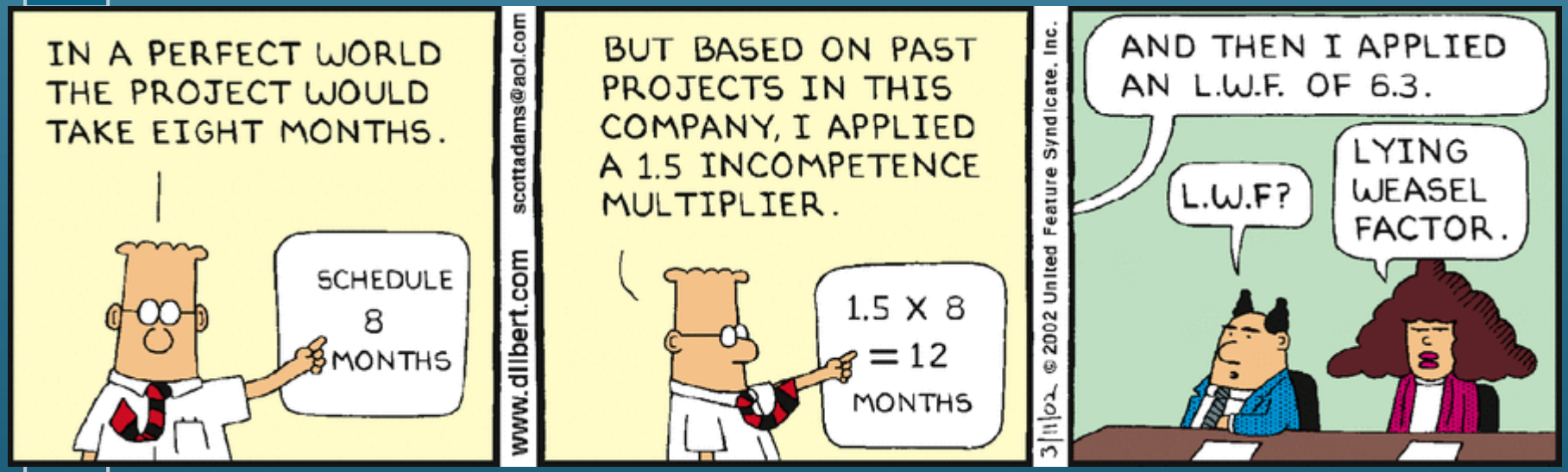
- Little / Misused historical data
  - Record estimates + actual data
  - Make sure data “fits the bill”
- Over-Optimistic / Hopeful management
  - Avoid gut feelings – use the methods
  - Avoid over-confidence
- Not using the estimate!
  - Don’t Confuse targets with estimates
- Not updating the estimate
  - Acknowledge uncertainty in estimation
  - Estimate often – it will become more accurate

What's next

- Cost
  - Personnel (in person days or valued in personnel cost)
    - Person day: Effort of one person per working day
  - Material (PCs, software, tools etc.)
  - Extra costs (travel expenses etc.)
- Development Time
  - Project duration
  - Dependencies
- Infrastructure
  - Rooms, technical infrastructure, especially in offshore scenarios



L.W.F Factor


IN A PERFECT WORLD THE PROJECT WOULD TAKE EIGHT MONTHS.
SCHEDULE 8 MONTHS
www.dilbert.com scottadams@aol.com
BUT BASED ON PAST PROJECTS IN THIS COMPANY, I APPLIED A 1.5 INCOMPETENCE MULTIPLIER.
1.5 X 8 = 12 MONTHS
3/11/02 © 2002 United Feature Syndicate, Inc.
AND THEN I APPLIED AN L.W.F. OF 6.3.
L.W.F?
LYING WEASEL FACTOR.

Thank you for your time