

# IT607 - Software Engineering

Kavi Arya  
KReSIT, IIT-Bombay



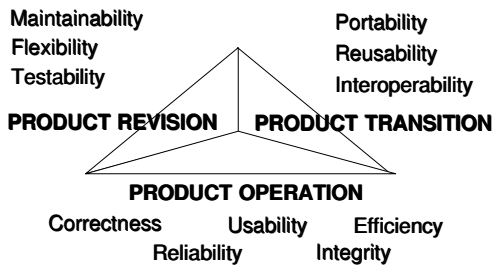
1

# Product Metrics for Software

Adapted from © R.S. Pressman & Associates, Inc. *Software Engineering: A Practitioner's Approach*.

2

## McCall's Triangle of Quality



3

## A Comment

McCall's quality factors were proposed in the early 1970s. They are as valid today as they were in that time. It's likely that software built to conform to these factors will exhibit high quality well into the 21st century, even if there are dramatic changes in technology.

4

## Measures, Metrics and Indicators

- A *measure* provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process
- The IEEE glossary defines a *metric* as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."
- An *indicator* is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself

5

## Measurement Principles

- The objectives of measurement should be established before data collection begins;
- Each technical metric should be defined in an unambiguous manner;
- Metrics should be derived based on a theory that is valid for the domain of application (e.g., metrics for design should draw upon basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable);
- Metrics should be tailored to best accommodate specific products and processes [BAS84]

6

## Measurement Process

- **Formulation.** The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
- **Collection.** The mechanism used to accumulate data required to derive the formulated metrics.
- **Analysis.** The computation of metrics and the application of mathematical tools.
- **Interpretation.** The evaluation of metrics results in an effort to gain insight into the quality of the representation.
- **Feedback.** Recommendations derived from the interpretation of product metrics transmitted to the software team.

7

## Goal-Oriented Software Measurement

- The Goal/Question/Metric Paradigm
  - (1) establish an explicit measurement *goal* that is specific to the process activity or product characteristic that is to be assessed
  - (2) define a set of *questions* that must be answered in order to achieve the goal, and
  - (3) identify well-formulated *metrics* that help to answer these questions.
- Goal definition template
  - Analyze {the name of activity or attribute to be measured}
  - for the purpose of {the overall objective of the analysis}
  - with respect to {the aspect of the activity or attribute that is considered}
  - from the viewpoint of {the people who have an interest in the measurement}
  - in the context of {the environment in which the measurement takes place}.

8

## Metrics Attributes

- *simple and computable.* It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time
- *empirically and intuitively persuasive.* The metric should satisfy the engineer's intuitive notions about the product attribute under consideration
- *consistent and objective.* The metric should always yield results that are unambiguous.
- *consistent in its use of units and dimensions.* The mathematical computation of the metric should use measures that do not lead to bizarre combinations of unit.
- *programming language independent.* Metrics should be based on the analysis model, the design model, or the structure of the program itself.
- *an effective mechanism for quality feedback.* That is, the metric should provide a software engineer with information that can lead to a higher quality end product

9

## Collection and Analysis Principles

- Whenever possible, data collection and analysis should be automated;
- Valid statistical techniques should be applied to establish relationship between internal product attributes and external quality characteristics
- Interpretative guidelines and recommendations should be established for each metric

10

## Analysis Metrics

- Function-based metrics: use the function point as a normalizing factor or as a measure of the "size" of the specification
- Specification metrics: used as an indication of quality by measuring number of requirements by type

11

## Function-Based Metrics

- The *function point metric* (FP), first proposed by Albrecht [ALB79], can be used effectively as a means for measuring the functionality delivered by a system.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity
- Information domain values are defined in the following manner:
  - number of external inputs (EIs)
  - number of external outputs (EOs)
  - number of external inquiries (EQs)
  - number of internal logical files (ILFs)
  - Number of external interface files (EIFs)

12

## Metrics for OO Design-I

- Whitire [WHI97] describes nine distinct and measurable characteristics of an OO design:
  - Size
    - Size is defined in terms of four views: population, volume, length, and functionality
  - Complexity
    - How classes of an OO design are interrelated to one another
  - Coupling
    - The physical connections between elements of the OO design
  - Sufficiency
    - "the degree to which an abstraction possesses the features required of it, or the degree to which a design component possesses features in its abstraction, from the point of view of the current application."
  - Completeness
    - An indirect implication about the degree to which the abstraction or design component can be reused

13

## Function Points

Information Domain Value	Count	Weighting factor			=	[ ]
		simple	average	complex		
External Inputs ( EIs)	[ ]	3	3	4	6	[ ]
External Outputs ( EOs)	[ ]	3	4	5	7	[ ]
External Inquiries ( EQs)	[ ]	3	3	4	6	[ ]
Internal Logical Files ( ILFs)	[ ]	3	7	10	15	[ ]
External Interface Files ( EIFs)	[ ]	3	5	7	10	[ ]
Count total	→					[ ]

14

## Architectural Design Metrics

- Architectural design metrics
  - Structural complexity =  $g(\text{fan-out})$
  - Data complexity =  $f(\text{input \& output variables, fan-out})$
  - System complexity =  $h(\text{structural \& data complexity})$
- HK metric: architectural complexity as a function of fan-in and fan-out
- Morphology metrics: a function of the number of modules and the number of interfaces between modules

15

## Metrics for OO Design-II

- Cohesion
  - The degree to which all operations working together to achieve a single, well-defined purpose
- Primitiveness
  - Applied to both operations and classes, the degree to which an operation is atomic
- Similarity
  - The degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose
- Volatility
  - Measures the likelihood that a change will occur

16

## Distinguishing Characteristics

Berard [BER95] argues that the following characteristics require that special OO metrics be developed:

- Localization—the way in which information is concentrated in a program
- Encapsulation—the packaging of data and processing
- Information hiding—the way in which information about operational details is hidden by a secure interface
- Inheritance—the manner in which the responsibilities of one class are propagated to another
- Abstraction—the mechanism that allows a design to focus on essential details

17

## Class-Oriented Metrics

Proposed by Chidamber and Kemerer:

- weighted methods per class
- depth of the inheritance tree
- number of children
- coupling between object classes
- response for a class
- lack of cohesion in methods

18

## Class-Oriented Metrics

*Proposed by Lorenz and Kidd [LOR94]:*

- class size
- number of operations overridden by a subclass
- number of operations added by a subclass
- specialization index

19

## Class-Oriented Metrics

*The MOOD Metrics Suite*

- Method inheritance factor
- Coupling factor
- Polymorphism factor

20

## Operation-Oriented Metrics

*Proposed by Lorenz and Kidd [LOR94]:*

- average operation size
- operation complexity
- average number of parameters per operation

21

## Component-Level Design Metrics

- Cohesion metrics: a function of data objects and the locus of their definition
- Coupling metrics: a function of input and output parameters, global variables, and modules called
- Complexity metrics: hundreds have been proposed (e.g., cyclomatic complexity)

22

## Interface Design Metrics

- Layout appropriateness: a function of layout entities, the geographic position and the "cost" of making transitions among entities

23

## Code Metrics

- Halstead's Software Science: a comprehensive collection of metrics all predicated on the number (count and occurrence) of operators and operands within a component or program
  - It should be noted that Halstead's "laws" have generated substantial controversy, and many believe that the underlying theory has flaws. However, experimental verification for selected programming languages has been performed (e.g. [FEL89]).

24

## Metrics for Testing

- Testing effort can also be estimated using metrics derived from Halstead measures
- Binder [BIN94] suggests a broad array of design metrics that have a direct influence on the “testability” of an OO system.
  - Lack of cohesion in methods (LCOM).
  - Percent public and protected (PAP).
  - Public access to data members (PAD).
  - Number of root classes (NOR).
  - Fan-in (FIN).
  - Number of children (NOC) and depth of the inheritance tree (DIT).

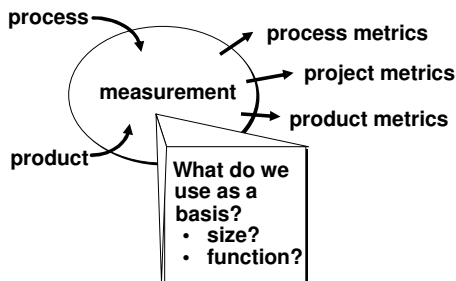
25

## Process and Project Metrics

copyright © R.S. Pressman & Associates, Inc.

26

## A Good Manager Measures



27

## Why Do We Measure?

- assess the status of an ongoing project
- track potential risks
- uncover problem areas before they go “critical,”
- adjust work flow or tasks,
- evaluate the project team’s ability to control quality of software work products.

28

## Process Measurement

- We measure the efficacy of a software process indirectly.
  - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
- Outcomes include
  - measures of errors uncovered before release of the software
  - defects delivered to and reported by end-users
  - work products delivered (productivity)
  - human effort expended
  - calendar time expended
  - schedule conformance
  - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

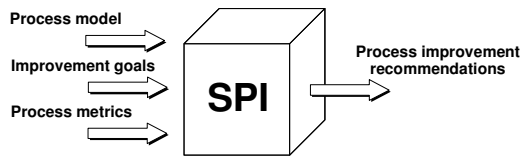
29

## Process Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered “negative.” These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

30

## Software Process Improvement



31

## Process Metrics

- Quality-related
  - focus on quality of work products and deliverables
- Productivity-related
  - Production of work-products related to effort expended
- Statistical SQA data
  - error categorization & analysis
- Defect removal efficiency
  - propagation of errors from process activity to activity
- Reuse data
  - The number of components produced and their degree of reusability

32

## Project Metrics

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- every project should measure:
  - *inputs*—measures of the resources (e.g., people, tools) required to do the work.
  - *outputs*—measures of the deliverables or work products created during the software engineering process.
  - *results*—measures that indicate the effectiveness of the deliverables.

33

## Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

34

## Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who have worked to collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered "negative." These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

35

## Typical Size-Oriented Metrics

- errors per KLOC (thousand lines of code)
- defects per KLOC
- \$ per LOC
- pages of documentation per KLOC
- errors per person-month
- Errors per review hour
- LOC per person-month
- \$ per page of documentation

36

## Typical Function-Oriented Metrics

- errors per FP (thousand lines of code)
- defects per FP
- \$ per FP
- pages of documentation per FP
- FP per person-month

37

## Comparing LOC and FP

Programming Language	LOC per Function point			
	avg.	median	low	high
Ada	154	-	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	63	53	77	-
JavaScript	58	63	42	75
Perl	60	-	-	-
PL1	78	67	22	263
Powerbuilder	32	31	11	105
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

Representative values developed by QSM

38

## Why Opt for FP?

- Programming language independent
- Used readily countable characteristics that are determined early in the software process
- Does not “penalize” inventive (short) implementations that use fewer LOC than other more clumsy versions
- Makes it easier to measure the impact of reusable components

39

## Object-Oriented Metrics

- Number of scenario scripts (use-cases)
- Number of support classes (required to implement the system but are not immediately related to the problem domain)
- Average number of support classes per key class (analysis class)
- Number of subsystems (an aggregation of classes that support a function that is visible to the end-user of a system)

40

## WebE Project Metrics

- Number of static Web pages (the end-user has no control over the content displayed on the page)
- Number of dynamic Web pages (end-user actions result in customized content displayed on the page)
- Number of internal page links (internal page links are pointers that provide a hyperlink to some other Web page within the WebApp)
- Number of persistent data objects
- Number of external systems interfaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

41

## Measuring Quality

- Correctness — the degree to which a program operates according to specification
- Maintainability—the degree to which a program is amenable to change
- Integrity—the degree to which a program is impervious to outside attack
- Usability—the degree to which a program is easy to use

42

## Defect Removal Efficiency

$$DRE = E / (E + D)$$

*E* is the number of errors found before delivery of the software to the end-user

*D* is the number of defects found after delivery.

43

## Metrics for Small Organizations

- time (hours or days) elapsed from the time a request is made until evaluation is complete,  $t_{queue}$
- effort (person-hours) to perform the evaluation,  $W_{eval}$
- time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel,  $t_{eval}$
- effort (person-hours) required to make the change,  $W_{change}$
- time required (hours or days) to make the change,  $t_{change}$
- errors uncovered during work to make change,  $E_{change}$
- defects uncovered after change is released to the customer base,  $D_{change}$

44

## Establishing a Metrics Program

- Identify your business goals.
- Identify what you want to know or learn.
- Identify your subgoals.
- Identify the entities and attributes related to your subgoals.
- Formalize your measurement goals.
- Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
- Identify the data elements that you will collect to construct the indicators that help answer your questions.
- Define the measures to be used, and make these definitions operational.
- Identify the actions that you will take to implement the measures.
- Prepare a plan for implementing the measures.

45

## Estimation for Software Projects

46

## Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

*So the end result gets done on time, with quality!*

47

## Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
  - Risk analysis is considered in detail in Chapter 25.
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources

48

## Project Planning Task Set-II

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates
- Develop a project schedule
  - Scheduling is considered in detail in Chapter 24.
    - Establish a meaningful task set
    - Define a task network
    - Use scheduling tools to develop a timeline chart
    - Define schedule tracking mechanisms

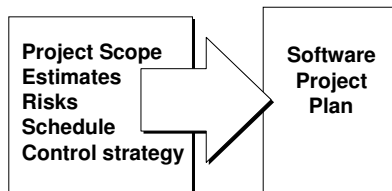
49

## Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
  - experience
  - access to good historical information (metrics)
  - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

50

## Write it Down!



51

## To Understand Scope ...

- Understand the customers needs
- understand the business context
- understand the project boundaries
- understand the customer's motivation
- understand the likely paths for change
- understand that ...

*Even when you understand,  
nothing is guaranteed!*

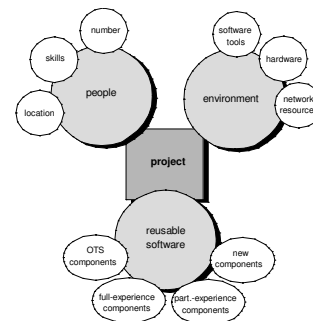
52

## What is Scope?

- *Software scope* describes
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the "content" that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of two techniques:
  - A narrative description of software scope is developed after communication with all stakeholders.
  - A set of use-cases is developed by end-users.

53

## Resources



54

## Project Estimation



- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process

55

## Estimation Techniques

- Past (similar) project experience
- Conventional estimation techniques
  - task breakdown and effort estimates
  - size (e.g., FP) estimates
- Empirical models
- Automated tools



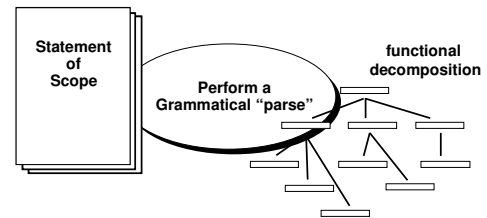
56

## Estimation Accuracy

- Predicated on ...
  - the degree to which the planner has properly estimated the size of the product to be built
  - the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects)
  - the degree to which the project plan reflects the abilities of the software team
  - the stability of product requirements and the environment that supports the software engineering effort.

57

## Functional Decomposition



58

## Conventional Methods: LOC/FP Approach

- compute LOC/FP using estimates of information domain values
- use historical data to build estimates for the project

59

## Example: LOC Approach

Function	Estimated LOC
user interface and control facilities (UI/CF)	2,300
two-dimensional geometric analysis (2D-GA)	5,300
three-dimensional geometric analysis (3D-GA)	6,600
database management (DBM)	3,200
computer graphics display facilities (CGDF)	4,900
peripheral control (PC)	2,100
design analysis modules (D&M)	6,400
<i>estimated lines of code</i>	<b>33,200</b>

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate = \$8000 per month, the cost per line of code is approximately \$13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is \$431,000 and the estimated effort is 54 person-months.

60

## Example: FP Approach

Information Domain	Value	opt.	likely	poss.	est. count	weight	FP count
number of inputs	20	24	30	24	4		97
number of outputs	10	15	22	16	5		78
number of inquiries	46	22	25	22	3		98
number of files	4	4	5	4	10		42
number of external interfaces	2	2	3	2	7		15
<b>count total</b>							<b>321</b>

The estimated number of FP is derived:

$$FP_{\text{estimated}} = \text{count-total} \cdot 3 [0.65 + 0.01 \cdot 3 \cdot S (F_i)]$$

$$FP_{\text{estimated}} = 375$$

organizational average productivity = 6.5 FP/pm.

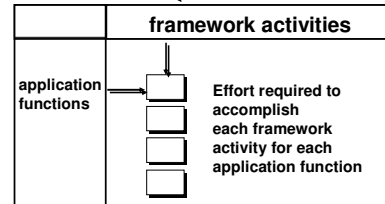
burdened labor rate = \$8000 per month, the cost per FP is approximately \$1230.

Based on the FP estimate and the historical productivity data, the total estimated project cost is \$461,000 and the estimated effort is 58 person-months.

61

## Process-Based Estimation

Obtained from "process framework"



62

## Process-Based Estimation Example

Activity	CC	Planning	Risk Analysis	Engineering	Construction Release	CE	Totals	
Task				analysis	design	code	test	
Function								
UICF				0.50	2.50	0.40	5.00	7.9
SDGA				0.75	4.00	0.60	2.00	7.35
SDGA				0.50	4.00	1.00	2.00	8.50
CCDF				0.50	3.00	1.00	1.50	6.00
DSM				0.50	3.00	0.75	1.50	5.75
PCF				0.25	2.00	0.50	1.50	4.25
DAM				0.50	2.00	0.50	2.00	5.00
<b>Totals</b>	<b>0.25</b>	<b>0.25</b>	<b>0.25</b>	<b>3.50</b>	<b>20.50</b>	<b>4.50</b>	<b>16.50</b>	<b>46.00</b>
<b>% effort</b>	<b>1%</b>	<b>1%</b>	<b>1%</b>	<b>8%</b>	<b>45%</b>	<b>10%</b>	<b>36%</b>	

CC = customer communication CE = customer evaluation

Based on an average burdened labor rate of \$8,000 per month, the total estimated project cost is \$368,000 and the estimated effort is 46 person-months.

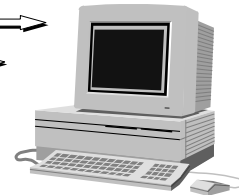
63

## Tool-Based Estimation

project characteristics →

calibration factors →

LOC/FP data →



64

## Estimation with Use-Cases

	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

Using 620 LOC/pm as the average productivity for systems of this type and a burdened labor rate of \$8000 per month, the cost per line of code is approximately \$13. Based on the use-case estimate and the historical productivity data, the total estimated project cost is \$552,000 and the estimated effort is 68 person-months.

65

## Empirical Estimation Models

General form:

$$\text{effort} = \text{tuning coefficient} \cdot \text{size}^{\text{exponent}}$$

usually derived as person-months of effort required

empirically derived

either a constant or a number derived based on complexity of project

usually LOC but may also be function point

66

## COCOMO-II

- COCOMO II is actually a hierarchy of estimation models that address the following areas:
  - *Application composition model.* Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - *Early design stage model.* Used once requirements have been stabilized and basic software architecture has been established.
  - *Post-architecture-stage model.* Used during the construction of the software.

67

## The Software Equation

A dynamic multivariable model

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4)$$

where

- E = effort in person-months or person-years
- t = project duration in months or years
- B = "special skills factor"
- P = "productivity parameter"

68

## Estimation for OO Projects-I

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented analysis modeling (Chapter 8), develop use-cases and determine a count.
- From the analysis model, determine the number of key classes (called analysis classes in Chapter 8).
- Categorize the type of interface for the application and develop a multiplier for support classes:
 

Interface type	Multiplier
■ No GUI	2.0
■ Text-based user interface	2.25
■ GUI	2.5
■ Complex GUI	3.0

69

## Estimation for OO Projects-II

- Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
- Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
- Cross check the class-based estimate by multiplying the average number of work-units per use-case

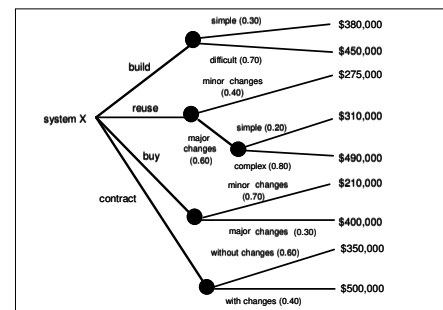
70

## Estimation for Agile Projects

- Each user scenario (a mini-use-case) is considered separately for estimation purposes.
- The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
- Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or "experience."
  - Alternatively, the 'volume' of the scenario can be estimated in LOC, FP or some other volume-oriented measure (e.g., use-case count).
- Estimates for each task are summed to create an estimate for the scenario.
  - Alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

71

## The Make-Buy Decision



72

## Computing Expected Cost

$$\text{expected cost} = \sum (\text{path probability})_i \times (\text{estimated path cost})_i$$

For example, the expected cost to build is:

$$\begin{aligned} \text{expected cost}_{\text{build}} &= 0.30 (\$380\text{K}) + 0.70 (\$450\text{K}) \\ &= \$429 \text{ K} \end{aligned}$$

similarly,

$$\begin{aligned} \text{expected cost}_{\text{bus}} &= \\ \text{expected cost}_{\text{buy}} &= \\ \text{expected cost}_{\text{rent}} &= \\ \text{expected cost}_{\text{r}} &= \end{aligned}$$