ops CONSULTING

Object-Oriented Estimation Techniques

Presented at the ISPA SCEA National Conference

Industry Hills, California

June 24 – 27, 2008

Leah Upshaw

OPS Consulting, L.L.C.

Agenda

- What is the Object-Oriented Design Paradigm?
- Why Estimate Software Development in its Native Format?
- Description of Object-Oriented Estimation Methods
- How to Apply Technology to an Organization's Estimation Processes and Environment

What is the Object-Oriented Design Paradigm?

Evolution of Software – Paradigms

ops CONSULTING

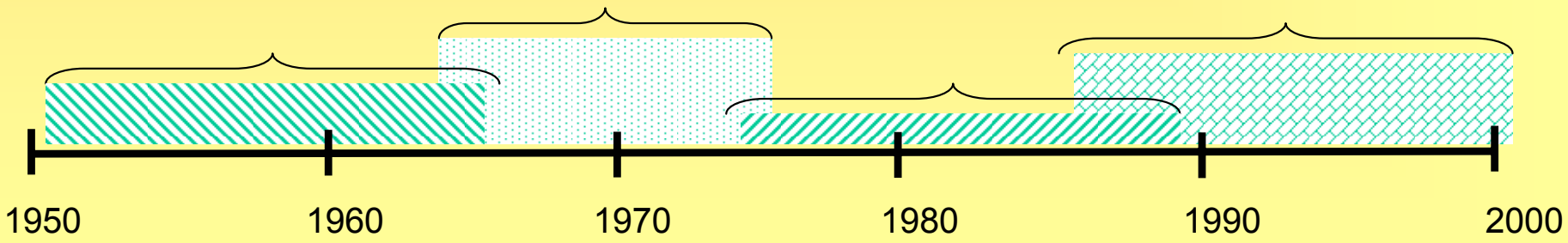
- Programming paradigms have evolved over the past 60 years and continue to evolve
- Object-Oriented design paradigm first proposed in 1960's, but was not used in the mainstream until early 1990's
- Transition due to increased size and complexity of systems

The early years
- Batch orientation
- Limited distribution
- Custom software

The second era
- Multiuser
- Real-time
- Database
- Product software

The third era
- Distributed systems
- Embedded "intelligence"
- Low cost hardware
- Consumer impact

The fourth era
- Powerful desk-top systems
- *Object-oriented technologies*
- Expert systems
- Artificial neural networks
- Parallel computing
- Network computers

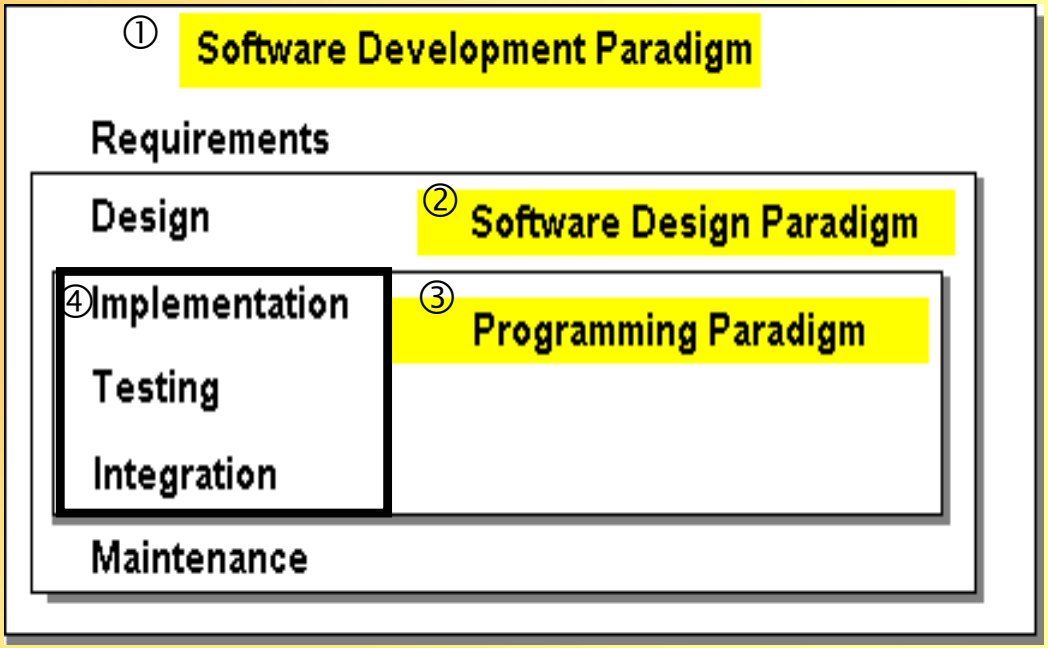


[1]Roger Pressman, **Software Engineering, A Practitioner's Approach, Fourth Edition**

What is the Object-Oriented Design Paradigm?

Evolution of Software – Paradigms

- There are several languages that fall into four main programming paradigms
  1. Functional (LISP, ML, Haskell)
  2. *Imperative/Structured* (Fortran, C, Ada)
  3. Logical (Prolog)
  4. *Object Oriented* (SmallTalk, Java, C++)
- Software Design Paradigms have methodologies associated with them
  - Imperative or Structure
    - Yourdon
    - SSDM
    - Jackson Structure Programming
  - Object-Oriented
    - UML
    - Shlaer-Mellor



Implementation Language → Programming Paradigm → Software Design Paradigm → Software Development Paradigm & Associated Activities (Implementation, Testing, Integration)

What is the Object-Oriented Design Paradigm?

Evolution of Software – Paradigms, Examples

- Jim will develop a payroll system
- He selects an implementation language – FORTRAN
  - STRUCTURED Programming Paradigm
    - STRUCTURED ANALYSIS Software Design Paradigm
      - WATERFALL Software Development Paradigm

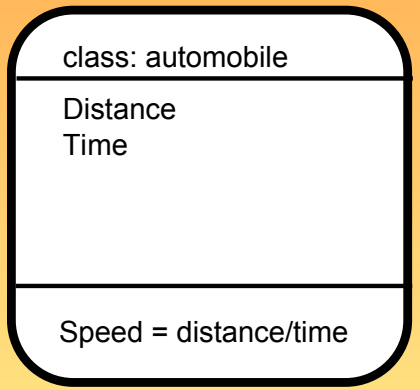
- Kim will also develop a payroll system
- She selects an implementation language – JAVA
  - OBJECT-ORIENTED Programming Paradigm
    - OBJECT-ORIENTED ANALYSIS Software Design Paradigm
      - AGILE Software Development Paradigm

What is the Object-Oriented Design Paradigm?

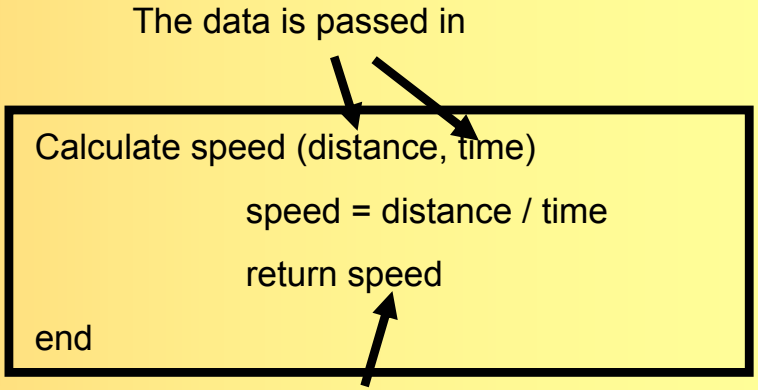
Object-Oriented vs. Structured

- Forces programmers to think in terms of *objects* rather than *procedures*

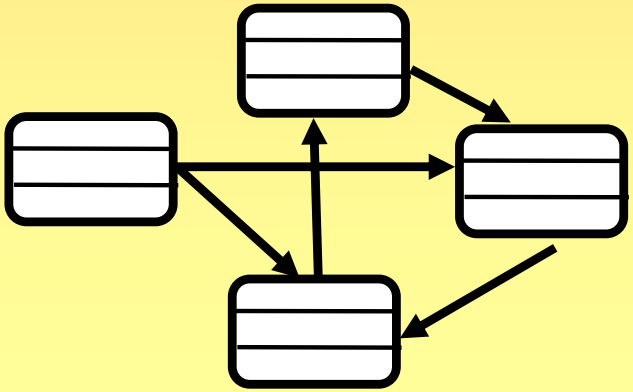
The object contains both data and instructions



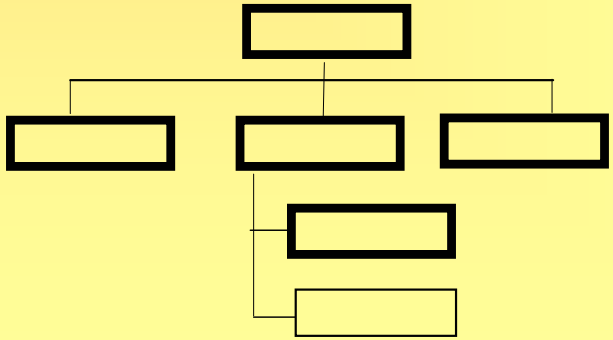
The object contains only instructions



OO is a network of structures



Structured programming is a hierarchy of structures



What is the Object-Oriented Design Paradigm?

Measuring Object-Oriented vs. Structured Code

- Traditional software size measures are based on data and procedure model of structured analysis
- These include:
  - Lines of code
    - Physical entity
    - Intuitive metric

```
for (i=0; i<100; ++i){    printf("hello");}
 /* Now how many lines of code is this? */
```

  - Function Points
    - Effective; given estimation artifacts can be determined in requirements definition

(a) Data Functions -> Internal Logical Files
(b) Data Functions -> External Interface Files
(c) Transaction Functions -> External Inputs
(d) Transaction Functions -> External Outputs
(e) Transaction Functions -> External Inquiries

Traditional Size Measures Work Well with Structured Design…

What is the Object-Oriented Design Paradigm?

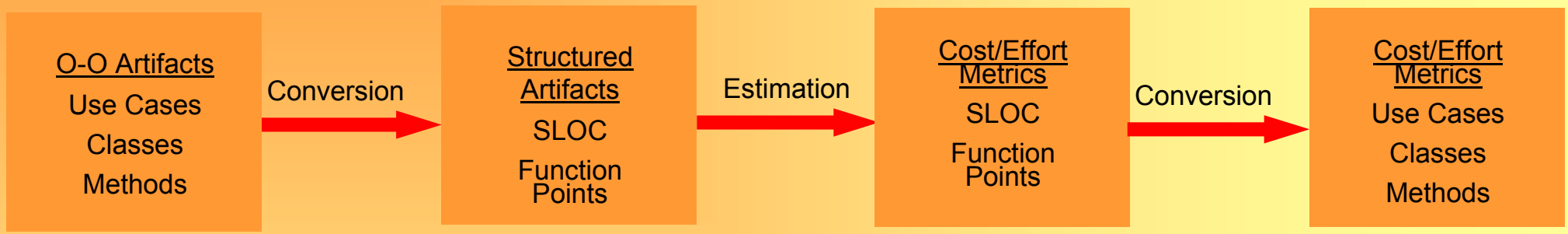
Measuring Object-Oriented vs. Structured Code

- Traditional software size measures do not capture the following aspects of object-oriented design
  - Inheritance – Indicator of reuse
  - Polymorphism – Another indicator of reuse
  - Modularity – Indicator of integration and interfaces
  - Encapsulation – Indicator of code stability, given change - volatility

For object-oriented software design, it's best to use a object-oriented estimation technique

Why Estimate in Native Format?

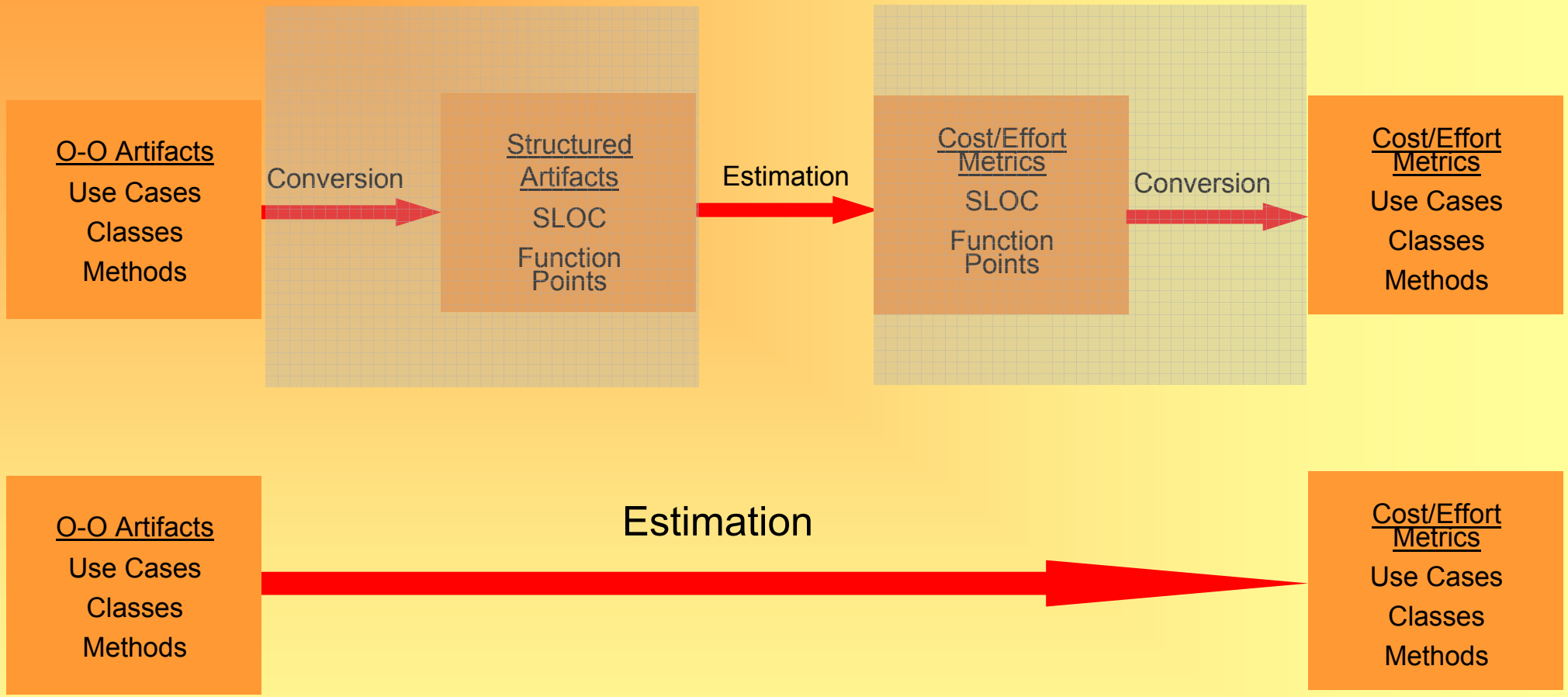
Today's Typical Estimation for O-O Design



- Determine size from artifacts
- Convert software size to SLOC or FPs
- Run estimate to get cost and productivity metrics in SLOC or FPs
- Convert measures back to object-oriented artifacts of origin to communicate meaningful metrics to developers

Why Estimate in Native Format?

Estimation Using O-O Methods



- Determine size from artifacts
- Run estimate
- Cost and productivity metrics expressed in terms that are same as input artifacts

Why Estimate in Native Format?

- Fundamentally, different paradigms require different estimating methods

- A paradigm-specific estimate is easier to calibrate, thus producing more accurate estimates
  - Post-development metrics are the same as the estimating metrics
  - Post development metrics can be collected in an automatic fashion

Why Estimate in Native Format?

- Using an Object Paradigm allows for integration with the developers' environment
  - Common Language between developers and estimator
  - Utilizes development artifacts
  - Easily updated as artifacts change
  - No need to convert artifacts into "estimation-only" terms

In O-O design, no need for an intermediate metric such as SLOC or FP

Object-Oriented Estimation Methods

Methodologies	Tools
Predictive Object Points (POPs) – driven by 4 metrics: TLC - # of top-level classes; WMC – weighted methods per class; DIT – average depth in inheritance tree; NOC – average # of children; and **Use Cases**	PRICE Systems' True S
ObjectMetrix – uses project scope (scope elements) as defined in Unified Modeling Language (UML): use cases, classes, subsystems components, and interfaces, qualified by size, complexity, reuse and genericity	TASSC: Estimator
Use Case Points – based on use case diagrams (use cases and actors);	Duvessa's Estimate Easy Use Case (EEUC)
Application Points (COCOMO II Level 1) - utilizes object points which counts screens, reports, and 3GL modules, with weights based on complexity	Duvessa's Estimate Easy Use Case (EEUC)
Model-Based Sizing –combination of OO metrics: number and complexity of use cases, objects, classes	QSM's SLIM Estimate

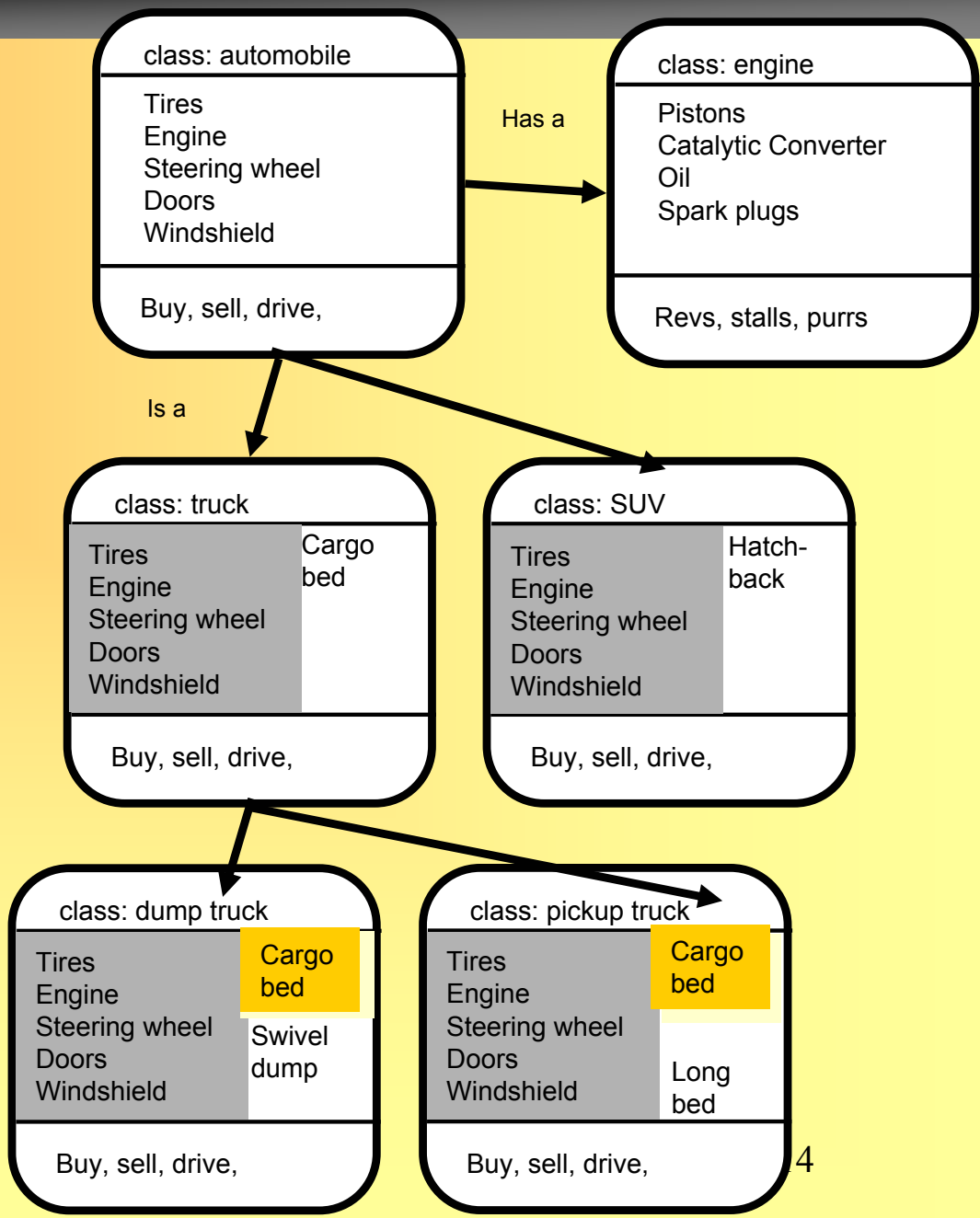
Object-Oriented Estimation Methods

Specifically, what does "object-oriented" mean?

ops CONSULTING

- A class describes an object
  - A class has attributes and behavior
  - Classes can have children
  - Classes can have relationships between other classes
- As an example, consider a truck
  - Part of a larger *class* – automobile
    - Has a set of generic attributes associated with every other object in the class of automobile
    - Each object within class has same attributes
  - Once class is defined, attributes can be *reused* when new instances of classes are created

The child class inherits all attributes of the parent class



Object-Oriented Estimation Methods

Predictive Object Points

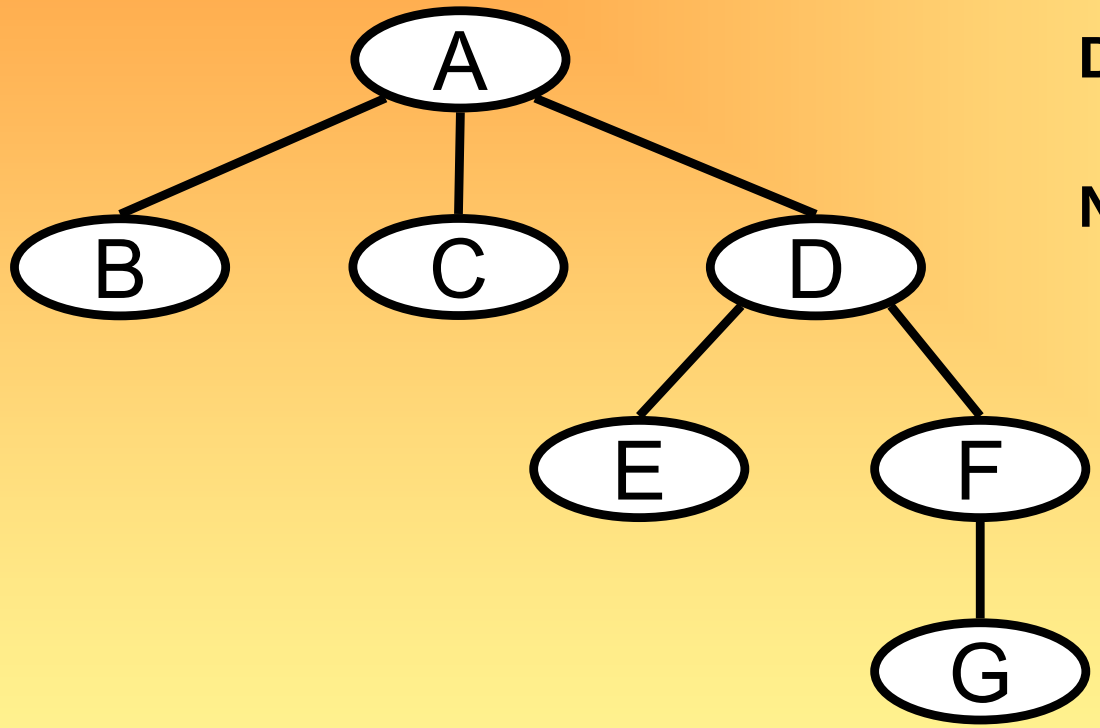
- Cost Drivers
  - **Weighted Methods per Class (WMC)**
  - Number of Top Level Classes (TLC)
  - Average Depth of Inheritance Tree (DIT)
  - Average Number of Children per Base Class (NOC)

The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class

The depth of a class within a hierarchy and the number of children for a class are predictors of the potential for reuse of inherited methods

Object-Oriented Estimation Methods

Predictive Object Points



TLC = 1

DIT = [(1x0)+(3x1)+(2x3)+[1x4)]/7

NOC = [(1x3)+(1x2)+(1x1)]/3

$$\textbf{WMC} = \frac{\sum \text{Methods} * \text{Weight}}{\text{Classes}}$$

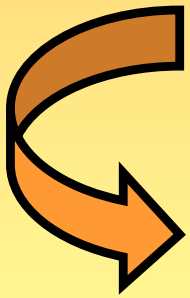
TLC = Top Level Classes
DIT = Depth In Tree
NOC = Number of Children
WMC = Weighted Methods Per Class

Object-Oriented Estimation Methods

Predictive Object Points

WMC WEIGHT

- Number of messages responded to
- Number of properties affected
- Method Type
  - Constructors/Destructors
  - Selectors
  - Modifiers
  - Iterators



POPs[1] = WMC *TLC*(1+(DIT*(1+NOC)))

Compute Core Cost/Schedule

- Production rate
- Effort rating
- Size characteristics
- Reuse

[2]Minkiewicz - June 6, 2000 United States Patent 6,073,107

Predictive Object Points

- Advantages
  - Primary cost driver, WMC, relates to behavior, a metric that has meaning to non-software, non-object individuals
  - Cost drivers are known by completion of preliminary design, facilitating a credible estimate later in the software lifecycle (WMC)
  - Patented process with relatively easy to implement counting methods
- Disadvantages
  - Not confirmed by exhaustive coverage of types of software, or implementation techniques
  - Cost drivers are not known until completion of preliminary design
  - Requires availability and application historical data rates

Object-Oriented Estimation Methods

ObjectMetrix

- ObjectMetrix is a technique for estimating and forecasting duration, resource requirements, and cost of object-oriented and component-based software development projects

- Cost-Drivers (Built-in Scope Elements)
  - Use Cases
  - Subsystems
  - Components
  - Interfaces
  - Classes
  - Web Pages
  - Scripts

- User-defined classifications and associated metrics can be added

Object-Oriented Estimation Methods

ObjectMetrix

- Model Scope Elements
  - Concept metric – represents a prediction of the effort to analyze, design, and build scope elements from a high-level analysis specification
  - Concrete metric – represents the construction effort to design and build from a low-level design specification
    - Concept metrics are larger than concrete metrics to reflect the earlier stage in the lifecycle and the likely scope population increase during analysis and design
  - Discovery metric – represents the effort required to replace a high-level analysis specification with a low-level design specification

ObjectMetrix provides built-in metric data, but organizations can populate with own historical data to produce increased accuracy

Object-Oriented Estimation Methods

- Scope Qualifiers
  - Size – a function of the quantity of work to be done or the scale of the task to be undertaken
  - Complexity – an indication of a high degree of diversity, numerous inter-relationships, significant algorithmic content and/or many decision points
  - Reuse – an indication of extensive use of pre-existing software elements from COTS class library or existing software infrastructure
  - Genericity – an indication that software element is required to be very general purpose, well-documented, highly reliable, and efficient

Object-Oriented Estimation Methods

ObjectMetrix

- Advantages
  - Flexible cost driver selection supports estimation early in the software lifecycle, with refinement throughout
  - User-defined cost drivers
- Disadvantages
  - Proprietary process, internals not well known

Use Case Points

- Estimates effort by analyzing complexity of actors and use cases
- Assigns weights and relevance to technical and environmental factors
- Originally developed by Gustav Karner
- Influenced by function point method
- Sought to take advantage of the use case models increasingly employed to capture and describe use case requirements of a software system

Use Case Points

- Categorize the actors as simple, average, or complex
- Calculate the total unadjusted actor weight (UAW)
- Categorize use cases as simple, average, or complex
- Calculate unadjusted use case weights (UUCW)
- Add UAW to UUCW to get unadjusted use case points (UUPC)
- Adjust use case points based on values assigned to number of technical and environmental factors (between 0 and 5)
  - Technical Complexity Factor: TCF = 0.6 + (0.1 * TFactor)
  - Environmental Factor: EF = 1.4 + (-0.03 * EFactor)
  - Adjusted use case points: UCP – UUCP & TCF * EF
- UCP is multiplied by a historically collected figure representing productivity to arrive at a project effort estimate

Use Case Points

- Advantages
  - Provides estimate at requirements phase of development cycle
  - Single cost driver input (Use Case)
  - Influenced by function point method
- Disadvantages
  - Provides estimate at requirements phase of development cycle
    - Lack accuracy of later phase models
  - Requires availability and application historical data rates

Application Points (COCOMO II Level 1)

- Developed by consortium of organizations led by Barry Boehm (USC)
- Based on Banker, Kauffman, and Kumar's Object Point Counting
- Developed for estimation of projects developed using Applications Composition Methods in Integrated Computer-Aided Software Engineering (ICASE)
- Cost Drivers
  - Numbers of Screens
  - Reports
  - Third-Generation (3GL) Components

Object-Oriented Estimation Methods

Application Points (COCOMO II Level 1)

- Count the number of screens, reports, and 3GL components
- Classify each screen, report, and 3GL module based on count of views and sections

Screens				Reports			
	Number and source of data tables				Number of source of data tables		
Number of views contained	Total < 4: (< 2 srvr < 3 clnt)	Total < 8: (2/3 srvr 3-5 clnt)	Total 8+: (> 3 srvr > 5 clnt)	Number of sections contained	Total < 4: (< 2 srvr < 3 clnt)	Total < 8: (2/3 srvr 3-5 clnt)	Total 8+: (> 3 srvr> 5 clnt)
< 3	Simple	Simple	Medium	0 or 1	Simple	Simple	Medium
3 to 7	Simple	Medium	Difficult	2 or 3	Simple	Medium	Difficult
> 8	Medium	Difficult	Difficult	4+	Medium	Difficult	Difficult

(Srvr : number of server data tables used in conjunction with screens and reports.)
(Clnt : number of client data tables used in conjunction with screens and reports.)

Application Points (COCOMO II Level 1)

- Weigh the classified components according to complexity. The weights reflect the effort required to implement an instance of that complexity level.

Object Type	Complexity - Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	-	-	10

- Calculate the Application Points. Multiply the number of screens, reports, and 3GL modules by the complexity-weight and sum those numbers to obtain the Application Point count.

Application Points (COCOMO II Level 1)

- Estimate the percentage of reuse and calculate the New Application Points to be developed:

$$NAP = (\text{Application Points}) * (100 - \%\ \text{reuse})/100$$

- Select appropriate productivity rate

Developer's experience and capability	Very Low	Low	Nominal	High	Very High
ICASE maturity and capability	Very Low	Low	Nominal	High	Very High
PROD (NAP/month)	4	7	13	25	50

- Compute the estimated person-months (PM) as:

$$PM = NAP/PROD$$

Application Points (COCOMO II Level 1)

- Advantages
  - Application Points and Function Point produced comparably accurate results
  - Study Managers considered Application Points easier to use than Function Points
  - Cost drivers known early in design
  - Model contains built-in productivity values
- Disadvantages
  - Model contains built-in productivity values
    - Users often neglect to calibrate
  - Restricted to I-CASE type applications

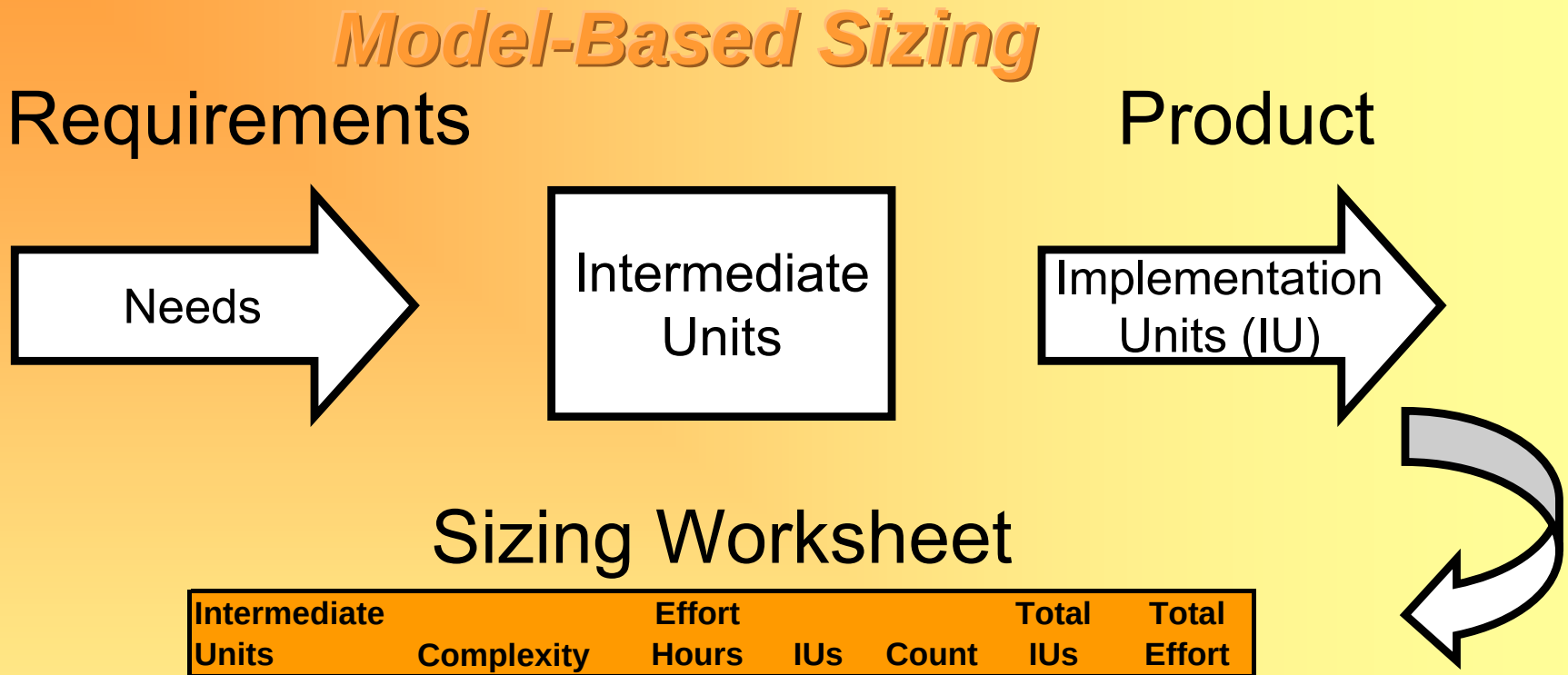
Model-Based Sizing

- Model-based sizing is an estimation technique that determines the size of software elements through decomposition to low-level software implementation units (IU)[4]
- Cost Drivers
  - Intermediate Units (translated requirements)
    - Forms
    - Reports
    - Table Changes
    - Script Changes
    - SQL Changes
  - Implementation Units (lowest level of programming construct that software developer performs)

[3]A Method for Improving Developers' Software Size Estimates Putnum, Putnum, and Beckett 2005

Object-Oriented Estimation Methods

Model-Based Sizing

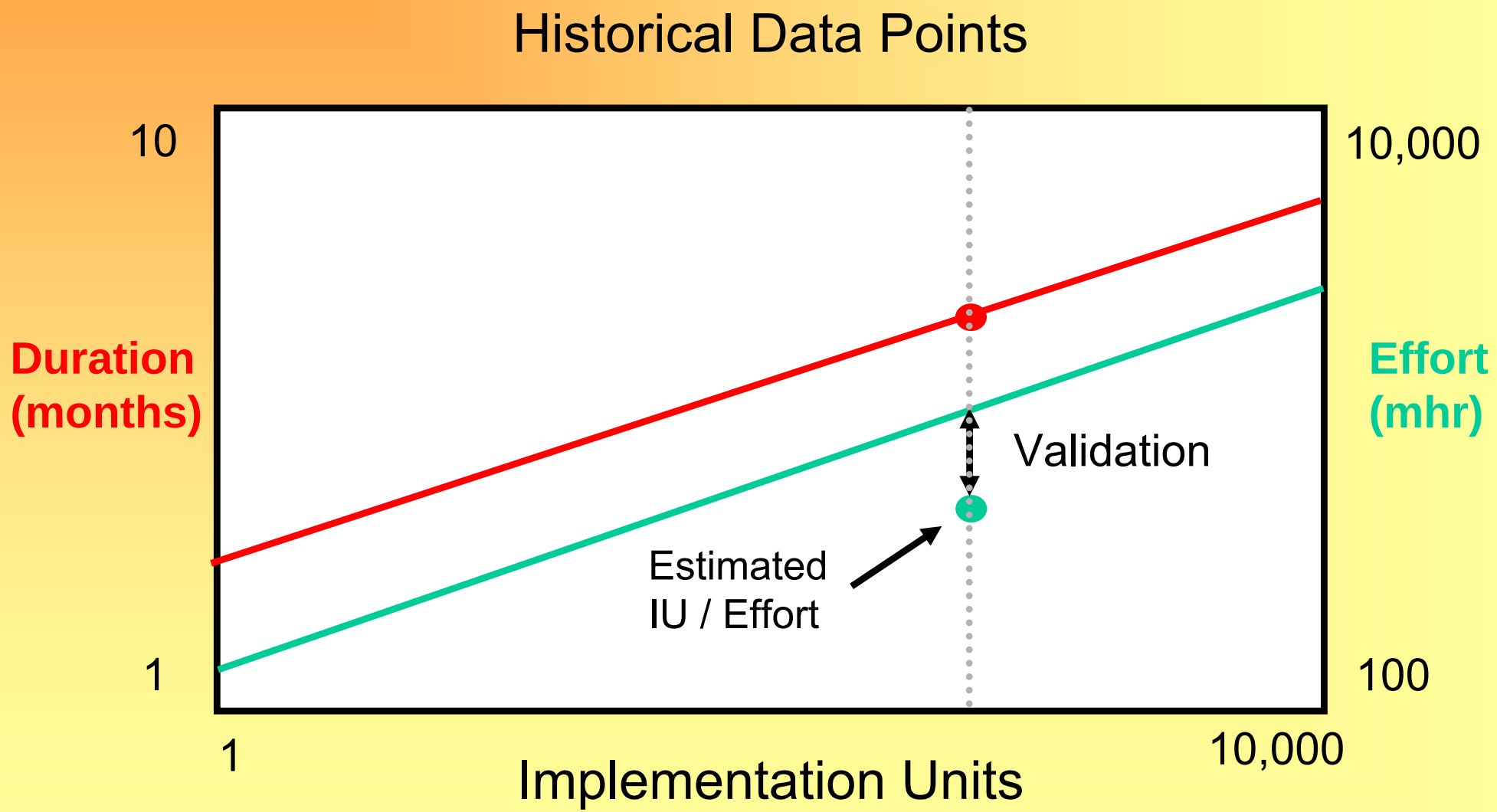


Sizing Worksheet

Intermediate Units	Complexity	Effort Hours	IUs	Count	Total IUs	Total Effort
Forms	Simple	8	72		0	0
Forms	Average	15	180	5	900	75
Forms	Complex	30	420	2	840	60
New Report	Simple	13	150	1	150	13
New Report	Average	32	300		0	0
New Report	Complex	42	450	7	3,150	294
Table Changes	Simple	10	90	6	540	60
Table Changes	Average	24	250	8	2,000	192
Table Changes	Complex	31	320		0	0
SQL Changes	Simple	5	60	10	600	50
SQL Changes	Average	13	140		0	0
SQL Changes	Complex	20	220	1	220	20
Total					8,400	764

Object-Oriented Estimation Methods

Model-Based Sizing



Model-Based Sizing

- Advantages
  - Improved communication between developers and estimators
  - Involves developers in estimating process
  - Applicable to many different development paradigms, not just O-O
  - Easy to implement with or without extensive tooling
- Disadvantages
  - Requires developers in estimating process
  - IUs not known until detailed factoring of model
    - lowest level of programming construct
  - Requires historical data points to validate/compute IU associated effort and duration

How Does This Apply?

Your organization's estimation and process environment

- Begin collecting object-oriented artifacts
  - Incorporate inquiry of design methodology
    - What high-level and low-level strategies will be used to allocate requirements for each configuration item to design entities? (Design entities such as objects, classes, modules, and CSCs)
    - Activities can include the following
      - Creating/Maintaining SDRs
      - Analysis of preliminary software design
      - Derive and map out high-level (top) software design specifications
      - Devise and map out low-level (detail) software design specifications
      - Analysis of preliminary interface design specifications
      - Define and describe interface design specifications
      - Generate input into software test planning
      - Formalize test requirements for design entities

How Does This Apply?

Your organization's estimation and process environment

- Begin collecting these metrics, continued
  - Describe primary approach used to estimate size of project or its components
    - Sizing methodology (such as one of the methods described)
    - Unit of measure (such as use cases, object points, etc)
  - Provide sizing metrics
    - Number of use cases
    - Number of classes
    - Number of web pages
    - Number of interfaces

How Does This Apply?

Your organization's estimation and process environment

- Consider software estimation tools
  - Evaluate and experiment with these methods on your current tool
  - Check integration between software estimation tools, cost models, and software development tools
    - Some O-O software estimation tools import CASE files
    - Export software PMP line item into your organization's current cost estimation model
- Get familiar with object-oriented terms and techniques
  - See references
  - Consult with software engineers to understand O-O design
- Time permitting, estimate past engagements using object-oriented estimating methods
  - Automated import of artifacts from development tools especially helpful with this
  - Compare results with prior estimates and/or actual results

Questions/Comments

References

ops CONSULTING

[1] **Software Engineering, A Practitioner's Approach, Fourth Edition**
Roger Pressman
Management Science, Vol. 44, No. 2 (Feb., 1998), pp. 203-218

[2] Minkiewicz - June 6, 2000 United States Patent 6,073,107

[3] A Method for Improving Developers' Software Size Estimates Putnum, Putnum, and Beckett 2005

How to Contact OPS

ops CONSULTING

Leah Upshaw, Senior Associate
leahupshaw@opsconsulting.com
202-744-9434

OPS Consulting, L.L.C.
2017 Martins Grant Court
Crownsville, MD 21032
http://www.opsconsulting.com/