Project 3 Scope and Evaluations

- **Scope proposed by teams in Project 3 is OK**

(notes will be available in the observations that accompany your Project 3 grade)

- **Submit your Project 3 evaluations via Blackboard**

(if you haven’t already done so)

Software Quality and Metrics

Prof. Alex Bardas

What is Software Quality?

User Satisfaction = **compliant product** + **good quality**

+ **delivery within budget and schedule**

- Software quality can be described from different points of views
  - Transcendental view
  - User's view: requirements
  - Manufacturer's view: specifications
  - Product view: functions
  - Value-based view

What is Software Quality?

- **Quality of design**
  - How the *design* of the system meets the *specifications* in the *requirements model*
- **Quality of conformance**
  - How the *implementation* follows the design so that the system meets the requirements

Software with a high technical quality can evolve with low cost and risk to keep meeting functional and non-functional requirements.

What are the “implicit requirements”

- **Software quality factors**
  - A ***non-functional requirement*** for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program.

- The factors are NOT binary
  - Evaluate the *degree* to which the software demonstrate a factor

Quality Factors

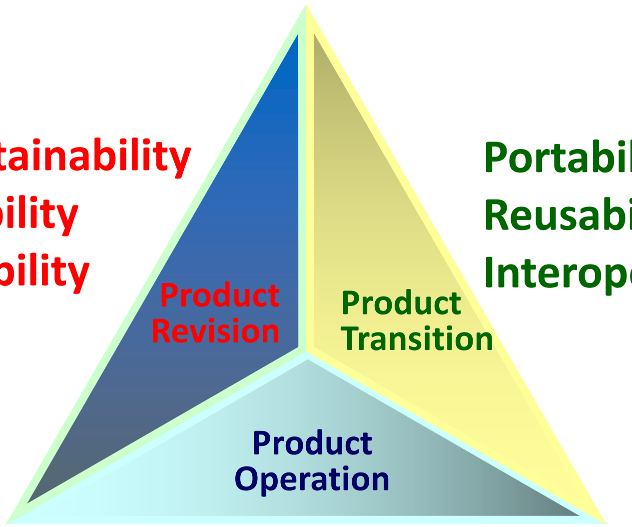
- Dimensions of quality [Garvin'87]
  - Performance
  - Feature
  - Reliability
  - Conformance
  - Durability
  - Serviceability
  - Aesthetics
  - Perception

soft, subjective, but solid indication

Quality factors [McCall'78]

Maintainability
Flexibility
Testability

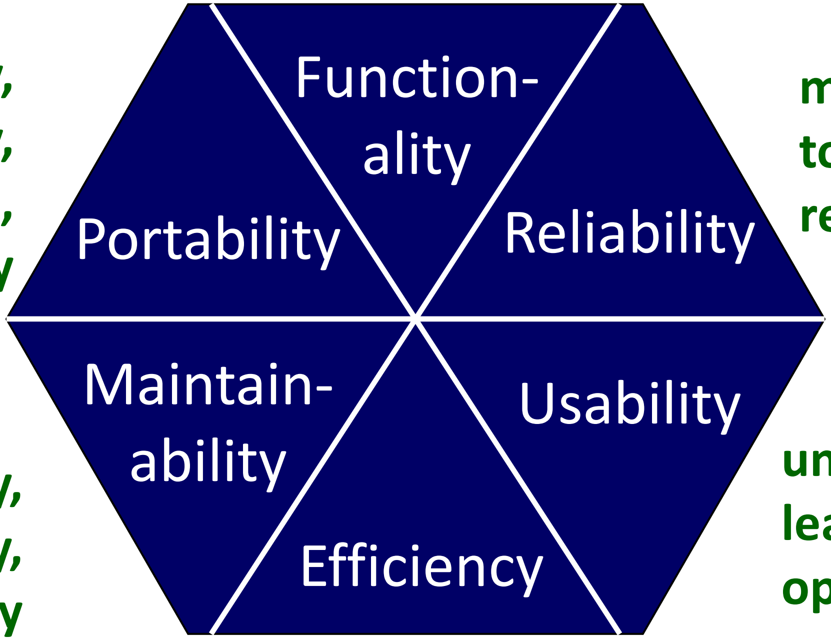
Portability
Reusability
Interoperability



Correctness
Reliability
Efficiency

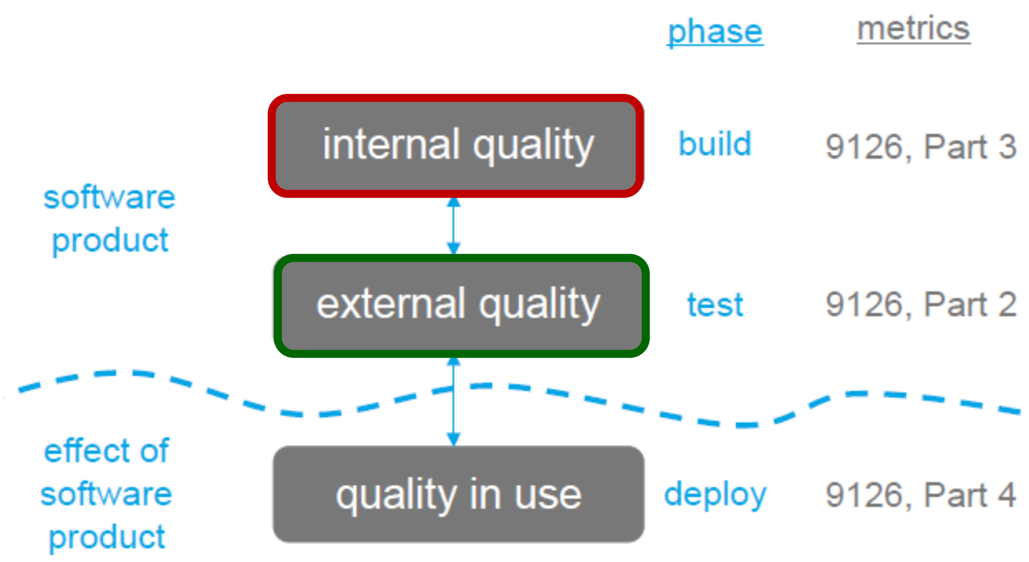
Usability
Integrity

ISO 9126 Quality Factors



ISO 9126 Quality Factors

- Leads to software metrics for quantitative assessments
  - e.g., 16 external quality measures and 9 internal quality measures for maintainability



- "activity recording" (analysability)
  - ratio between actual and required # of data items
- "change impact" (changeability)
  - # of modifications and problems introduced
- "re-test efficiency" (testability)
  - time spent to correct a deficiency
- "change implementation elapse time" (changeability)
  - time between diagnosis and correction

Quality Dilemma

- “Good Enough” software
  - Deliver software with known bugs and incomplete features
- Cost of quality
  - Costs too much time and money to get to the expected quality level
  - **Prevention costs**: quality planning, adding formal technical activities, test planning, training
  - **Appraisal costs**: technical review, data collection & evaluation, testing
  - **Failure costs**: internal failure costs (repair, rework, failure analysis) and external failure costs (help line support, complaint resolution, product return and replacement, warranty)

Cost of Quality

- **Prevention costs**
  - Quality planning and coordination
  - Adding formal technical reviews, planning test
  - Training
- **Appraisal costs**
  - Process inspection, technical review, data collection, etc.
  - Testing and debugging
- **Failure costs**
  - ***Internal failure costs:*** rework, repair, failure mode analysis before shipment
  - ***External failure costs:*** complaint resolution, product return and replacement, help line support, warranty work after delivery

Cost of Fixing Critical Defects

[Cigital, “Case Study: Finding Defects Earlier Yields Enormous Savings”]

Cost of Fixing Vulnerabilities **EARLY**			
Stage	**Critical Bugs Identified**	**Cost of Fixing 1 Bug**	**Cost of Fixing All Bugs**
Requirements		$139	
Design		$455	
Coding	200	$977	$195,400
Testing		$7,136	
Maintenance		$14,102	
Total	**200**		**$195,400**

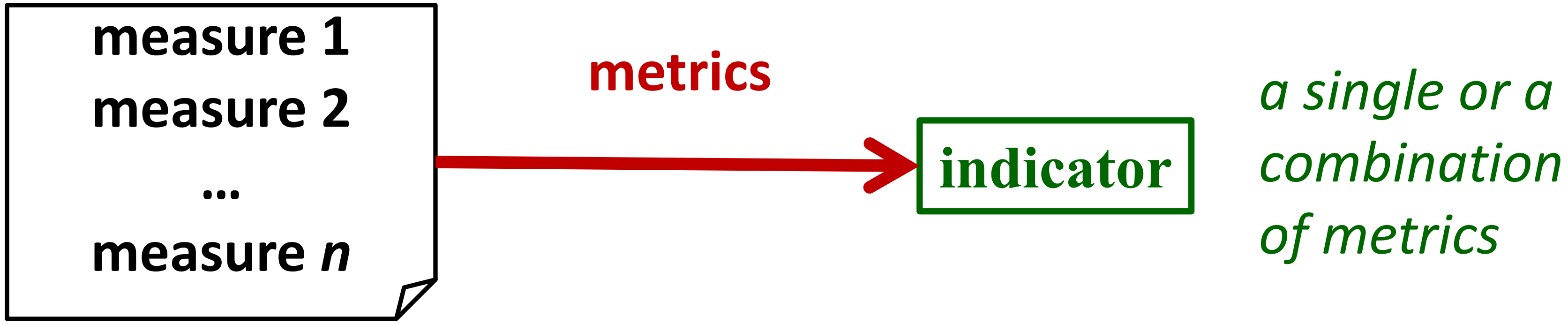
Cost of Fixing Vulnerabilities **LATER**			
Stage	**Critical Bugs Identified**	**Cost of Fixing 1 Bug**	**Cost of Fixing All Bugs**
Requirement		$139	
Design		$455	
Coding		$977	
Testing	50	$7,136	$356,800
Maintenance	150	$14,102	$2,115,300
Total	**200**		**$2,472,100**

Identifying the critical bugs earlier in the lifecycle reduced costs by $2.3M

Software Metrics

- **Measures:** quantitative indication of product attributes
  - About its extent, amount, dimension, capacity, or size

    e.g., # of errors in a software unit
- **Metrics:** quantitative measure of the degree to which a system possess a given attribute
  - Relates individual measures e.g., avg. # of errors per unit test



Why Measure Software?

- To improve software quality
- To estimate development time and budget
- Measurement guidelines
  - Data collection and analysis should be **automated**
  - Apply valid **statistical techniques** to establish relationships between internal attributes and external quality characteristics
  - For each metric, establish interpretative guidelines and recommendations

Use Case-Oriented Metrics

- **Use Case Points (UCP)**
  - A size and effort metric
  - Pros: defined earlier, user visible, language independent
  - Cons: no standard size, subjective estimation
- A function of:
  - Size of functional features (“unadjusted” UCPs)
  - Non-functional factors
    - TCF: technical complexity factors
  - Environmental complexity factors (ECF)

Use Case-Oriented Metrics

- **Actor Classification and Weights**

Actor type	Description of how to recognize the actor type	Weight
Simple	The actor is another system which interacts with our system through a defined API.	1
Average	The actor is a person interacting through a text-based user interface, or another system interacting through a protocol, such as a network communication protocol.	2
Complex	The actor is a person interacting via a graphical user interface.	3

Use Case-Oriented Metrics

Example: Safe Home Access (SHA)

- **Unadjusted Actor Weight (UAW)**

UAW(SHA) = $5 \times$ Simple $+ 2 \times$ Average $+ 1 \times$ Complex $= 5\times1 + 2\times2 + 1\times3 = 12$

Actor name	Description of relevant characteristics	Complexity	Weight
Landlord	Landlord is interacting with the system via a graphical user interface (when managing users on the central computer).	Complex	3
Tenant	Tenant is interacting through a text-based user interface (assuming that identification is through a keypad; for biometrics based identification methods Tenant would be a complex actor).	Average	2
LockDevice	LockDevice is another system which interacts with our system through a defined API.	Simple	1
LightSwitch	Same as LockDevice.	Simple	1
AlarmBell	Same as LockDevice.	Simple	1
Database	Database is another system interacting through a protocol.	Average	2
Timer	Same as LockDevice.	Simple	1
Police	Our system just sends a text notification to Police.	Simple	1

Use Case-Oriented Metrics

- **Unadjusted Use Case Weights (UUCW)**
  - Determine based on the number of transactions

Use case category	Description of how to recognize use case category	Weight
Simple	Simple user interface. Up to one participating actor (plus initiating actor). Number of steps for the success scenario: ≤ 3. If presently available, its domain model includes ≤ 3 concepts.	5
Average	Moderate interface design. Two or more participating actors. Number of steps for the success scenario: 4 to 7. If presently available, its domain model includes between 5 and 10 concepts.	10
Complex	Complex user interface or processing. Three or more participating actors. Number of steps for the success scenario: ≥ 7. If available, its domain model includes ≥ 10 concepts.	15

Use Case-Oriented Metrics

UUCW(SHA) = 1 × Simple + 5 × Average + 2 × Complex = 1×5 + 5×10 + 2×15 = 85

Use case	Description	Category	Weight
Unlock (UC-1)	Simple user interface. 5 steps for the main success scenario. 3 participating actors (LockDevice, LightSwitch, and Timer).	Average	10
Lock (UC-2)	Simple user interface. 2+3=5 steps for the all scenarios. 3 participating actors (LockDevice, LightSwitch, and Timer).	Average	10
ManageUsers (UC-3)	Complex user interface. More than 7 steps for the main success scenario (when counting UC-6 or UC-7). Two participating actors (Tenant, Database).	Complex	15
ViewAccessHistory (UC-4)	Complex user interface. 8 steps for the main success scenario. 2 participating actors (Database, Landlord).	Complex	15
AuthenticateUser (UC-5)	Simple user interface. 3+1=4 steps for all scenarios. 2 participating actors (AlarmBell, Police).	Average	10
AddUser (UC-6)	Complex user interface. 6 steps for the main success scenario (not counting UC-3). Two participating actors (Tenant, Database).	Average	10
RemoveUser (UC-7)	Complex user interface. 4 steps for the main success scenario (not counting UC-3). One participating actor (Database).	Average	10
Login (UC-8)	Simple user interface. 2 steps for the main success scenario. No participating actors.	Simple	5

Use Case-Oriented Metrics

- **Technical Complexity Factors (TCFs)**

Technical factor	Description	Weight
T1	Distributed system (running on multiple machines)	2
T2	Performance objectives (are response time and throughput performance critical?)	1
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install (are automated conversion and installation included in the system?)	0.5
T7	Easy to use (including operations such as backup, startup, and recovery)	0.5
T8	Portable	2
T9	Easy to change (to add new features or modify existing ones)	1
T10	Concurrent use (by multiple users)	1
T11	Special security features	1
T12	Provides direct access for third parties (the system will be used from multiple sites in different organizations)	1
T13	Special user training facilities are required	1

Use Case-Oriented Metrics

- **Technical Complexity Factors (TCFs)**
  - For each of the 13 factors, assign a perceived complexity factor (F_i), whose value is between 0 and 5
    - Technical factor is irrelevant (0), average (3), or influential (5)
  - Assume overall TCF impacts UCP from a range of 0.6 to 1.3
    - 0.6 if all perceived complexity are 0
    - 1.3 if all 5

So, the impact is modeled as:

TCF = Constant-1 + Constant-2 × Technical Total Factor = $C_1 + C_2 \sum_{i=1}^{13} W_i F_i$

Constant-1 (C_1) = 0.6
Constant-2 (C_2) = 0.01

Use Case-Oriented Metrics

Technical factor	Description	Weight	Perceived Complexity	Calculated Factor (Weight×Perceived Complexity)
T1	Distributed, Web-based system	2	3	2×3 = 6
T2	Users expect good performance but nothing exceptional	1	3	1×3 = 3
T3	End-user expects efficiency but there are no exceptional demands	1	3	1×3 = 3
T4	Internal processing is relatively simple	1	1	1×1 = 1
T5	No requirement for reusability	1	0	1×0 = 0
T6	Ease of install is moderately important (will probably be installed by technician)	0.5	3	0.5×3 = 1.5
T7	Ease of use is very important	0.5	5	0.5×5 = 2.5
T8	No portability concerns beyond a desire to keep database vendor options open	2	2	2×2 = 4
T9	Easy to change minimally required	1	1	1×1 = 1
T10	Concurrent use is required	1	4	1×4 = 4
T11	Security is a significant concern	1	5	1×5 = 5
T12	No direct access for third parties	1	0	1×0 = 0
T13	No unique training needs	1	0	1×0 = 0
			Technical Total Factor:	**31**

TCF = 0.6 + 0.01 * 31 = 0.91

Use Case-Oriented Metrics

- **Environmental Complexity Factors (ECFs)**
  - Team determines each factor's *perceived impact* of the factor based on its experiences
  - No impact (0), strong negative (1), average (3), or strong positive (5)

Environmental factor	Description	Weight
E1	Familiar with the development process (e.g., UML-based)	1.5
E2	Application problem experience	0.5
E3	Paradigm experience (e.g., object-oriented approach)	1
E4	Analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time worker	–1
E8	Difficult programming language	–1

Use Case-Oriented Metrics

- **Environmental Complexity Factors (ECFs)**
  - Larger ECF should have a greater impact on UCP
  - Suggest ECF has on the UCP equation from 0.425 to 1.4
    - 0.425 (Part-Time Workers and Difficult Programming Language = 0, all other values = 5)
    - 1.4 (perceived impact all 0)

ECF = Constant-1 $+$ Constant-2 $\times$ Environment Total Factor = $C_1 + C_2 \sum_{i=1}^{8} W_i F_i$

Constant-1 (C_1) = 1.4
Constant-2 (C_2) = -0.03

Use Case-Oriented Metrics

Environmental factor	Description	Weight	Perceived Impact	Calculated Factor (Weight× Perceived Impact)
E1	Beginner familiarity with the UML-based development	1.5	1	1.5×1 = 1.5
E2	Some familiarity with application problem	0.5	2	0.5×2 = 1
E3	Some knowledge of object-oriented approach	1	2	1×2 = 2
E4	Beginner lead analyst	0.5	1	0.5×1 = 0.5
E5	Highly motivated, but some team members occasionally slacking	1	4	1×4 = 4
E6	Stable requirements expected	2	5	2×5 = 5
E7	No part-time staff will be involved	–1	0	–1×0 = 0
E8	Programming language of average difficulty will be used	–1	3	–1×3 = –3
			Environmental Total Factor:	**11**

ECF = 1.4 - 0.03 * 11 = 1.07

Use Case-Oriented Metrics

- **Calculate the Use Case Points:** UCP = (UUCW + UAW) $\times$ TCF $\times$ ECF
  - UAW + UUCW = 12 + 85 = 97
  - TCF = 0.91
  - ECF = 1.07

UCP = 97 $\times$ 0.91 $\times$ 1.07 = 94.45 → 94 use case points

- *For this case study, the TCF and ECF reduced the* (UUCW + UAW) *by approximately 3 percent (94/97*100)*

Use Case-Oriented Metrics

- How to use UCP?
  - The UCP value by itself is not very useful
  - To estimate the effort (time) of the project, we need another factor
- Productivity Factor (*PF*)
  - *The ratio of development man-hours needed per use case point*

Total estimated number of hours = UCP * PF

  - Use statistics from past projects
  - If no historical data,
    - Count previous components to establish a baseline
    - Use a value between 15 and 30 based on team's past experience

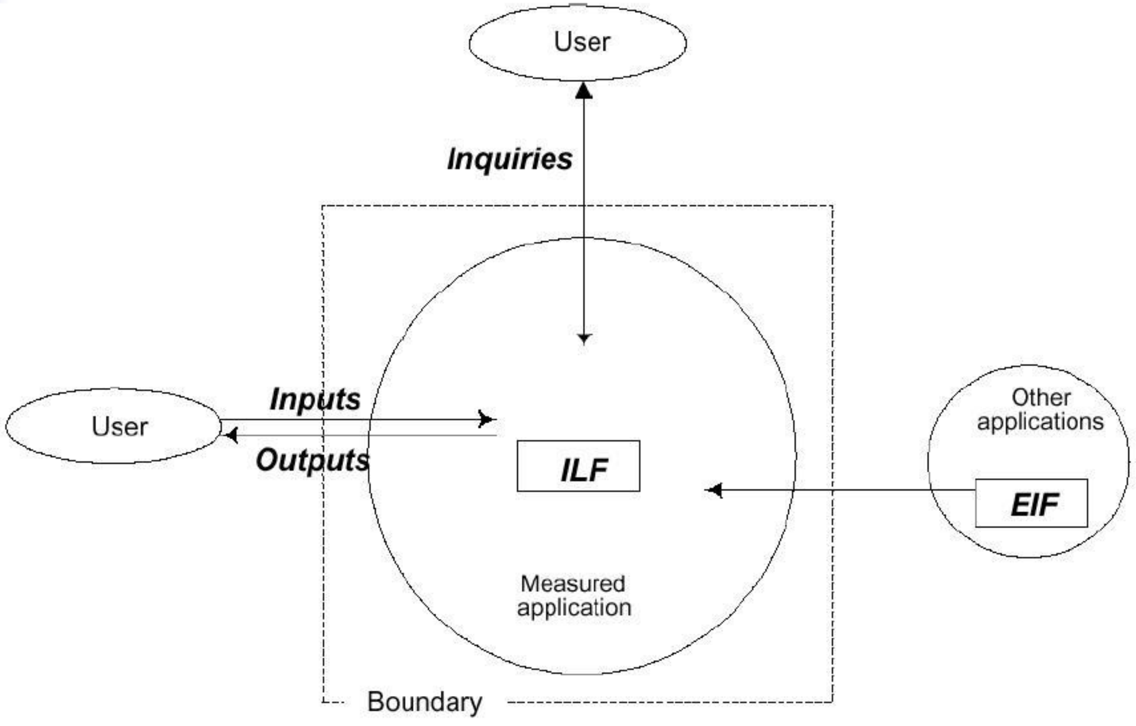
Function-Oriented Metrics

- **Function Points (FP)**
  - Most widely used
  - Measure functionality of a system from users' point of view
    - What users request and receive in return
  - Build an empirical relationship between direct measures of software information domain and complexity assessments

  - Pros: Based on data known early, language independent
  - Cons: Still subjective

Function-Oriented Metrics

- **Function Points (FP)**
  - Information domain values:
    - # of internal logical files (ILF)
    - # of external inputs (EI)
      - Often updates ILF
    - # of external inquiries (EQ)
      - Inputs resulting in some response
    - # of external outputs (EO)
      - Screens, reports, or error messages
    - # of external interface files (EIF)



Function-Oriented Metrics

- Computing FP
  - Count # of FP in each information domain category
  - **Assign a weight factor to each category**
  - Calculate the Count Total=$\sum_{i=1}^{5} \#_i \times W_i$

Information Domain Value	Count		Weighting Factor Simple	Avg.	Complex		
External Inputs (EIs)	$\square$	X	3	4	6	=	$\square$
External Outputs (EOs)	$\square$	X	4	5	7	=	$\square$
External Inquiries (EQs)	$\square$	X	3	4	6	=	$\square$
Internal Logic Files (ILFs)	$\square$	X	7	10	15	=	$\square$
External Interface Files (EIFs)	$\square$	X	5	7	10	=	$\square$
Count Total						→	$\square$

Function-Oriented Metrics

- **Value Adjustment Factors (VAF)**
  - General system characteristics
  - 14 factors (F_i)
  - Assign weights from 0 (not important) to 5 (essential)

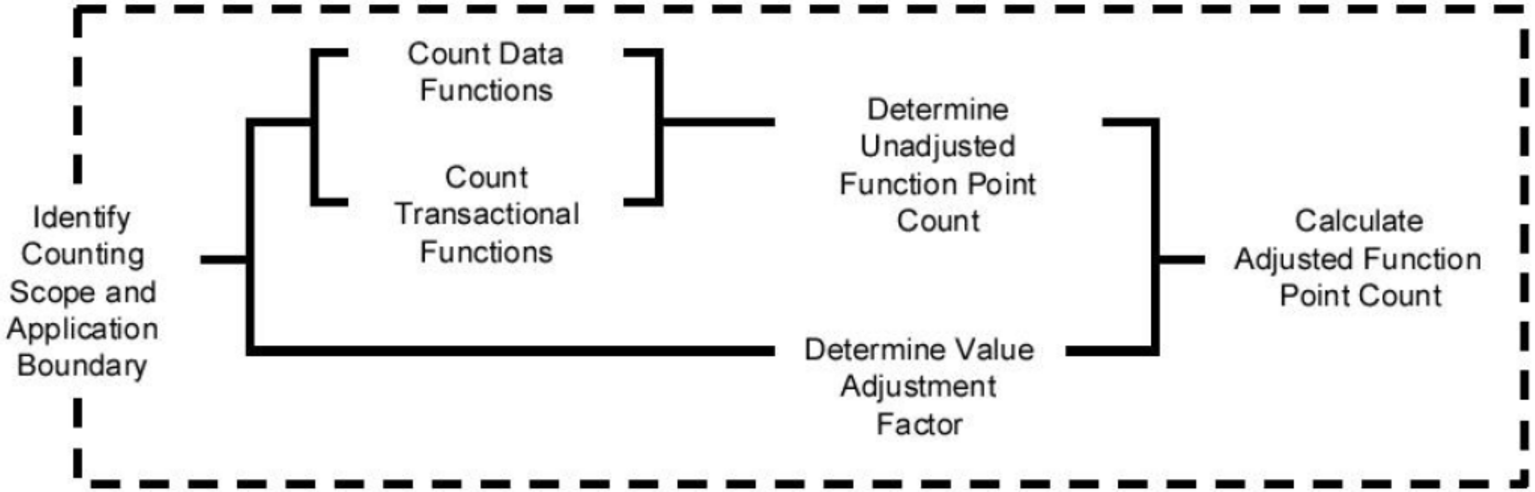
General System Characteristic		Brief Description
1	Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
2	Distributed data processing	How are distributed data and processing functions handled?
3	Performance	Did the user require response time or throughput?
4	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
5	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?
6	On-Line data entry	What percentage of the information is entered On-Line?
7	End-user efficiency	Was the application designed for end-user efficiency?
8	On-Line update	How many ILF's are updated by On-Line transaction?
9	Complex processing	Does the application have extensive logical or mathematical processing?
10	Reusability	Was the application developed to meet one or many user's needs?
11	Installation ease	How difficult is conversion and installation?
12	Operational ease	How effective and/or automated are start-up, back up, and recovery procedures?
13	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
14	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

Function-Oriented Metrics

- Scaling Factor (derived empirically): $C_1 = 0.65;\ C_2 = 0.01$

  FP = Count Total $\times[\boldsymbol{C_1}+\boldsymbol{C_2}\times\sum_{\boldsymbol{i=1}}^{\boldsymbol{14}}\boldsymbol{F_i}]$

- The entire process for counting FP:



Size-Oriented Metrics

- Size-Oriented Metrics
  - Normalize quality by lines of code (LOC)

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per LOC
- Page of documentation per KLOC
- Errors per person-month
- LOC per person-month
- $ per page of documentation

  - Widely used as a quality/productivity measure
    - In the 70s or 80s, IBM paid people per line-of-code

Size-Oriented Metrics

- LOC is dangerous (cons)
  - Penalize well-designed but short programs
  - Programming language-dependent

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**	-	-	-
PL/1	**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

Specification-Based Quality Metrics

- Assess the quality of requirement specifications
  - Specificity
  - Completeness
  - Correctness, understandability, verifiability, internal and external consistency, achievability, concision, traceability, modifiability, precision, and reusability
- Each can be quantified using one or more metrics

Specification-Based Quality Metrics

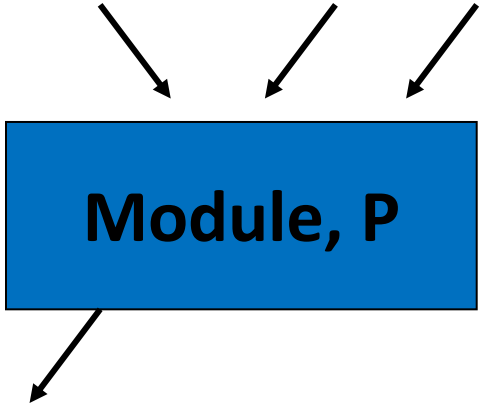
- Assume there are n_r requirements in a specification: $n_r = n_f + n_{nf}$
  - n_f and n_{nf} are the # of functional and non-functional requirements
- **Specificity** = $\frac{n_{ui}}{n_r}$
  - n_{ui} is the # of requirements for which all reviewers have identical interpretations
- **Completeness of functional requirements** = $\frac{n_u}{n_i \times n_s}$
  - n_u is the # of unique functional requirements
  - n_i is the # of inputs
  - n_s is the # of states specified
  - Measures the percentage of necessary functions that have been specified

Design Metrics

- Various metrics have been derived to measure the **complexity** of design
- **Architectural Design**
  - Consider architectural modules/components
  - Not consider the inner workings of each model/component

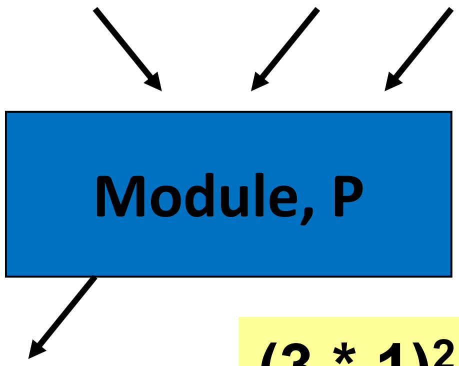
e.g., in a hierarchical architecture:

- fan-in: # of modules that invoke P
- fan-out: # of modules immediately subordinate P



Design Metrics

- Henry-Kafura (HK) metric
  - "Structural complexity"
  - Measure interactions between modules
  - $C_p=[f_{in}(P) \times f_{out}(P)]^2$
- Later modified to include internal complexity
  - $HC_p= C_{ip} \times [f_{in}(P) \times f_{out}(P)]^2$
  - C_{ip} is any code metric for internal complexity
    e.g., LOC as length(P)



$(3 * 1)^2 = 9$

Design Metrics

- A higher-level complexity measure
  - *Structural complexity*
    - $S(i) = (f_{out}(i))^2$
  - *Data complexity*
    - $D(i) = v(i) / [\ f_{out}(i) + 1\ ]$
    - $v(i)$ is the number of inputs and outputs passed to and from i
  - *System complexity*
    - $C(i) = S(i) + D(i)$

Design Metrics

- **Component-Level Metrics (coding metrics)**
- Cyclomatic Complexity: developed by Thomas McCabe (1974)
  - # of linearly independent paths through the code
  - **Measures the complexity of a program's conditional logic**
  - High V(G) leads to high error probability: should be less than 10
  - Count the number of decisions in the program
    - Cyclomatic complexity of graph G = #edges - #nodes + 2

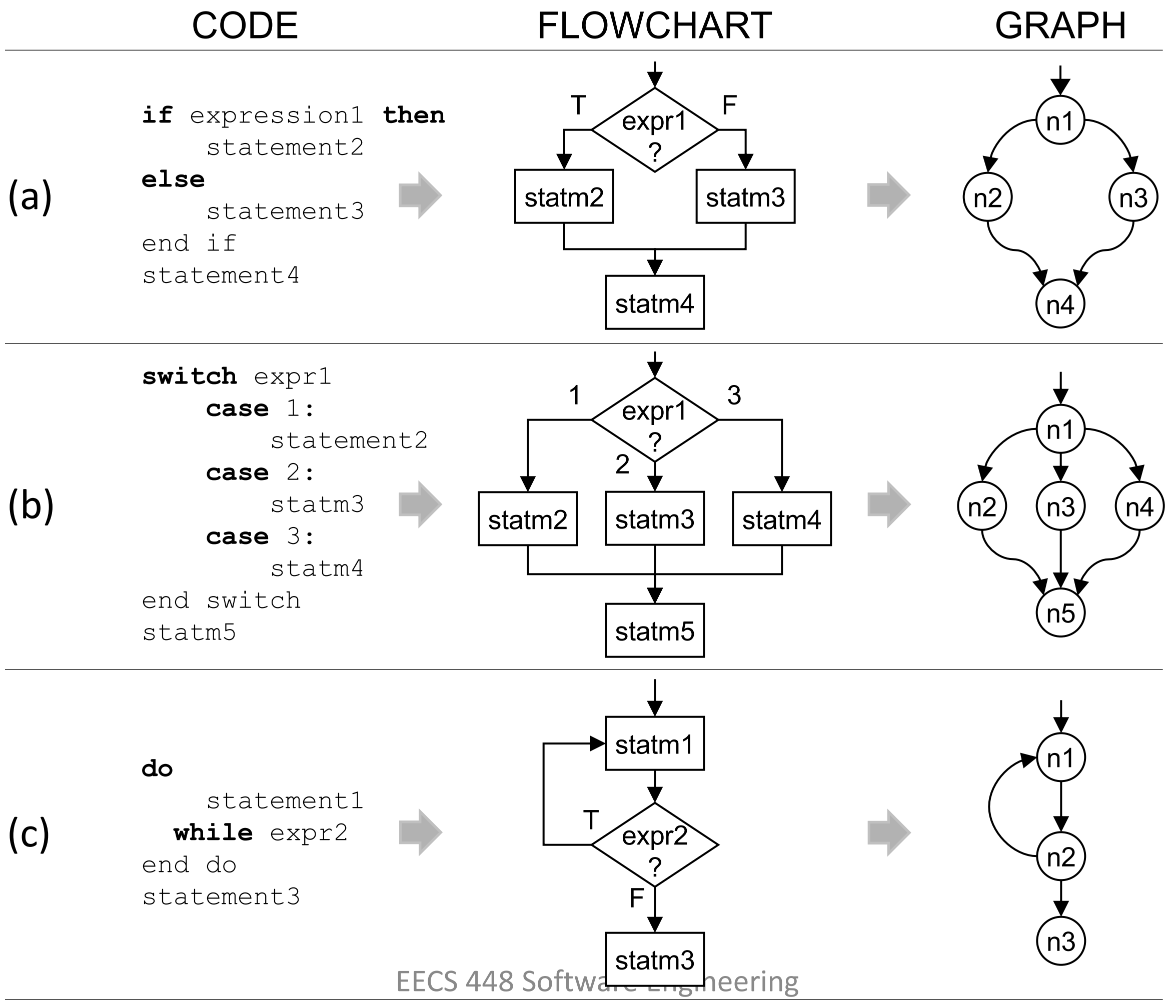
$$V(G) = e - n + 2$$

    - Cyclomatic complexity of graph G = #binary decisions + 1

$$V(G) = p + 1$$

Design Metrics

Convert code to graph



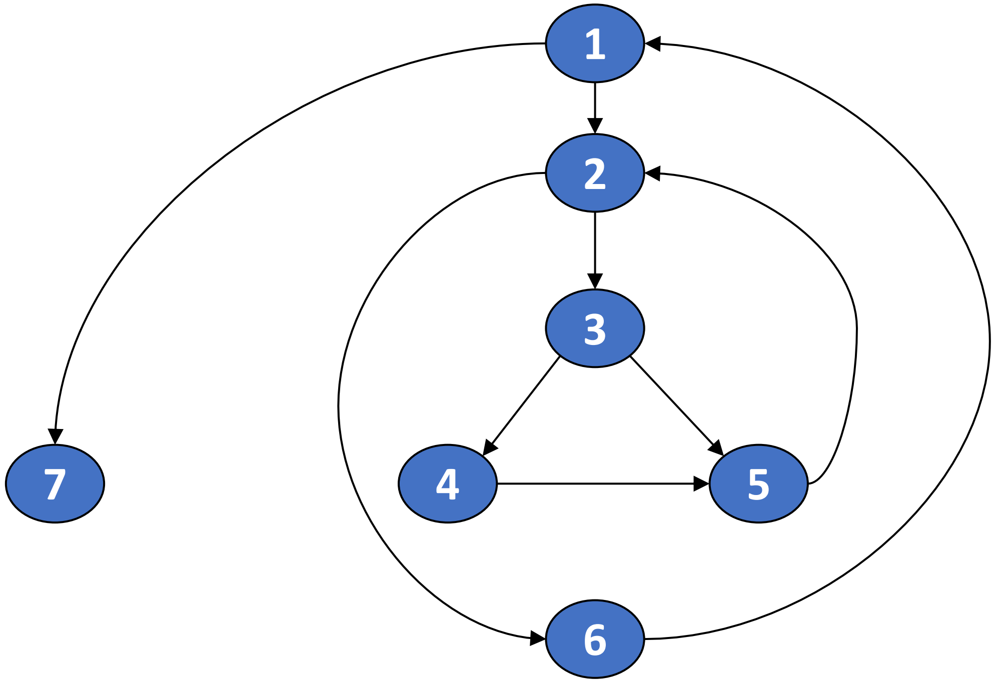
(a) 1 binary decisions

(b) 2 binary decisions

(c) 1 binary decisions

Design Metrics

- Cyclomatic Complexity example
  - V(G) = e - n + 2 = 9 – 7 + 2 = 4
  - V(G) = p + 1 = 3 + 1 = 4
  - Basis path set:
    - {1, 7}
    - {1, 2, 6, 1, 7}
    - {1, 2, 3, 4, 5, 2, 6, 1, 7}
    - {1, 2, 3, 5, 2, 6, 1, 7}



Design Metrics

- **Component-Level Metrics**
  - Measure module cohesion

    6 - Functional cohesion

    module performs a single well-defined function

    5 - Sequential cohesion

    >1 function, but they occur in an order prescribed by the specification

    4 - Communication cohesion

    >1 function, but on the same data (not a single data structure or class)

    3 - Procedural cohesion

    multiple functions that are procedurally related

    2 - Temporal cohesion

    >1 function, but must occur within the same time span (e.g., initialization)

    1 - Logical cohesion

    module performs a series of similar functions, e.g., Java class java.lang.Math

    0 - Coincidental cohesion

high cohesion

low cohesion

Design Metrics

- Cohesion measure depends on subjective human assessment
  - Most cohesion metrics focus on syntactic cohesion
  - **LCOM**: lack of cohesion in method
    - Count the # of pairs of methods that do not share class attributes
    - Consider a class C has
      - A set of methods: M_i, i=1…m
      - A set of attributes: A_j, j=1…a
    - LCOM(C)= $\frac{m-(\frac{1}{a}\sum_{1}^{a} A_j)}{m-1}$
    - LCOM is included in the Chidamber & Kemerer object-oriented metrics suite

Design Metrics

- **Component-Level Metrics**
  - Measure module coupling: # of input/output parameters, global variables, and modules called
  - Data and control flow coupling
    - d_i = # of input data parameters
    - c_i = # of input control parameters
    - d_o = # of output data parameters
    - c_o = # of output control parameters
  - Global coupling
    - g_d = # of global variables used as data
    - g_c = # of global variables used as control

Design Metrics

- **Component-Level Metrics**
  - Environmental coupling
    - w = # of modules called (fan in)
    - r = # of modules calling (fan out)

  - Coupling metric $m_c = \frac{k}{M}$
    - $M = d_i + (a \times c_i) + d_o + (b \times c_o) + g_d + (c \times g_c) + w + r$
    - k is a proportionality constant (k =1)
    - $a = b = c = 2$

Design Metrics

- **Metrics for User Interface Design**
  - Layout appropriateness (LA): consider layout entities
    - Absolute and relative position of layout entity
    - Frequency of using an entity
    - Transition cost from one entity to another

cost = sum[frequency of transition(k) x cost of transition(k)]

- Find an optimal layout,

LA = 100×[cost of LA-optimal layout/cost of proposed layout]

References

- Prof. Fengjun Li's EECS 448 Fall 2015 slides
- This slide set has been extracted and updated from the slides designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill 2014) by Roger Pressman