# Software Measurement


**Dr. Mark C. Paulk**

**SE 4367 – Software Testing, Verification, Validation, and Quality Assurance**

Jonsson School of Engineering and Computer Science

# *Topics: Software Testing*

**Part I: Preliminaries**

## 1. Software Testing

- **Humans, Errors, and Testing**
- **Software Quality**
- **Requirements, Behavior, and Correctness**
- **Correctness vs Reliability**
- **Testing and Debugging**
- **Test Metrics**
- **Software and Hardware Testing**
- **Testing and Verification**
- **Defect Management**
- **Test Generation Strategies**
- **Static Testing**
- **Model-Based Testing and Model Checking**
- **Types of Testing**
- **Saturation Effect**
- **Principles of Testing**

# *Two Measurement Questions*

**Are we measuring the right thing?**

- **Goal / Question / Metric (GQM)**
- **business objectives ⇔ data**
  - **cost (dollars, effort)**
  - **schedule (duration, effort)**
  - **functionality (size)**
  - **quality (defects)**

**Are we measuring it right?**

- **operational definitions**

# *Goals and Measures*

**One of the dangers in enterprises as complex as software engineering is that there are potentially so many things to measure…**

**In goal-driven measurement, the primary question is not**

***"What measures should I use?"***

**Rather, it is**

***"What do I want to know or learn?"***

**Goal-driven measurement is <u>not</u> based on a predefined set of measures.**

# *Goal-Driven Measurement*

**Goal / Question / Metric (GQM) paradigm**

- ***V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984.***

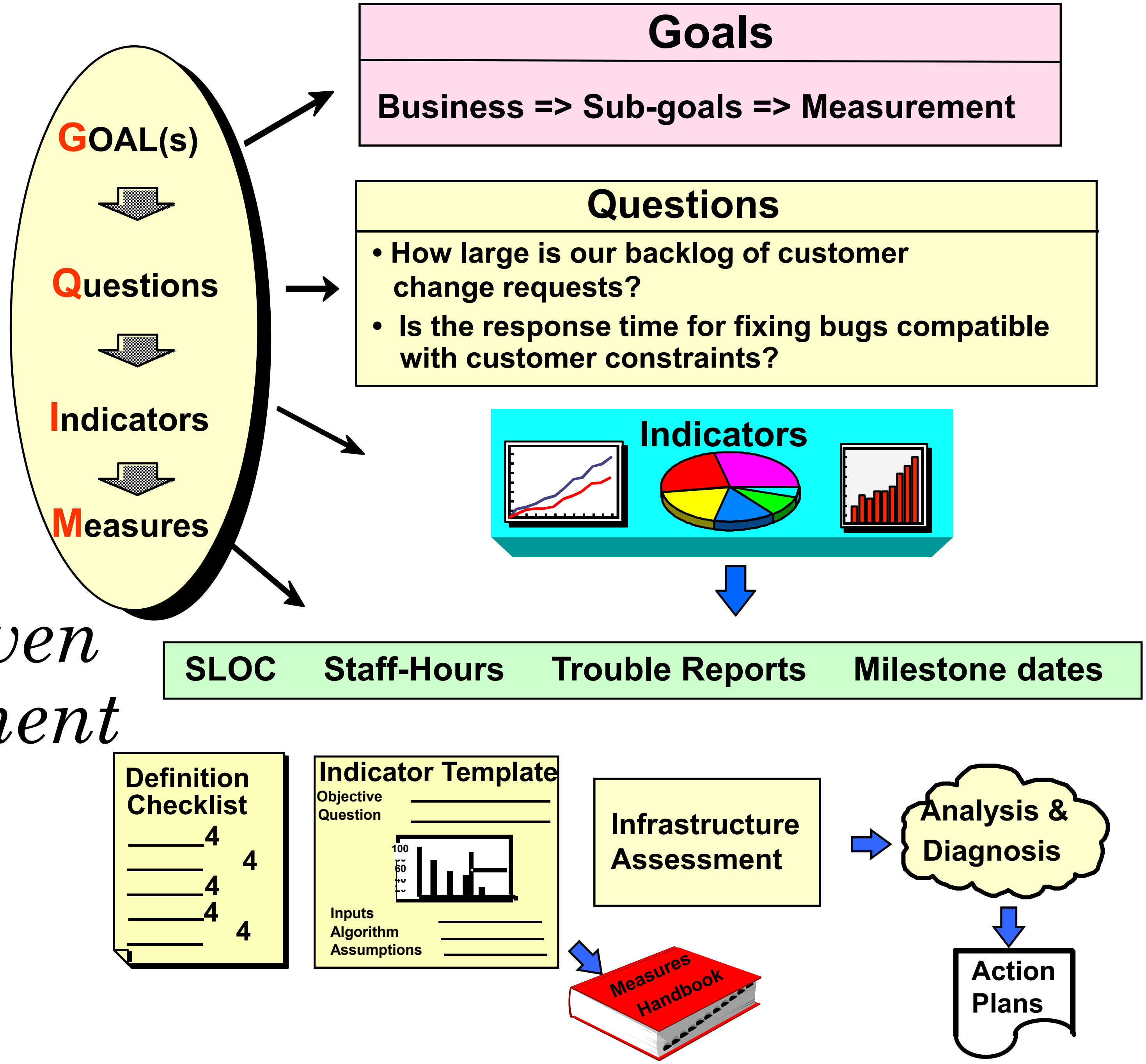
**SEI variant: goal-driven measurement**

- ***Robert E. Park, Wolfhart B. Goethert, and William A. Florac, "Goal-Driven Software Measurement – A Guidebook," CMU/SEI-96-HB-002, August 1996.***

**ISO 15939 and PSM variant: measurement information model**

- ***John McGarry, David Card, et al., <u>Practical Software Measurement: Objective Information for Decision Makers</u>, Addison-Wesley, Boston, MA, 2002.***

# *Goal-Driven Measurement*



# *Operational Definitions*

**The rules and procedures used to capture and record data**

**What the reported values <u>include</u> and <u>exclude</u>**

**Operational definitions should meet two criteria**

- ***Communication*** **– will others know what has been measured and what has been included and excluded?**
- ***Repeatability*** **– would others be able to repeat the measurements and get the same results?**

# *No True Value*

***An operational definition [is one] which reasonable men can agree on and do business with.***

***Shewhart believed his work on operational definitions to have been of greater importance than his development of the theory of variation and of the control chart.***

***There is no true value of anything.***

***Chapter 7 in Henry R. Neave, The Deming Dimension.***

# *Concerns in Operational Definitions*

**What is included or excluded? … in a line of code?**

- **comments? blank lines?**
- **compiler directives? #define? #include?**
- **variable declarations? integer i, j;?**

**Is the data binned in categories?**

- **new, modified, deleted, reused code?**
- **severity, criticality, impact of defects?**

**What is the unit of measure?**

- **hours vs minutes**
- **imperial vs metric**

**Where in the process is data collected?**

- **peer review before or after compile / unit test**

# *Measuring Software Size*

## Lines of code (LOC)

- physical lines of code
- (logical) source lines of code (SLOC, KSLOC)
- statements
- delivered source instructions (KDSI)

## Function points (FP)

- COSMIC (ISO/IEC 19761)
- FiSMA (ISO/IEC 29881)
- IFPUG (ISO/IEC 20926)
- Mk-II (ISO/IEC 20968)
- NESMA (ISO/IEC 24570)
- Bang measure
- Feature points
- Weighted Micro Function Points

# *SEI SLOC Definition Considerations*

**Whether to include or exclude**

- **executable and/or non-executable code statements**
- **code produced by programming, copying without change, automatic generation, and/or translation**
- **newly developed code and/or previously existing code**
- **product-only statements or also include support code**
- **counts of delivered and/or non-delivered code**
- **counts of operative code or include dead code**
- **replicated code**

**When the code gets counted**

- **at estimation, at design, at coding, at unit testing, at integration, at test readiness review, at system test complete**

# *Function Points*

**Albrecht (1979) based function points on the number of**

- **inputs (*Inp*)**
- **outputs (*Out*)**
- **inquiries (*Inq*)**
- **master files (*Maf*)**
- **interfaces (*Inf*)**

**For any product, the size in "function points" in its simplest form is given by**

$$FP = (4 \times Inp) + (5 \times Out) + (4 \times Inq) + (10 \times Maf) + (7 \times Inf)$$

**This is an simplified version of a multi-step process.**

# *Orthogonal Defect Classification*

**A taxonomy for defect types**

- **documentation**
- **syntax**
- **build, package**
- **assignment**
- **interface**
- **checking**
- **data**
- **function**
- **system**
- **environment**

*R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, and M.Y. Wong, "Orthogonal Defect Classification - A Concept for In-Process Measurements," IEEE Transactions on Software Engineering, November 1992.*

# *Human Nature and Measurement*

**The act of measuring and analyzing will change behavior – potentially in dysfunctional ways.**

**Use of measurement data to evaluate individuals will negatively affect the correctness and usefulness of the measurement data that are reported.**

***The squeaky wheel gets the grease…***

***What gets measured gets attention…***

# *Hawthorne Effect*

**The act of measuring (paying attention) will change behavior.**

- **self-interested behavior on the part of the "measured entity!"**
- **motivational use of measurement (Austin)**

**Is the Hawthorne effect bad?**

**Isn't the intention to change behavior?**

**Is the change "systematic?" Will it last?**

**Will management continue to "pay attention?"**

# *Dysfunctional Behavior*

**Austin's <u>Measuring and Managing Performance in Organizations</u>**

- **motivational versus information measurement**

**Dysfunctional behavior resulting from organizational measurement is <u>inevitable</u> unless**

- **measure system is "perfect"**
- **or <u>motivational</u> use is impossible**

**Is it possible to create a perfect measurement system? That addresses all possible needs?**

  - **Deming and many other measurement experts strongly oppose performance measurement, merit ratings, management by objectives, etc.**

# *Test Metrics*

**Metric – a standard of measurement**
- **syn: measure**

**Organizational measures**

**Project measures**

**Process measures**

**Product measures**
- **static**
- **dynamic**

# *McCabe Cyclomatic Complexity*

**In the control flow graph for a procedure reachable from the main procedure containing**

- **N nodes**
- **E edges**
- **p connected procedures**
  - **cyclomatic complexity is normally applied only to procedures**
  - **p is therefore 1 in practical use (frequently p is left out of the discussion of cyclomatic complexity)**
  - **Herraiz and Hassan (2011) use the maximum or average cyclomatic complexity for all functions in a file**

**V(G) = E – N + 2p**

# *Recommended Values for V(G)*

**Usual recommendation: V(G) should be less than 10**

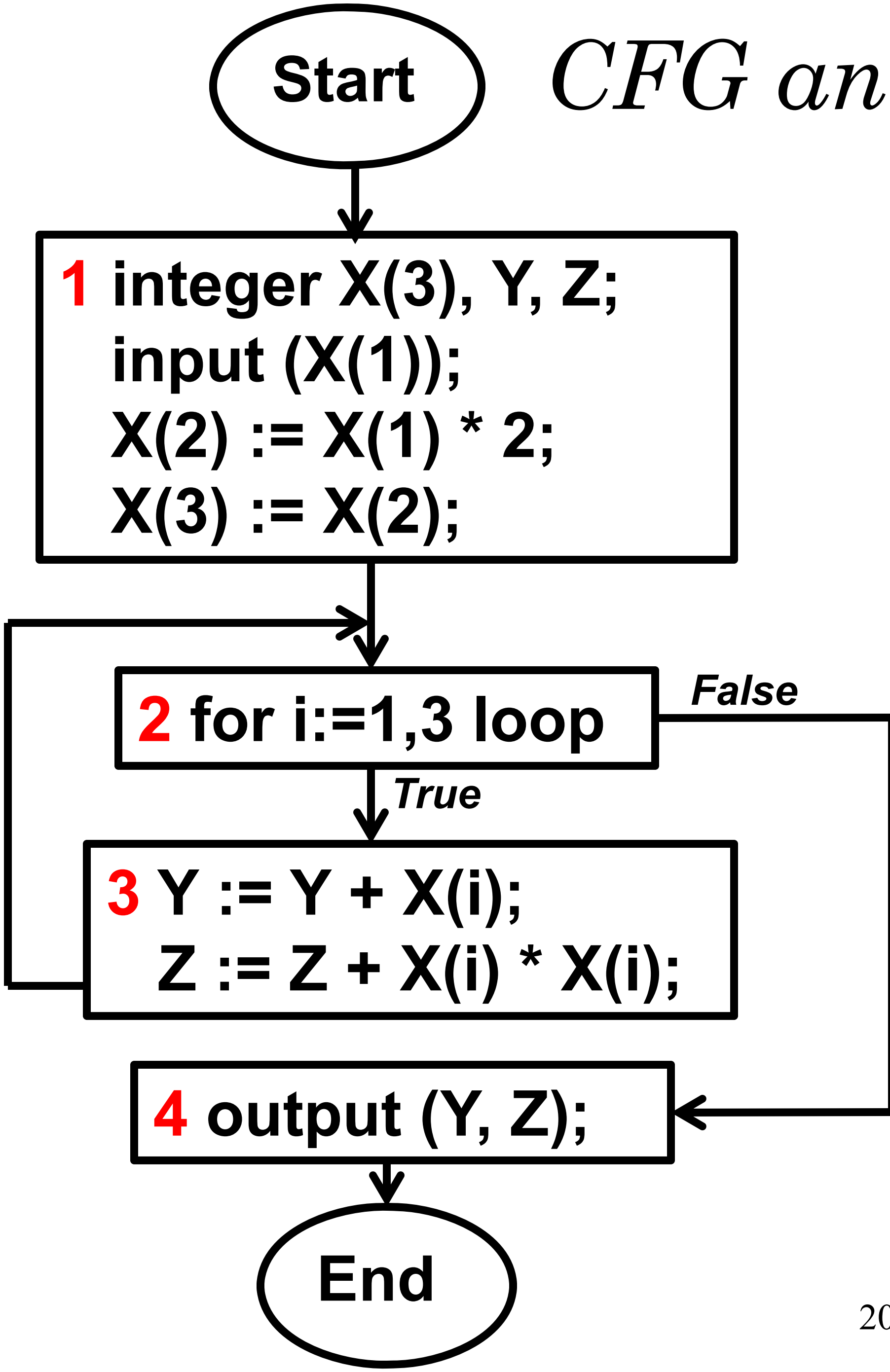
**Mathur recommends less than 5**

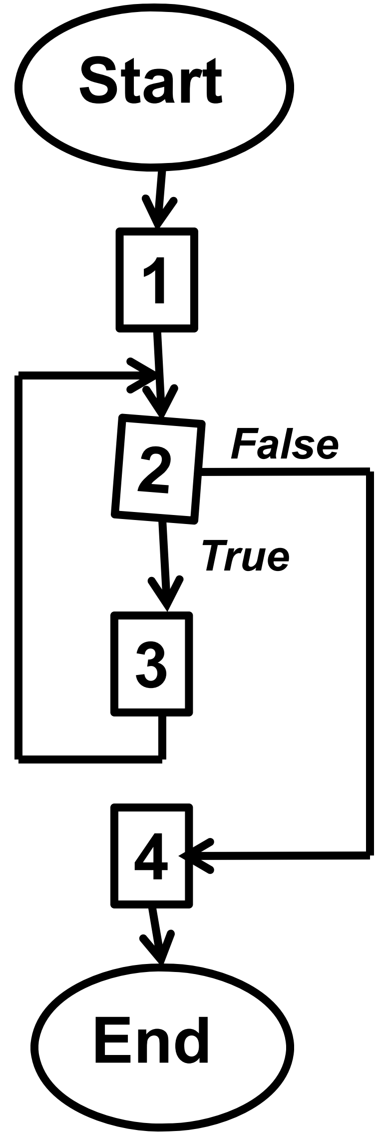
**Some corporate standards suggest that 10-20 should be considered a "gray" zone**

**Green zone: V(G) < 10**
**Yellow zone: V(G) of 10-14**
**Red zone: V(G) > 14 (Tockey 2019)**

- `Switch-case` **statements are arguably less complex**
- **Tockey suggests using** `log(#cases)` **for** `switch` **statements**

# *CFG and McCabe Example*



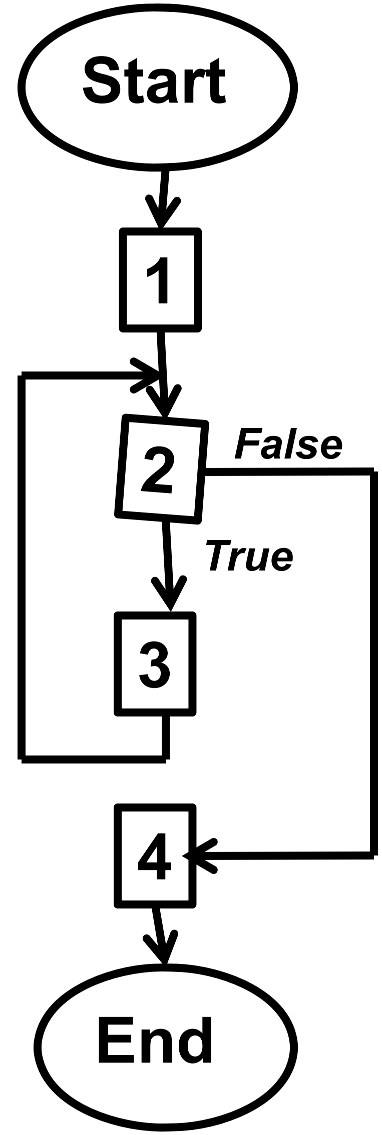



**Nodes = {Start, 1, 2, 3, 4, End}**
**N = 6**

**Edges = {(Start,1), (1,2), (2,3), (3,2), (2,4), (4,End)}**
**E = 6**

**p = 1**

**V(G) = E – N + 2p**
**= 6 – 6 + 2(1) = 2**



# *McCabe Cyclomatic Complexity for Structured Programs*

**V(G) = #decisions +1 for a structured program**

**A program with no decisions has a CFG with three nodes (including the Start and End nodes) and two edges: V(G) = 2 – 3 + 2(1) = 1**

- **adding if-then-else statement increases the number of nodes (N) by 3 and edges (E) by 4 → +1**
- **adding an if-then or while statement increases the number of nodes (N) by 2 and the number of edges (E) by 3 → +1**

**Net increase in cyclomatic complexity is 1 for each decision in the program over the base of 1.**

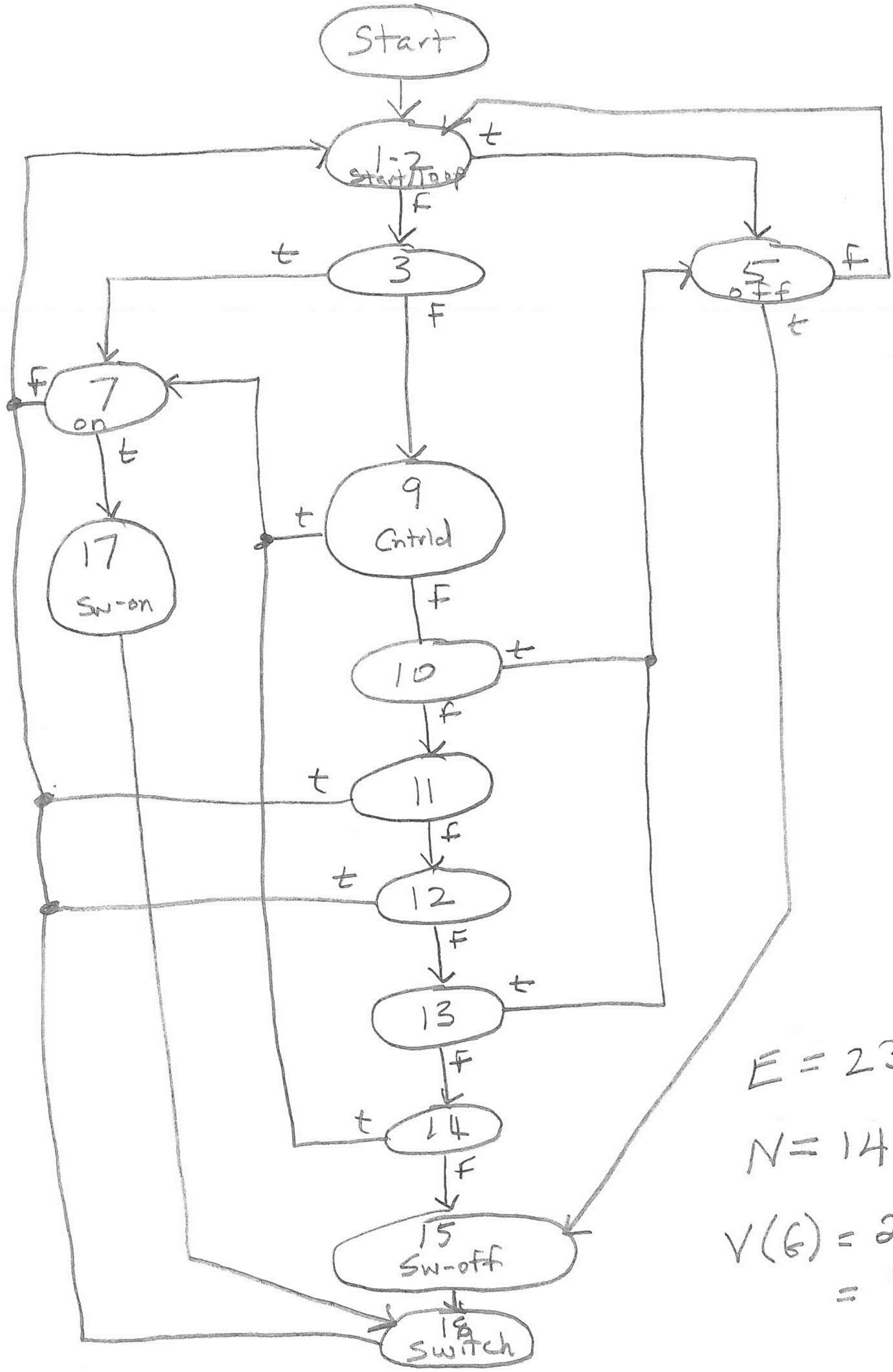
# *Unstructured Thermostat Example*

```
Start:    Get (Time-on, Time-off, Time, Setting, Temp, Switch)
          if Switch = off goto off
          if Switch = on goto on
          goto Cntrld
off:  if Heating-status = on goto Sw-off
          goto loop
on:  if Heating-status = off goto Sw-on
          goto loop
Cntrld:   if Time = Time-on goto on
          if Time = Time-off goto off
          if Time < Time-on goto Start
          if Time > Time-off goto Start
          if Temp > Setting then goto off
          if Temp < Setting then goto on
Sw-off:   Heating-status := off
          goto Switch
Sw-on:    Heating-status := on
Switch:   Switch-heating
loop:     goto Start
```

```
1  Start:    Get (Time-on, Time-off, Time, Setting, Temp, Switch)
2            if Switch = off goto off
3            if Switch = on goto on
4            goto Cntrld
5  off:  if Heating-status = on goto Sw-off
6            goto loop
7  on:   if Heating-status = off goto Sw-on
8            goto loop
9  Cntrld:   if Time = Time-on goto on
10           if Time = Time-off goto off
11           if Time < Time-on goto Start
12           if Time > Time-off goto Start
13           if Temp > Setting then goto off
14           if Temp < Setting then goto on
15 Sw-off:   Heating-status := off
16           goto Switch
17 Sw-on:    Heating-status := on
18 Switch:   Switch-heating
19 loop:     goto Start
```

Blocks

1-2
3
5
7
9
10
11
12
13
14
15
17
18




$E = 23$

$N = 14$

$V(G) = 23 - 14 + 2$
$= 11$

# *Structured Implementation*

```
loop
      -- The Get statement finds values for the given variables from the system's
-- environment.
      Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
      case Switch of
            when On => if Heating-status = off then
                              Switch-heating ; Heating-status := on ;
                        end if ;
            when Off => if Heating-status = on then
                              Switch-heating ; Heating-status := off ;
                        end if;
            when Controlled =>
                  if Time >= Time-on and Time < = Time-off then
                        if Temp > Setting and Heating-status = on then
                              Switch-heating; Heating-status = off;
                        elsif Temp < Setting and Heating-status = off then
                              Switch-heating; Heating-status := on ;
                        end if;
                  end if ;
      end case ;
end loop ;
```

## Is an infinite loop a decision?

```
loop
        -- The Get statement finds values for the given variables from the system's
-- environment.
        Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
        case Switch of
            when On => if Heating-status = off then
                           Switch-heating ; Heating-status := on ;
                       end if ;
            when Off => if Heating-status = on then
                           Switch-heating ; Heating-status := off ;
                       end if;
            when Controlled =>
                if Time >= Time-on and Time < = Time-off then
                    if Temp > Setting and Heating-status = on then
                        Switch-heating; Heating-status = off;
                    elsif Temp < Setting and Heating-status = off then
                        Switch-heating; Heating-status := on ;
                    end if;
                end if ;
        end case ;
end loop ;
```

**Number of decisions = 8**
**V(G) = 8 + 1 = 9**

# *Halstead's Software Science*

***M.H. Halstead, <u>Elements of Software Science</u>, 1977.***

**$N_1$ number of operators in a program**
**$N_2$ number of operands in a program**
**$\eta_1$ number of unique operators in a program**
**$\eta_2$ number of unique operands in a program**
**$\eta$ program vocabulary = $\eta_1 + \eta_2$**
**$N$ program length = $N_1 + N_2$**
**$V$ program volume = $N \times \log_2 \eta$**
**$D$ difficulty = $(\eta_1 / 2) \times (N_2 / \eta_2)$ *(Mathur text wrong!)***
**$E$ effort = $D \times V$**
**$B$ number of delivered bugs = $V / 3000$**
**$= (E^{2/3}) / 3000$**

# *Halstead Counts – Alternate Rules*

**Do not include {}; as operators (Mathur)**

**Count (), [], {} as one operator**

- **begin/end are usually counted as two…**

**Count if-then, begin-end, end if, end loop, etc., as a single operator**

**Count – (minus) as a sign separately from – as an operator**

- **count – separately for variables but combine with constants as part of the constant**
- **count – as an operator in all cases**

# *Halstead's Number of Errors Estimator*

**Halstead's original formulas for B (<u>Elements of Software Science</u>, page 87) were**

$$B = (E^{2/3}) / 3000$$

$$B = V / 3000$$

**The formula provided by Mathur**

$$B = 7.6\ (E^{0.667})\ (S^{0.333})$$

**comes from Schneider, 1989.**

# *Schneider's Formula for B*

**What is E? What is S?**

**You may have assumed that "S" was size, i.e., KSLOC.**

- **Mathur does not define S**
- **S is the Stroud number (18) in Halstead's software science**
- **Schneider defines S as KSLOC**

**If you read Schneider's paper, on eLearning, you would have also seen that his E is "overall reported months of programmer effort for the project."**

# *Halstead Time*

Halstead's E is in terms of discriminations per second

- Stroud number is 18 discriminations / second
  - see the discussion of Halstead Time at http://www.virtualmachinery.com/sidebar2.htm

One possible correction factor from Halstead's E to person months is 18 * 60 sec/min * 60 min/hr * 8 hr/day * 17 day/mon = 8,812,800

It is common to measure "Halstead time" in terms of minutes.

Halstead Time = E / (18 disc/sec * 60 sec/min)

# *Halstead Example*

```
begin
integer X(3), Y, Z;
input (X(1));
X(2) := X(1) * 2;
X(3) := X(2);
for i:=1,3 loop {
   Y := Y + X(i);
   Z := Z + X(i) * X(i);
   }
output (Y, Z);
end;
```

**<u>Operands</u>**

| | | |
|---|---|---|
| **X** | ***////// ////*** | **9** |
| **3** | ***///*** | **3** |
| **Y** | ***////*** | **4** |
| **Z** | ***////*** | **4** |
| **1** | ***///*** | **3** |
| **2** | ***///*** | **3** |
| **i** | ***////*** | **4** |
| | | **----** |
| | | **30** |

**Unique operands = 7**

```
begin
integer X(3), Y, Z;
input (X(1));
X(2) := X(1) * 2;
X(3) := X(2);
for i:=1,3 loop {
   Y := Y + X(i);
   Z := Z + X(i) * X(i);
   }
output (Y, Z);
end;
```

**Operators**

| | | |
|---|---|---|
| **begin** | / | **1** |
| **integer** | / | **1** |
| **(** | ///// ///// / | **11** |
| **)** | ///// ///// / | **11** |
| **,** | //// | **4** |
| **;** | ///// /// | **8** |
| **input** | / | **1** |
| **:=** | ///// | **5** |
| ***** | // | **2** |
| **for** | / | **1** |
| **loop** | / | **1** |
| **{** | / | **1** |
| **+** | // | **2** |
| **}** | / | **1** |
| **output** | / | **1** |
| **end** | / | **1** |
| | | **----** |
| | | **48** |

**Unique operators = 16**

$N_1$ operators = 48
$N_2$ operands = 30
$\eta_1$ unique operators = 16
$\eta_2$ unique operands = 7

$\eta = \eta_1 + \eta_2 = 23$
$N = N_1 + N_2 = 78$
$V = N \times \log_2 \eta = 353$
$D = (\eta_1 / 2) \times (N_2 / \eta_2) = 34$
$E = D \times V = 12{,}097$
$B = V / 3000 = 0.12$
$= 7.6 (E^{2/3}) (S^{1/3}) = 0.02$

**Halstead Time = 11 min**

```
begin
integer X(3), Y, Z;
input (X(1));
X(2) := X(1) * 2;
X(3) := X(2);
for i:=1,3 loop {
   Y := Y + X(i);
   Z := Z + X(i) * X(i);
   }
output (Y, Z);
end;
```

# *Object-Oriented Measures*

*(Chidamber and Kemerer 1994)*

**CBO (Coupling Between Objects)**

- **number of other classes that a class is coupled to**

**LCOM (Lack of Cohesion of Methods)**

- **dissimilarities between methods by using attributes used in the methods**

**NOC (Number of Children)**

- **number classes that directly inherit one class**

**DIT (Depth of Inheritance)**

- **maximum number of nodes between root and lowest node in the hierarchy**

**WMC (Weighted Methods per Class)**

- **counting the implemented methods in a class**

**RFC (Response for a Class)**

- **number of methods a class is accessible to, including methods implemented in own class as well as methods accessible due to inheritance**

# *Pareto Charts*

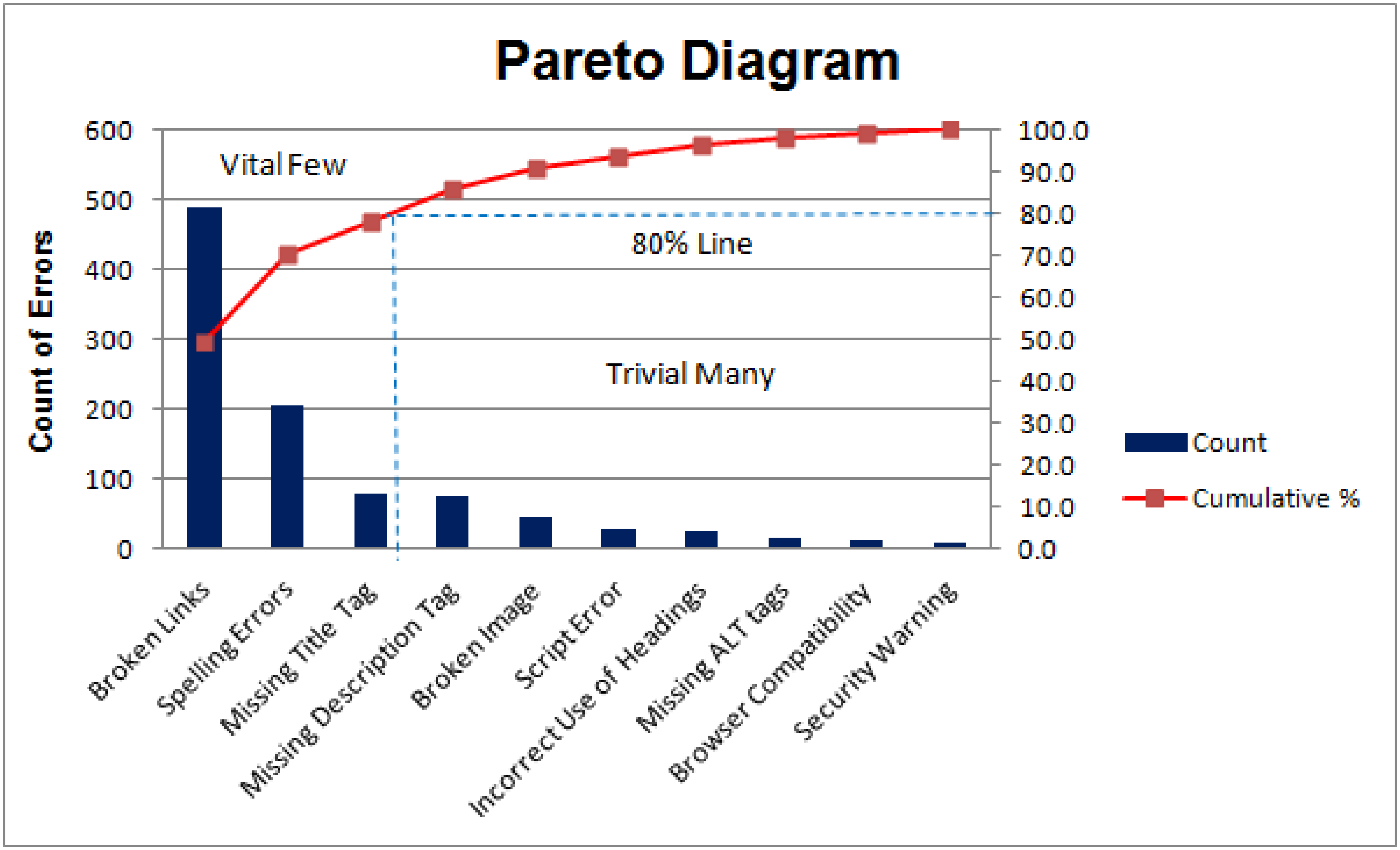
**Special form of a bar chart.**

- **order bars from largest to smallest**
- **include cumulative percentage on a second right-hand y-axis**

**Interpretation based on the "80/20 rule."**

- **80/20 is a convenient rule of thumb – actual percentages may be 70/30 or other**
- **based on power-law probability distribution**
- **from the few, many**
- **focus investigations by ranking problems, causes, or actions in terms of their amounts, frequency of occurrence, or economic consequences**

# *Pareto Chart Example*



# *About Pareto Charts*

**What if the 80/20 rule does not apply?**

**If not, you will see a "flat Pareto."**

**Possible causes**

- **an inconsistent causal process**
  - **an ad hoc or undefined process is being inconsistently implemented**
- **an inconsistent measurement process**
  - **e.g., data may be "arbitrarily" assigned to categories – the driver for ODC**
- **poor choice of categories for causes**
  - **leaving out some important causes**

# *Summary – Things to Remember*

**Goal-driven measurement**

**Operational definitions**

**Austin's motivational vs informational measurement**

**McCabe's cyclomatic complexity**

**Pareto charts – 80/20 rule**

# *Questions and Answers*