Software Engineering

Lecture-14

Metrics

Topics Covered

- Introduction
- Attributes Of Effective Software Metrics
- Metrics for SRS Attributes
- Component-level Design Metrics

DEFINITION

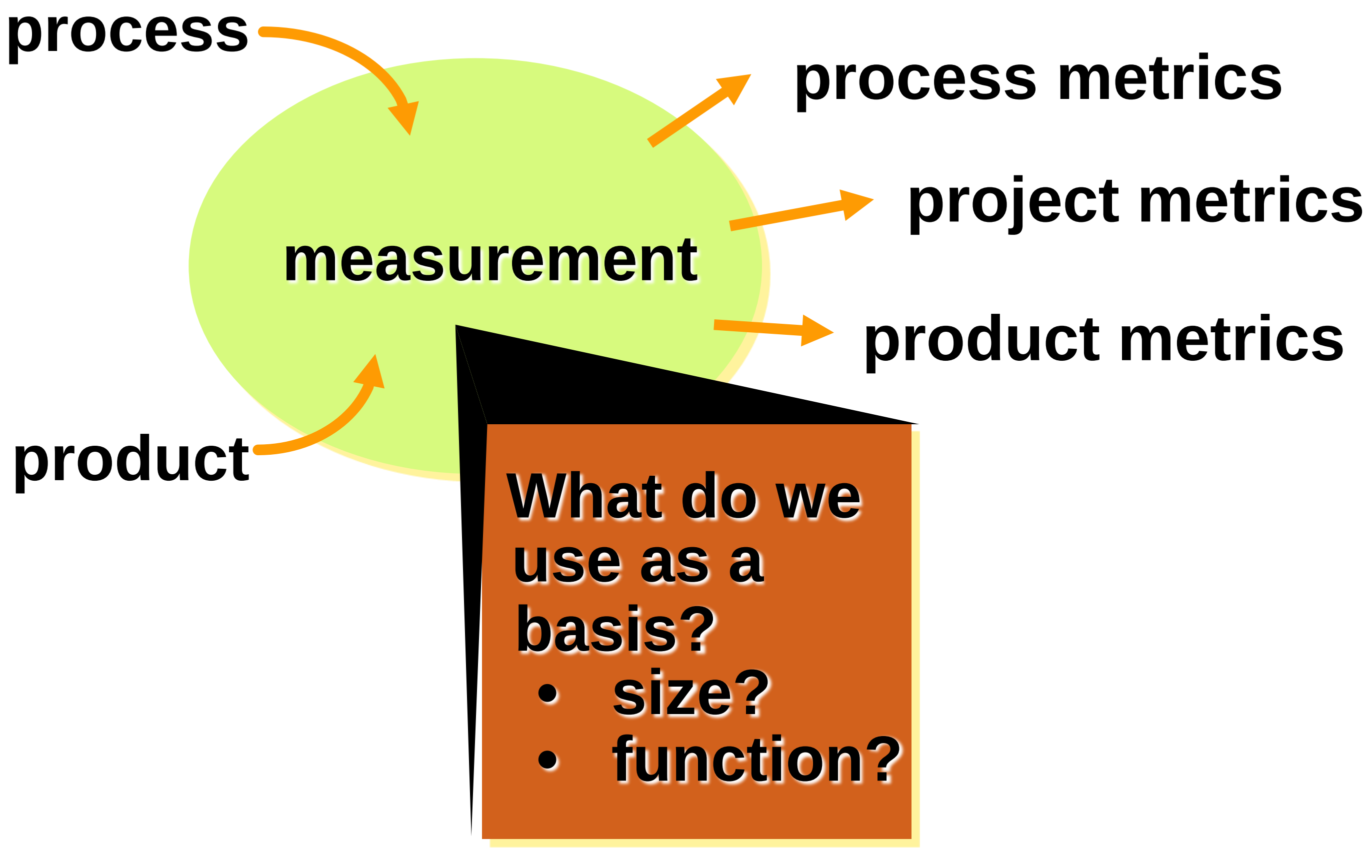
- **Measure**: A quantitative indication of the extent, amount, dimensions, capacity, or size of some attribute of a product or process.
- **Metric**: A quantitative measure of the degree to which a system, component, or process possesses a given attribute.

  A comparison of 2 or more measures.
- **Indicator**:A metric or combination of metrics that provide insight into the software process, a software project or the product.

Why Do We Measure?

- ***To understand*** what is happening during development and maintenance.
- ***To control*** what is happening on our projects.
- ***To improve*** our process and products.

A Good Manager Measures



Process Metrics

- majority focus on quality achieved as a consequence of a repeatable or managed process
- statistical SQA data
  - error categorization & analysis
- defect removal efficiency
  - propagation from phase to phase

Defect Removal Efficiency

DRE = (errors) / (errors + defects)

where

errors = problems found before release

defects = problems found after release

Project Metrics

- **Objectives:**
  - To minimize the development schedule
  - To assess product quality on an ongoing basis.
- **Examples:**
  - Effort/time per SE task
  - Errors uncovered per review hour
  - Scheduled vs. actual milestone dates
  - Changes (number) and their characteristics
  - Distribution of effort on SE tasks

Product Metrics

- **Objectives:**
  - focus on the quality of deliverables
- **Examples:**
  - measures of analysis model
  - complexity of the design
    - internal algorithmic complexity
    - architectural complexity
    - data flow complexity
  - code measures (e.g., Halstead)
  - measures of process effectiveness
    - e.g., defect removal efficiency

Measurement Process

- Formulation
- Collection
- Analysis
- Interpretation
- Feedback

Formulation 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.
- Metrics should be tailored to best accommodate specific products and processes.

Collection & Analysis Principles

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

Attributes Of Effective Software Metrics

- Simple and Computable
- Empirically and Intuitively
- Consistent and Objective
- Programming language independent
- An effective mechanism for quality feedback

Measuring Software Quality: McCall's Quality Factors

- **Product Operation**
  - Correctness
  - Reliability
  - Usability
  - Integrity
  - Efficiency
- **Product Revision**
  - Maintainability
  - Testability
  - Flexibility
- **Product Transition**
  - Reusability
  - Portability
  - Interoperability

Measuring Software Quality: McCall's Quality Factors

Reliability
- Consistency
- Accuracy
- Error-tolerance
- Simplicity

Maintainability
- Concision
- Consistency
- Modularity
- Self-documentation
- simplicity

Measuring Quality in Software Requirements Specification (SRS)

- Unambiguous
- Complete
- Correct
- Understandable
- Verifiable
- Internally consistent
- Externally consistent
- Achievable
- Concise
- Design independent
- Traceable
- Modifiable
- Electronically stored
- Executable/Interpretable
- Annotated by relative importance
- Annotated by relative stability
- Annotated by version
- Not redundant
- At right level of detail
- Precise
- Reusable
- Traced
- Organized
- Cross-referenced

METRICS FOR SRS ATTRIBUTES

- n_f = functional requirements
- n_{nf} = non-functional requirements
- n_r = total requirements = $n_f + n_{nf}$

UNAMBIGUOUS

- A SRS is unambiguous if and only if every requirement stated therein has only one possible interpretation.
- Metric:

$$Q_1 = \frac{n_{ui}}{n_r}$$

n_{ui} is the number of requirements for which all reviewers presented identical interpretations.

0 - every requirement has multiple interpretation

1 - every requirement has a unique interpretation

Completeness

- A SRS is complete if everything that the software is supposed to do is included in the SRS.
- Metric:

$$Q_2 = \frac{n_A}{n_r}$$

  - n_A is the number of requirements in block A

CORRECTNESS

- A SRS is correct if and only if every requirement represents something required of the system to be built
- Metric:

$$Q_3 = \frac{n_C}{n_C + n_I}$$

  - n_C is the number of correct requirements
  - n_I is the number of incorrect requirements

UNDERSTANDABLE

- A SRS is understandable if all classes of SRS readers can easily comprehend the meaning of all requirements with a minimum of explanation.
- Metric:

$$Q_4 = \frac{n_{ur}}{n_r}$$

n_{ur} is the number of requirements for which all reviewers thought they understood.

CONCISE

- A SRS is concise if it is as short as possible without adversely affecting any other quality of the SRS.
- Metric:

$$Q_5 = \frac{1}{size + 1}$$

  - size is the number of pages

NOT REDUNDANT

- A SRS is redundant if the same requirement is stated more than one.
- Metric:

$$Q_6 = \frac{n_f}{n_u}$$

  - n_f is the actual functions specified
  - n_u is the actual unique functions specified

High-level Design Metrics

- High-level design metrics focus on characteristics of the program architecture with an emphasis on the architectural structural and the effectiveness of modules
- Metrics:
  - Card and Glass (1990)
  - Henry and Kafura (1981)
  - Fenton (1991)

Card And Glass (1990)

- 3 software design complexity measures:
  - structural complexity
  - data complexity
  - system complexity

Card and Glass (1990)

- Structural complexity (S(*i*))

$$S(i) = f_{out}^2(i)$$

where f_{out} is the fan-out of module *i*

- Data complexity (D(*i*))

$$D(i) = v(i) / [f_{out}^2(i) + 1]$$

where v(*i*) is the number of input and output variables that are passed to and from module *i*

- System complexity (C(*i*))

$$C(i) = S(i) + D(i)$$

Henry & Kafura (1981)

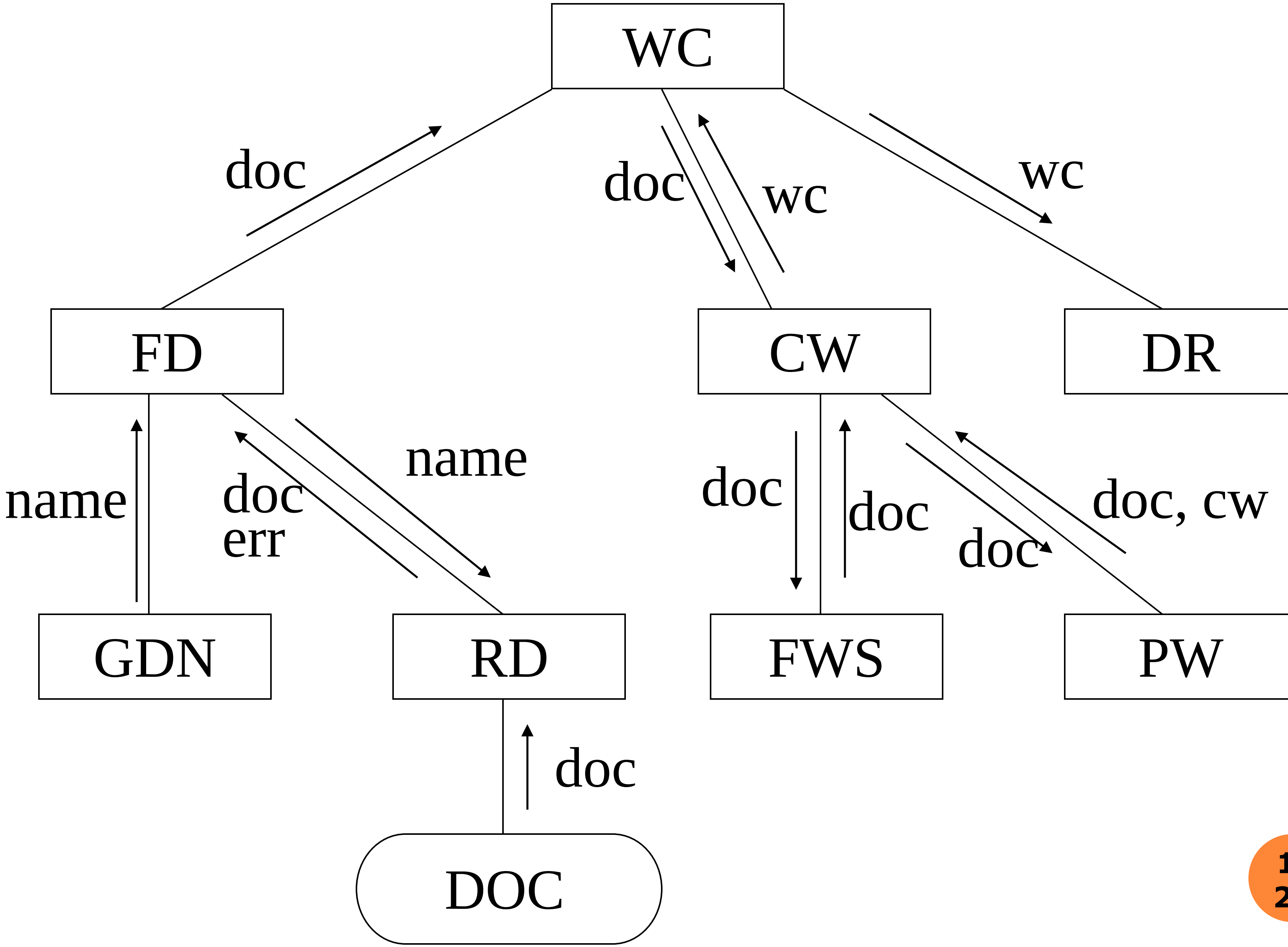
$$Complexity = length(i) \times [f_{in}(i) + f_{out}(i)]^2$$

where length (i) = the number of programming language statements in module i

$f_{in}(i)$ = the number of fan-in of module i

$f_{out}(i)$ = the number of fan-out of module i

- fan-in = the number of local flows of information that terminate at a module + the number of data structures from which information is retrieved.
- Fan-out = the number of local flows of information that emanate from a module plus_the number of data structures that are updated by that module


WC
doc
doc
wc
wc
FD
CW
DR
name
doc
err
name
doc
doc
doc
doc, cw
GDN
RD
FWS
PW
doc
DOC

Fenton (1991)

- Measure of the connectivity density of the architecture and a simple indication of the coupling of the architecture.

  r = a/n

  r = arc-to-node ratio

  a = the number of arcs (lines of control)

  n = the number of nodes (modules)

- Depth = the longest path from the root (top) to a leaf node
- Width = maximum number of nodes at any one level of the architecture

Component-level Design Metrics

- Cohesion Metrics
  - Bieman and Ott (1994)
- Coupling Metrics
  - Dhama (1995)
- Complexity Metrics
  - McCabe (1976)

BIEMAN AND OTT (1994)

- Data slice is a backward walk through a module that looks for data values that affect the module location at which the walk began.
- Data token are variables and constants defined for a module.
- Glue tokens are data tokens that lie on one or more data slice.
- Superglue tokens are the data tokens that are common to every data slice in a module.

Bieman and Ott (1994)

- Strong functional cohesion (SFC)

  SFC(i) = SG(SA(i))/tokens (i)

  SG(SA(i)) = superglue tokens

PROCEDURE SUM AND PRODUCT

```
(N  : Integer;
 Var   SumN, ProdN : Integer);
Var              I  : Integer
Begin
  SumN         : = 0;
  ProdN        : = 1;
  For   I      : =   1  to   N   do begin
       SumN : =    SumN   +   I
       ProdN: =   ProdN    +   I
  End;
End;
```

Data Slide for SumN

```
( N  : Integer;
 Var   SumN, ProdN : Integer);
Var                I  : Integer
Begin
    SumN        : = 0;
    ProdN       : = 1;
    For  I      : =   1  to  N   do begin
          SumN : =    SumN   +   I
          ProdN: =   ProdN    +   I
    End;
End;
```

Data Slice for SumN = $N_1 \cdot SumN_1 \cdot I_1 \cdot SumN_2 \cdot 0_1 \cdot I_2 \cdot 1_2 \cdot N_2 \cdot SumN_3 \cdot SumN_4 \cdot I_3$

Data Slide for ProdN

```
(        : Integer;
 Var   SumN,            : Integer);
Var                    : Integer
Begin
   SumN          : = 0;
                 : =   ;
   For           : =      to       do begin
         SumN : =   SumN   +   I
                : =            +
   End;
End;
```

Data Slice for ProdN = $N_1 \cdot ProdN_1 \cdot I_1 \cdot ProdN_2 \cdot 1_1 \cdot I_2 \cdot 1_2 \cdot N_2 \cdot ProdN_3 \cdot ProdN_4 \cdot I_4$

Data token	SumN	ProdN
N_1	1	1
$SumN_1$	1	
$ProdN_1$		1
I_1	1	1
$SumN_2$	1	
0_1	1	
$ProdN_2$		1
1_1		1
I_2	1	1
1_2	1	1
N_2	1	1
$SumN_3$	1	
$SumN_4$	1	
I_3	1	
$ProdN_3$		1
$ProdN_4$		1
I_4		1

SUPER GLUE

S_1	S_2	S_3	
I	I	I	Super Glue
I			
	I		
		I	
I	I	I	Super Glue
	I		
I	I		Glue
	I	I	Glue

Functional Cohesion

- Strong functional cohesion (SFC)

  SFC(i) = SG(SA(i))/tokens (i)

  SG(SA(i)) = superglue tokens

  SG(SumAndProduct) = 5/17 = 0.204

DHAMA (1995)

- Data and control flow coupling
  - d_i = number of input data parameters
  - c_i = number of input control parameters
  - d_o = number of output data parameters
  - c_o = number of output control parameters
- Global coupling
  - g_d = number of global variables used as data
  - g_c = number of global variables used as control
- Environmental coupling
  - w = number of modules called (fan-out)
  - r = number of modules calling the module under consideration (fan-in)

DHAMA (1995)

- Coupling metric (m_c)

  $m_c = k/M$, where $k = 1$

  $M = d_i + a^* c_i + d_o + b^* c_o + c^* g_c + w + r$

  where a=b=c=2

Coupling Metric - Example

MODULE 1

```
Package sort1 is
type array_type is arrary (1..1000) of integer;
procedure sort1 (n: in integer;
    to_be_sorted: in out array_type;
    a_or_d: in character) is
location, temp: integer;
begin
    for start in 1..n loop
    location := start;

loop to get min or max each time
for i in (start + 1)..n loop
    if a_or_d = ‘d’ then
            if to_be_sorted(i) > to_be_sorted(location) then
               location := i;
            endif;
    else if to_be_sorted(i) < to_be_sorted(location)            then
               location := i;
    endif
endloop;
    temp := to_be_sorted(start);
    to_be_sorted(start) := to_be_sorted(location);
    to_be_sorted(location) := temp;
endloop
```

COUPLING METRIC - EXAMPLE

```
MODULE2
Package sort2 is
type array_type is arrary (1..1000) of integer;
Procedure sort2 (n: in integer;
   to_be_sorted: in out array_type;
   a_or_d: in character);
procedure find_max (n: in integer;
   to_be_sorted: in out array_type;
   location: in out integer);
procedure find_min (n, start: in integer;
   to_be_sorted: in out array_type;
   location: in out integer);
procedure exchange (start: in integer;
   to_be_sorted: in out array_type;
   location: in out integer);
endsort2;
```

Coupling Metric - Example

```
procedure find_max (n, start : in
   integer; to_be_sorted: in out
   array_type; location: in out
   integer); is
begin
   location := start;
   for i in start + 1..n loop
   if to_be_sorted(i) >
          to_be_sorted(location)
   then
          location := i;
   endif;
   endloop
end find_max;
```

```
procedure find_min (n, start: in
   integer; to_be_sorted: in out
   array_type; location: in out
   integer) is
begin
   location := start;
   for i in start + 1..n loop
   if to_be_sorted(i) <
          to_be_sorted(location)
   then
          location := i;
   endif;
   endloop
end find_min;
```

Coupling Metric - Example

```
procedure exchange (start: in
    integer; to_be_sorted: in out
    array_type; location: in out
    integer) is
temp: integer;
begin
temp := to_be_sorted(start);
to_be_sorted(start) :=
    to_be_sorted(location);
to_be_sorted(location) := temp;
end exchange;
```

```
Procedure sort2 (n: in integer;
    to_be_sorted: in out array_type;
    a_or_d: in character)is
location : integer;
begin
    for start in 1..n loop
        if a_or_d = ‘d’ then
            find_max(n, start, to_be_sorted,
                location);
        else
            find_min(n, start, to_be_sorted,
                location);
        endif;
            exchange(start, to_be_sorted,
                location);
    endloop;
end sort2;
end sort2;
```

McCabe (1976)

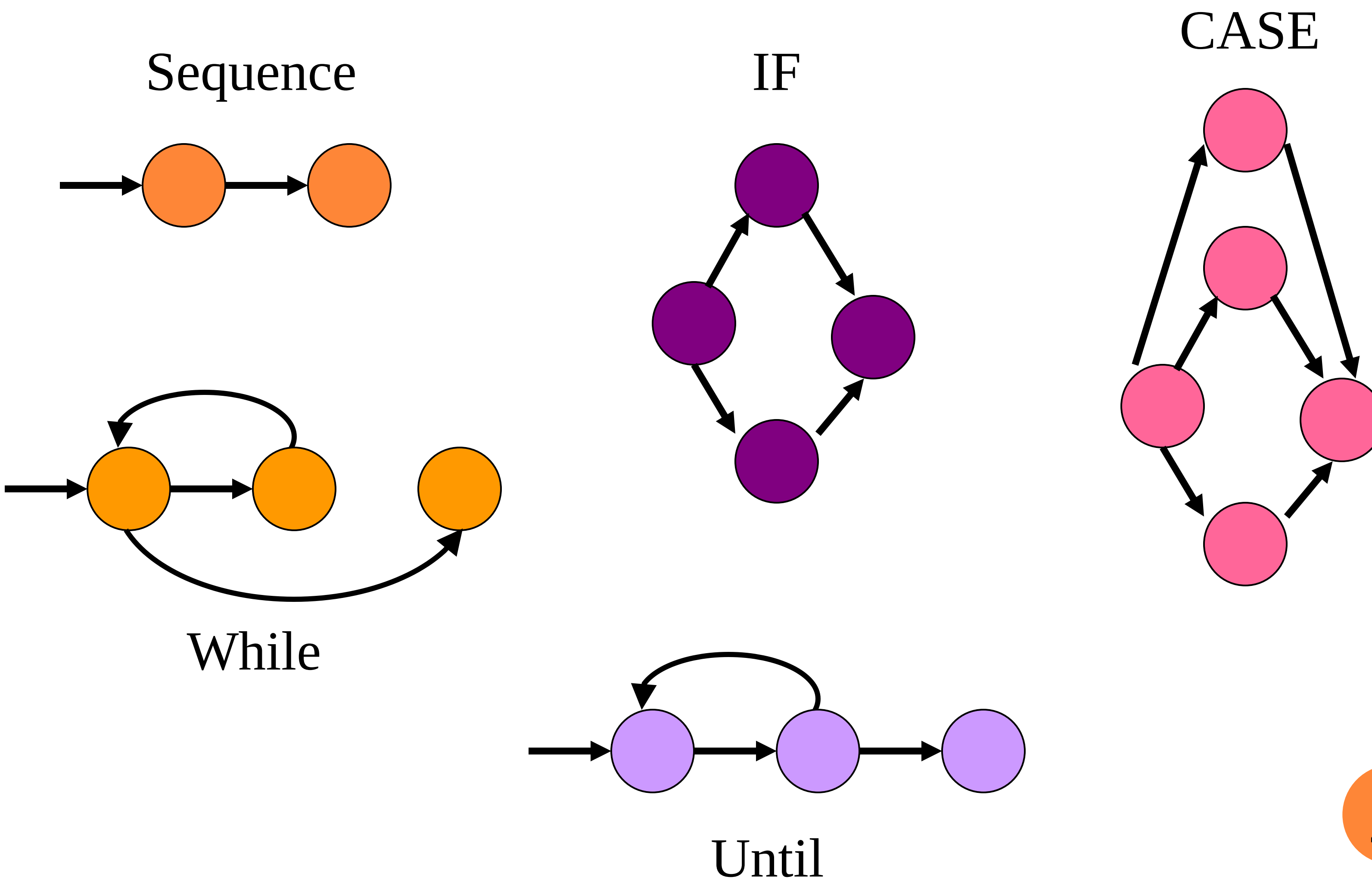
- Cyclomatic Complexity (V(G))
  - V(G) = the number of region of the flow graph + the area outside the graph
  - V(G) = E - N + 2

    where E = the number of flow graph edges

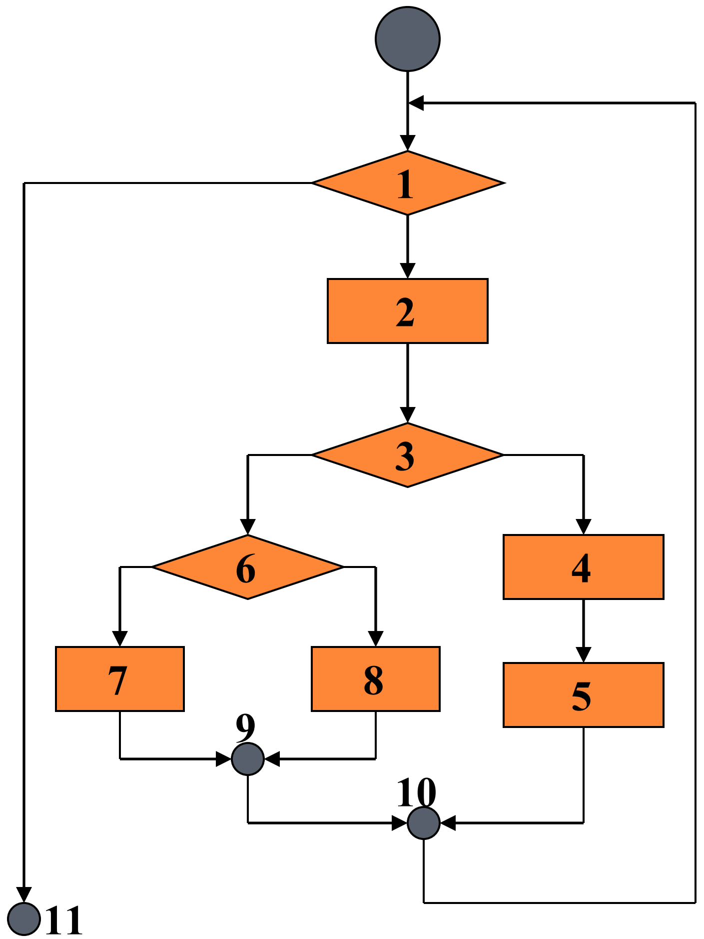
    N = the number of flow graph nodes
  - V(G) = P + 1

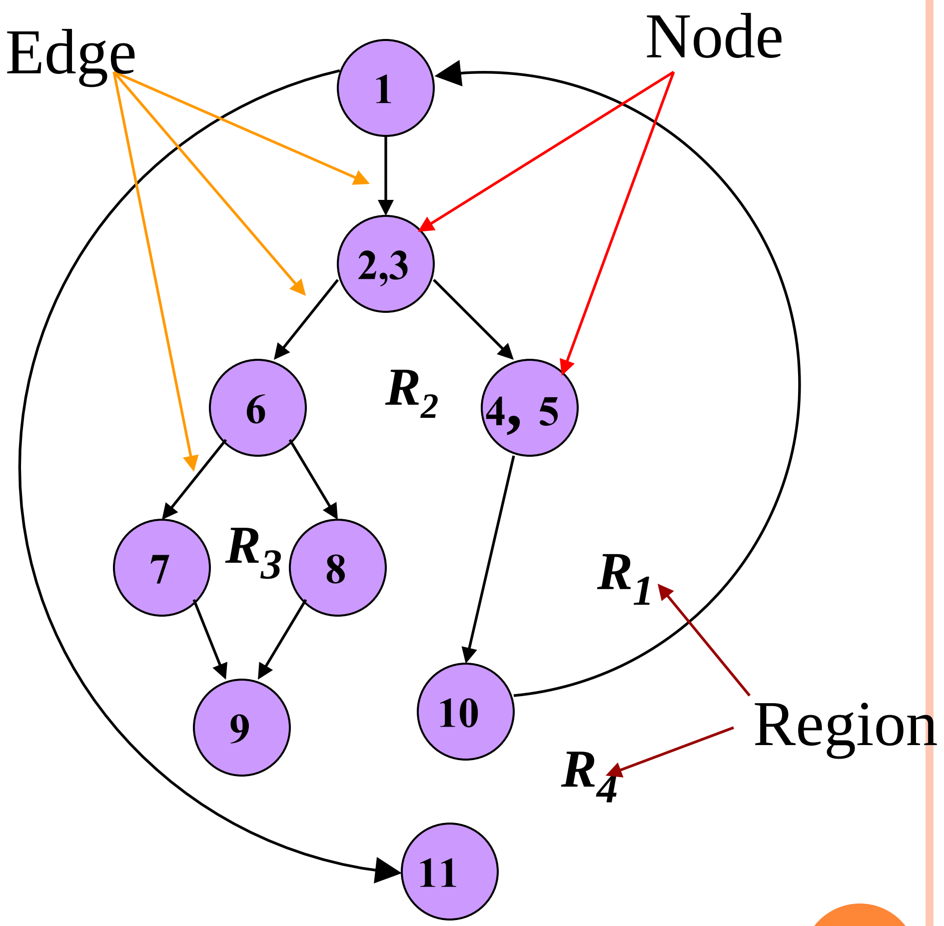
    where P = the number of predicate nodes

Flow Graph Notation



Cyclomatic Complexity - Example





METRICS FOR TESTING

- Size of the software
- High-level design metric
- Cyclomatic complexity

METRICS FOR MAINTENANCE

- Fix Backlog and Backlog Management Index
- Fix Response Time
- Percent Delinquent Fixes
- Fix Quality
- Software Maturity Index (SMI)

Fix Backlog and Backlog Management Index

- Fix backlog is a work load statement for software maintenance.
- It is a simple count of reported problems that remain opened at the end of each month or each week.
- Backlog management index (BMI)

$$BMI = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

FIX RESPONSE TIME

- Fix response time metric

  = Mean time of all problems from open to closed

PERCENT DELINQUENT FIXES

- For each fix, if the turnaround time exceeds the response time criteria by severity, then it is classified as delinquent
- Percent delinquent fixes =

$$\frac{\text{Number of fixes that exceeds the fix response time criteria by severity level}}{\text{Total number of fixes delivered in a specified time}} \times 100\%$$

Fix Quality

- Fix quality or the number of defective fixes metric = the percentage of all fixes in a time interval that are defective.
- A fix is defective if it did not fix the problem that was reported, or if it fixed the original problem but injected a new defect.
- A defective fix can be recorded in the month it was discovered or in the month when the fix was delivered.

Software Maturity Index (SMI)

- **$SMI = [M_T - (F_a + F_c + F_d)]/ M_T$**

  M_T = the number of modules in the current release

  F_c = the number of modules in the current release that have been changed

  F_a = the number of modules in the current release that have been added

  F_d = the number of modules from the preceding release that were deleted in the current release

Software Metrics Etiquette

- 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.