

A Short Course in Metrics and Measurement Dysfunction



Cem Kaner

International Software Quality Week

September, 2002

This research was partially supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Much material is from participants of *Software Test Managers Roundtable* (STMR) and the *Los Altos Workshop on Software Testing* (LAWST).

- STMR 1 (Oct-Nov 1999) focused on the question, *How to deal with too many projects and not enough staff?* Participants: Jim Bampos, Sue Bartlett, Jennifer Brock, David Gelperin, Payson Hall, George Hamblen, Mark Harding, Elisabeth Hendrickson, Kathy Iberle, Herb Isenberg, Jim Kandler, Cem Kaner, Brian Lawrence, Fran McKain, Steve Tolman and Jim Williams.
- STMR 2 (April-May 2000) focused on the topic, *Measuring the extent of testing.* Participants: James Bach, Jim Bampos, Bernie Berger, Jennifer Brock, Dorothy Graham, George Hamblen, Kathy Iberle, Jim Kandler, Cem Kaner, Brian Lawrence, Fran McKain, and Steve Tolman.
- STMR 5 (Oct. 2001) focused on *Measuring the effectiveness of test groups.* Lisa Anderson, Laura Anneker, James Bach, Sue Bartlett, Harold Crawford, David Gelperin, Mark Harding, Doug Hoffman, Cem Kaner, Brian Lawrence, Hung Quoc Nguyen, Alberto Savoia, Jennifer Smith-Brock, Steve Tolman, Jo Webb, Jim Williams, Garrin Wong,
- STMR 6 (May 2002) focused on *Measuring the effectiveness of software testers.* Laura Anneker, James Bach, Sue Bartlett, Rex Black, Jennifer Smith-Brock, Doug Hoffman, Kathy Iberle, Cem Kaner, Brian Lawrence, Bret Pettichord, Sid Snook, Steve Tolman, and Jo Webb.
- LAWST 8 (December 4-5, 1999) focused on *Measurement.* Participants: Chris Agruss, James Bach, Jaya Carl, Rochelle Grober, Payson Hall, Elisabeth Hendrickson, Doug Hoffman, III, Bob Johnson, Mark Johnson, Cem Kaner, Brian Lawrence, Brian Marick, Hung Nguyen, Bret Pettichord, Melora Svoboda, and Scott Vernon.

What is measurement?

- Is measurement really “the assignment of numbers to objects or events according to a clear cut rule”?
 - No, it can’t be. If it was, then many inappropriate rules would do.
- **Measurement is the assignment of numbers to objects or events (attributes) according to a rule derived from a model or theory.**
- A software metric is a standard way of measuring some attribute or result of the software process. Examples of these attributes are size, costs, defects, communications, difficulty and environment.

Types of attributes often measured

- **Resource**
 - Amount of resource available and/or used
- **Process**
 - Attributes of the development artifacts (other than the product), such as specifications, test materials
 - Attributes of the methods and practices employed
- **Product**
 - Attributes of the product under development, such as size, reliability, usability.
- **Impact**
 - The effect of the product, such as support costs, changed user productivity, change in user safety.

Why measure? (Some examples—add your own)

- Track project progress
- Gain control of processes
- Demonstrate the productivity of your staff
- Demonstrate the quality of your work
- Compare different engineering practices
- Increase your credibility with your management
- Identify where improvements are needed
- Determine (relative) complexity or other attributes of the software
- Help us understand whether we have achieved a certain quality level (value on some desirable attribute, such as reliability, performance, usability, accessibility, etc.)
- Gain control of characteristics of the products you make
- Gain the respect of your customers
- Demonstrate the effectiveness of the product
- Learn more about software engineering
- Evaluate models, provide a basis for scientific development of better ways to produce better products.

Models

- It's an abstraction—some details are omitted or simplified
 - Try to measure distances on a subway map
 - What is the scale of the subway map?
 - Is it useful?
- Abstractions allow us to focus on a few variables and their relationships.
- Abstractions allow us to use mathematics to study relationships.

Simple measurement



- You have a room full of tables that appear to be the same length. You want to measure their lengths.
- You have a one-foot ruler.
- You use the ruler to measure the lengths of a few tables. You get:
 - 6.01 feet
 - 5.99 feet
 - 6.05 feet
- You conclude that the tables are “6 feet” long.

Simple measurement (2)

- Note the variation
 - Measurement errors using the ruler
 - Manufacturing variation in the tables
- Note the rule:
 - We are relying on a direct matching operation and on some basic axioms of mathematics
 - The sum of 6 one-foot ruler-lengths is 6.
 - A table that is 6 ruler-lengths long is twice as long as one that is 3 ruler-lengths long.
- These rules don't always apply. What do we do when we have something hard to measure?

A Framework for Measurement

- A measurement involves at least 10 factors:
 - **Attribute to be measured**
 - appropriate scale for the attribute
 - variation of the attribute
 - **Instrument that measures the attribute**
 - scale of the instrument
 - variation of measurements made with this instrument
 - **Relationship between the attribute and the instrument**
 - **Likely side effects of using this instrument to measure this attribute**
 - **Purpose**
 - **Scope**

Framework for Measurement

- Attribute** ➤ Extent of testing – *What does that mean?*
- Instrument** ➤ What should we count? *Lines? Bugs? Test cases? Hours? Temper tantrums?*
- Mechanism** • How will increasing “extent of testing” affect the reading (the measure) on the instrument?
- Side Effect** ➤ If we do something that makes the measured result look better, will that mean that we’ve actually increased the extent of testing?
- Purpose** ➤ Why are we measuring this? What will we do with the number?
- Scope** ➤ Are we measuring the work of one tester? One team on one project? Is this a cross-project metrics effort? Cross-departmental research?

Attributes and Instruments

Length	Ruler
Duration	Stopwatch
Speed	Ruler / Stopwatch
Sound energy	Sound level meter
Loudness	Sound level comparisons by humans
Tester goodness	??? Bug count ???
Code complexity	??? Branches ???
Extent of testing???	???
----Product coverage	??? Count statements / branches tested ???
---- <u>Proportion</u> of bugs that we've found	??? Count bug reports or graph bug curves???

Defining the Attribute



- Imagine being on the job. Your local PBH (pointy-haired boss) drops in and asks

“So, tell me.

How much testing have you gotten done?”

The Question is Remarkably Ambiguous

Common answers are based on the:

Product ➤ We've tested 80% of the lines of code.

Plan ➤ We've run 80% of the test cases.

Results ➤ We've discovered 593 bugs.

Effort ➤ We've worked 80 hours a week on this for 4 months.
We've run 7,243 tests.

The Question is Remarkably Ambiguous

Common answers are based on the:

Obstacles ➤ We've been plugging away but we can't be efficient until X, Y, and Z are dealt with.

Risks ➤ We're getting a lot of complaints from beta testers and we have 400 bugs open. The product *can't be* ready to ship in three days.

Quality of Testing ➤ Beta testers have found 30 bugs that we missed. Our regression tests seem ineffective.

History across projects ➤ At this milestone on previous projects, we had fewer than 12.3712% of the bugs found still open. We should be at that percentage on this product too.

What Are We Measuring?

- Before we can measure something, we need some sense of what we're measuring. It's easy to come up with "measurements" but we have to understand the relationship between the thing we want to measure and the statistic that we calculate to "measure" it.
- **If we want to measure the "extent of testing", we have to start by understanding what we mean by "extent of testing."**

Surrogate measures

- "Many of the attributes we wish to study do not have generally agreed methods of measurement. To overcome the lack of a measure for an attribute, some factor which can be measured is used instead. This alternate measure is presumed to be related to the actual attribute with which the study is concerned. These alternate measures are called surrogate measures."
 - Mark Johnson's MA Thesis
- "Surrogates" provide unambiguous assignments of numbers according to rules, but they don't provide an underlying theory or model that relates the measure to the attribute allegedly being measured.

Consider bug counts

- Do bug counts measure testers?
- Do bug counts measure thoroughness of testing?
- Do bug counts measure the effectiveness of an automation effort?
- Do bug counts measure how near we are to being ready to ship the product?

How would we know?

Bug counts and testers

To evaluate an instrument that is supposed to measure an attribute, we have to ask two key questions:

- What *underlying mechanism*, or fundamental relationship, justifies the use of the reading we take from this instrument as a measure of the attribute? *If the attribute increases by 20%, what will happen to the reading?*
- What can we know from the instrument reading? How tightly is the reading traceable to the underlying attribute? *If the reading increases by 20%, does this mean that the attribute has increased 20%.* If the linkage is not tight, we risk serious *side effects* as people push the reading (the “measurement”) up and down without improving the underlying attribute.

Bug counts and testers: mechanism?



Suppose we could improve testing by 20%.

This might mean that:

- We find more subtle bugs that are important but that require more thorough investigation and analysis
- We create bug reports that are more thorough, better researched, more descriptive of the problem and therefore more likely to yield fixes.
- We do superb testing of a critical area that turns out to be relatively stable.

The bug counts might even go down, even though tester goodness has gone up.

Bug counts & testers: Side effects



What if you could increase the count of reported bugs by 20%?

If you reward testers for higher bug counts, won't you make changes like these more likely?

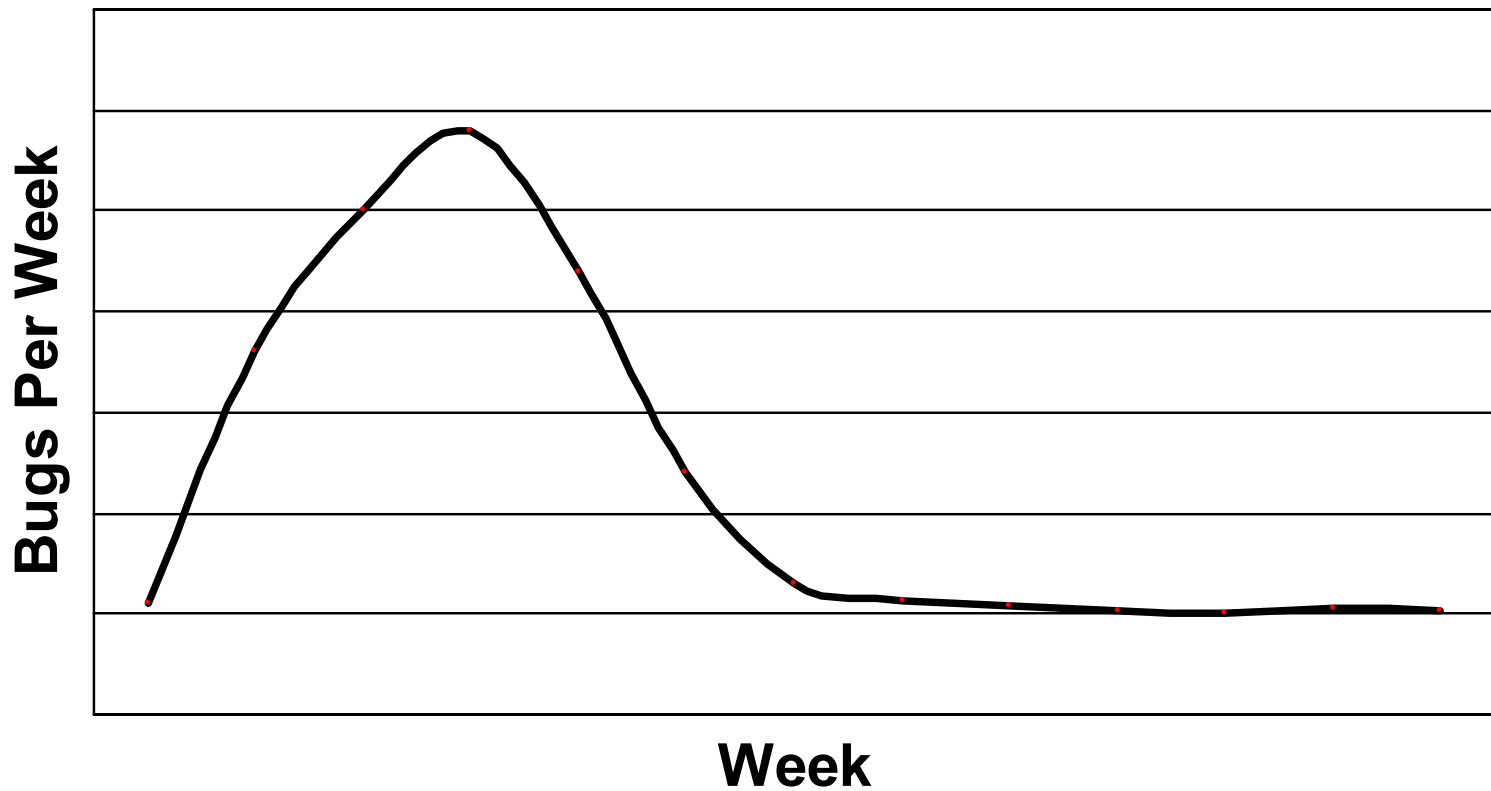
- Testers report easier-to-find, more superficial bugs
- Testers report multiple instances of the same bug
- Programmers dismiss design bugs as non-bugs that testers put in the system to raise their bug counts
- No one will work on the bug tracking system or other group infrastructure.
- Testers become less willing to spend time coaching other testers.

Example: Bug Counts

- Attribute** ➤ Not sure. Maybe we're thinking of percentage found of the total population of bugs in this product.
- Instrument** ➤ Bugs found. (Variations: bugs found this week, etc., various numbers based on bug count.)
- Mechanism** ➤ If we increase the extent of testing, does that result in more bug reports? *Not necessarily.*
- Side Effect** ➤ If we change testing to maximize the bug count, does that mean we've achieved more of the testing? *Maybe in a trivial sense, but what if we're finding lots of simple bugs at the expense of testing for a smaller number of harder-to-find serious bugs.*

The Bug Curve

What Is This Curve?



Weibull Model Assumptions

(From Eric Simmons's Talk at this Meeting)

1. Testing occurs in a way that is similar to the way the software will be operated.
2. All defects are equally likely to be encountered.
3. All defects are independent.
4. There is a fixed, finite number of defects in the software at the start of testing.
5. The time to arrival of a defect follows the Weibull distribution.
6. The number of defects detected in a testing interval is independent of the number detected in other testing intervals for any finite collection of intervals.

The Weibull Distribution

- Eric Simmons gives a good and clear talk on this distribution and presents interesting data. The fact that we sharply disagree about this metric should not be taken as criticism of him.
- That said, I think it is outrageous that we rely on a distributional model when every assumption it makes about testing is obviously false.
- Eric points out that “Luckily, the Weibull is robust to most violations.” This illustrates the use of surrogate measures—we don’t have an attribute description or model for the attribute we really want to measure, so we use something else, that is allegedly “robust”, in its place.
- The Weibull distribution has a shape parameter that allows it to take a very wide range of shapes. If you have a curve that generally rises then falls (one mode), you can approximate it with a Weibull.

BUT WHAT DOES THAT TELL US? HOW SHOULD WE INTERPRET IT?

Example: Bug Curves

- Attribute** ➤ We have a model of the rate at which new bugs will be found over the life of the project.
- Instrument** ➤ Bugs per week. A key thing that we look at is the agreement between the predictive curve and the actual bug counts.
- Mechanism** ➤ As we increase the extent of testing, will our bug numbers conform to the curve? *Not necessarily. It depends on the bugs that are left in the product.*
- Side Effect** ➤ If we do something that makes the measured result look better, will that mean that we've actually increased the extent of testing? *No, no, no. See side effect discussion.*

Side Effects of Bug Curves

Earlier in testing: (Pressure is to increase bug counts)

- Run tests of features known to be broken or incomplete.
- Run multiple related tests to find multiple related bugs.
- Look for easy bugs in high quantities rather than hard bugs.
- Less emphasis on infrastructure, automation architecture, tools and more emphasis of bug finding. (Short term payoff but long term inefficiency.)

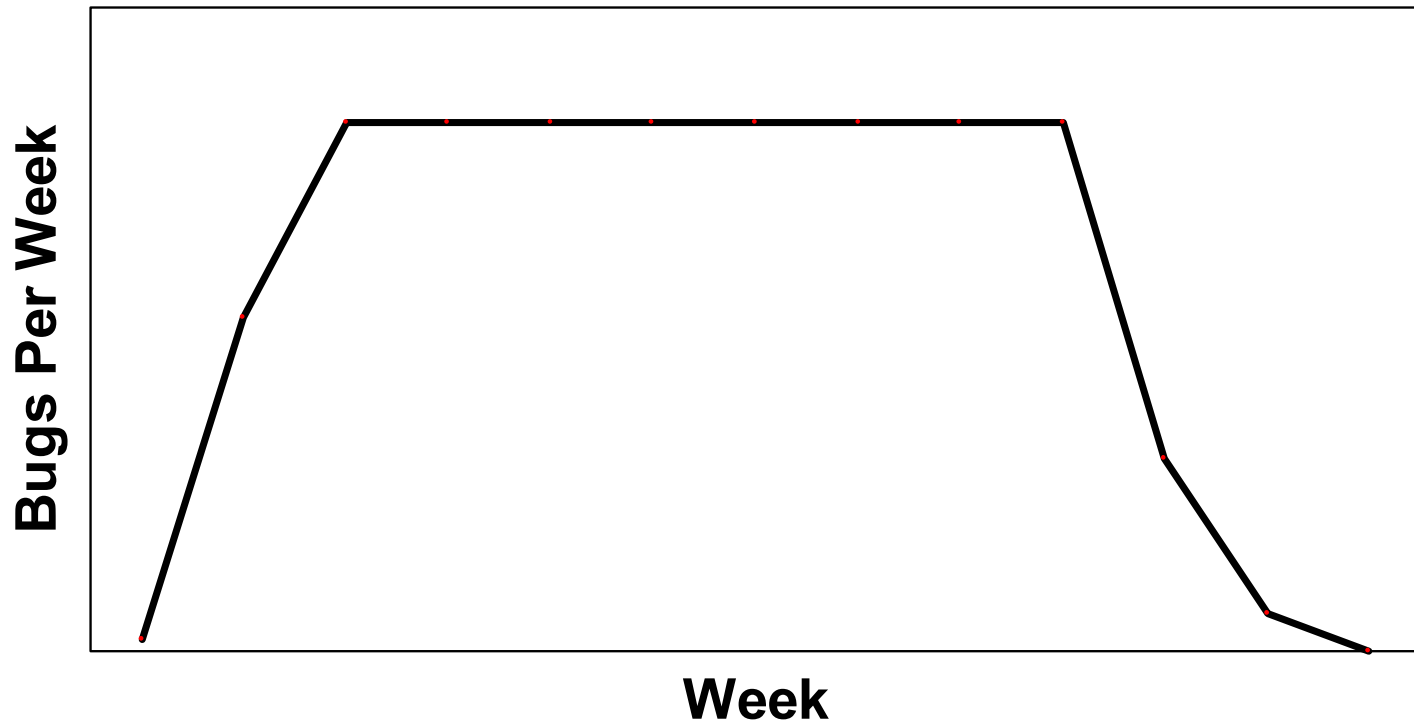
Some Side Effects of Bug Curves

Later in testing: (Pressure is to decrease new bug rate)

- Run lots of already-run regression tests
- Don't look as hard for new bugs.
- Shift focus to appraisal, status reporting.
- Classify unrelated bugs as duplicates
- Class related bugs as duplicates (and closed), hiding key data about the symptoms / causes of the problem.
- Postpone bug reporting until after the measurement checkpoint (milestone). (Some bugs are lost.)
- Report bugs informally, keeping them out of the tracking system
- Testers get sent to the movies before measurement checkpoints.
- Programmers ignore bugs they find until testers report them.
- Bugs are taken personally.
- More bugs are rejected.

Bug Curve Counterproductive?

Shouldn't We Strive For This ?



Austin on Measurement Dysfunction



- Schwab & U.S. Steel
 - Counting ingots
 - How might these people have improved *measured* productivity?

Robert Austin, *Measuring and Managing Performance in Organizations*.

Measurement Dysfunction

- In an organizational context, dysfunction is defined as the consequences of organizational actions that interfere with the attainment of the spirit of stated intentions of the organization.
(Austin, p. 10)
- Dysfunction involves fulfilling the letter of stated intentions but violating the spirit.

Measurement Dysfunction



- Examples from law enforcement
 - Quotas
 - Percentage successful prosecutions
 - Ratio of arrests to prosecutions
 - Measured from the perspective of the police
 - Measured from the perspective of the prosecutor
 - Measured from the perspective of the crime lab

Austin on the 2-Party Model

- Principal
 - E.g. the employer, the person who wants the result and who directly profits from the result.
 - In Austin's model, we assume that the employer is motivated by maximum return on investment
- Agent
 - E.g. the employee.
 - In Austin's model, the employee wants to do the least work for the most money

Austin: 2-Party Model – Supervisory Issues



- No supervision
 - No work
- Partial supervision
 - Work only on what is measured
- Full supervision
 - Work according to the production guidelines laid out by the employer

Austin's 3-Party Model

- Principal (Employer)
 - With respect to the agent, same as before: least pay for the most work.
 - With respect to the customer, wants to increase customer satisfaction
- Agent (Employee)
 - With respect to principal, same as before: least work for the most pay
 - With respect to the customer, motivated by customer satisfaction
- Customer
 - Wants the most benefit for the lowest price

Austin's 3-Party Model Supervisory Model



- No supervision
 - Agent works to the extent that increasing customer satisfaction provides more “benefit” to the agent (worker) than it costs the agent to provide the work
- Full supervision
 - Agent does exactly what should be done to increase customer satisfaction

Austin's 3-Party Model Supervisory Model

- Partial supervision
 - Agent is motivated by
 - increased customer satisfaction and by
 - rewards for performing along measured dimensions.
 - To the extent that the agent works in ways that don't maximize customer satisfaction at a given level of effort, we have *distortion*.
 - To the extent that the agent works in ways that reduce customer satisfaction below the level that would be achieved without supervision, we have *dysfunction*

Austin's 3-Party Model Supervisory Model

- Back to full supervision
 - What benefits are associated with full supervision?
 - What costs are associated with full supervision?
 - Imagine you were supervising a programmer who had a 6-week (best guess) programming task. What would you have to know / measure in order to achieve full supervision?
 - In general, what are the obstacles to achieving full supervision of knowledge workers?
 - Is it reasonable to try for full supervision or are we stuck with partial (or no) supervision?

Austin's 3-Party Model Supervisory Model

- A key aspect of this model is that it builds in the notion of internal motivation.
- Under **full supervision** with forcing contracts, perhaps internal motivation is unnecessary. (I disagree, but perhaps we can pretend that it is unnecessary.)
- Under **partial supervision** and **no supervision**, internal motivation plays an important role in achieving customer satisfaction and in eliciting effort and results from the agent.
- This comes into play in Austin's vision of delegatory management.

Tie This Back to Our Measurement Model

- Full supervision:
 - There is an instrument for every significant attribute and every instrument directly measures the attribute. ***PROBABLY IMPOSSIBLE.***
- Partial supervision:
 - Some attributes are unmeasured or are measured using proxies (or surrogates), measures that are loosely tied to the attribute under study.
- Distortion:
 - Side effects
- Dysfunction
 - Really bad side effects
- Informational vs Motivational Measures
 - Purpose of use
 - Scope of use

What protects data gathered for informational purposes from being used for motivational purposes?

Flowgraphs: Basic definitions

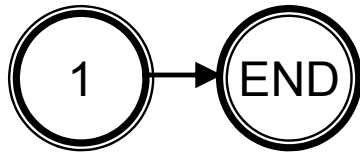
(Tutorial Slides for Practice at Home)



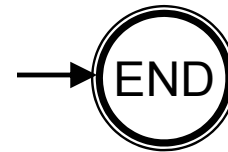
- Flowgraph is a directed graph
 - Nodes
 - Start
 - Terminal
 - Predicate / Decision
 - Procedural
 - Edges - connect two nodes
 - Arcs - directed edges

Nodes

A single statement



The GOTO statement

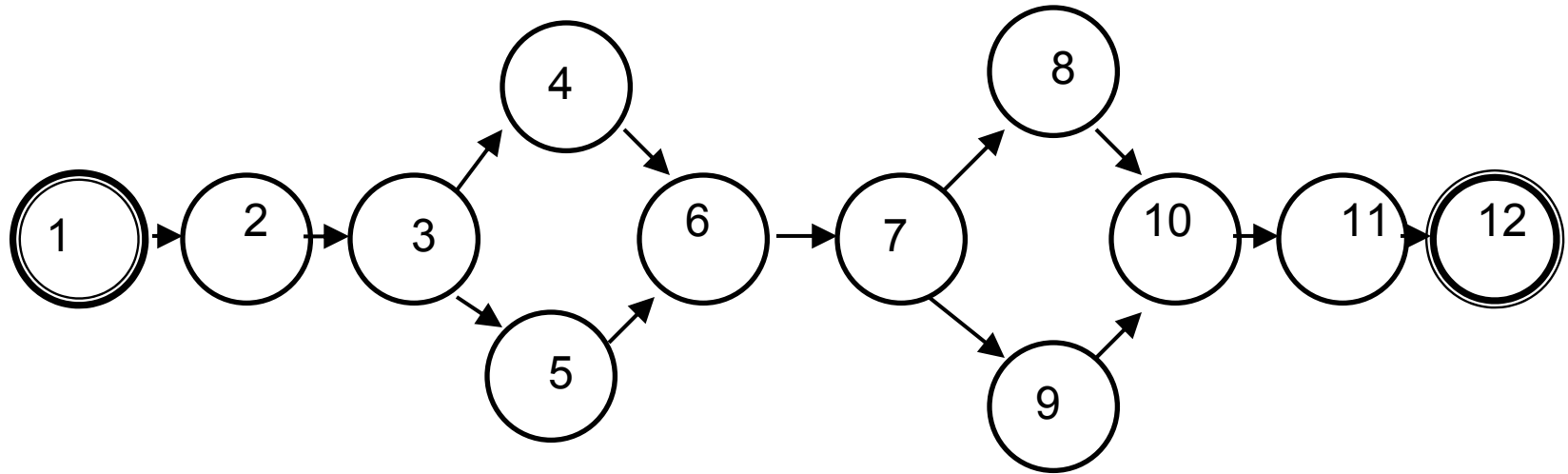


- In our course (and in our text), nodes are “statement nodes”. They normally correspond to a single statement. Other computer scientists often represent states with nodes. The action that transforms the program from one state to another (such as execution of a statement) is shown on an arc.
- A GOTO statement does not appear on a node. It is a pure vector, pointing to the place to transfer control.
- The terminal node has a single function—it is the end, such as an endif. It is a logical connection point, not a source of action.

Degree of a Node

- In-degree = # arcs to node
 - In-degree of start node is often but not necessarily 0
- Out-degree = # arcs from the node
 - Out-degree of terminal node = 0
- Procedure Node = out-degree = 1
- Predicate Node = out-degree > 1

Zuse example Figure 2.3



Identify the following types of nodes: Start, Terminal, Predicate, Procedure
Identify the in-degrees of the nodes. Can you find nodes with in-degree of 0? 1? 2?
Identify the out-degrees of the nodes. Can you find nodes with out-degree of 0? 1? 2? 3?

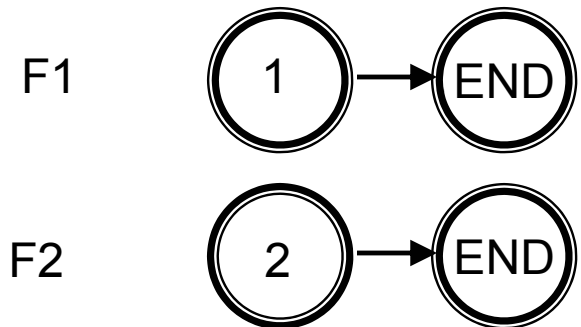
“Proper” Flowgraph

- Execution starts at the start node, S and ends at the terminal node, T
- For each node, N,
 - There is a path from start node, S, to N
 - There is a path from N to terminal node, T
 - N could be replaced with a proper flowgraph and the resulting flowgraph will still be a proper flowgraph

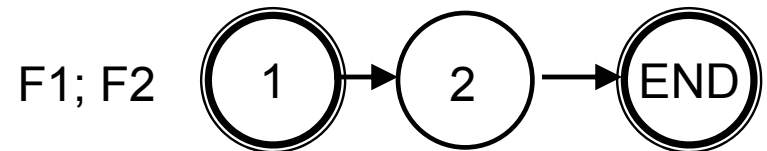
Sequencing and Nesting of Flowgraphs

- If F1 and F2 are two flowgraphs
 - We make a **sequence** of F1 and F2 by replacing the terminal node of F1 with the start node of F2.
 - Notation:
 - F1; F2
 - Seq (F1, F2)
 - P2 (F1, F2)
 - F1 o F2

Sequencing these:



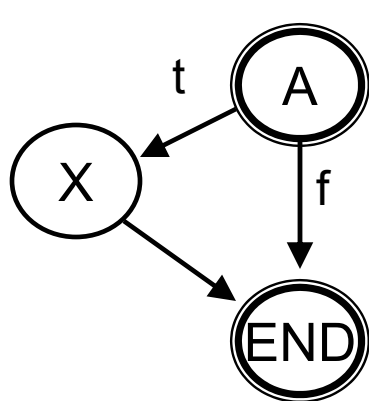
Yields this



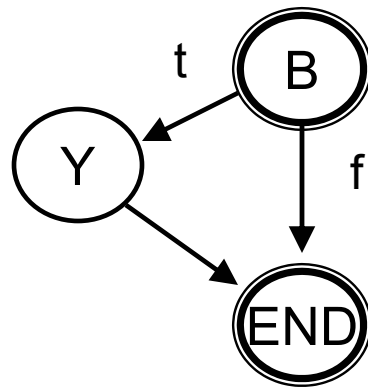
Sequencing and Nesting of Flowgraphs

- If F1 and F2 are two flowgraphs and X is a procedure node.
 - F2 is **nested in F1 at X** if we replace the arc from X with the flowgraph F2 (F2's start node is X)
 - Notation:
 - F1(F2 on X)
 - F1(F2) is OK if there is no ambiguity

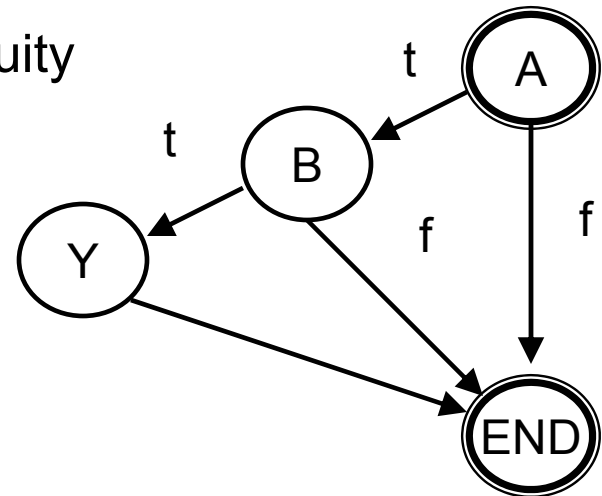
F1(F2):
if A then if B then Y



F1: if A then X



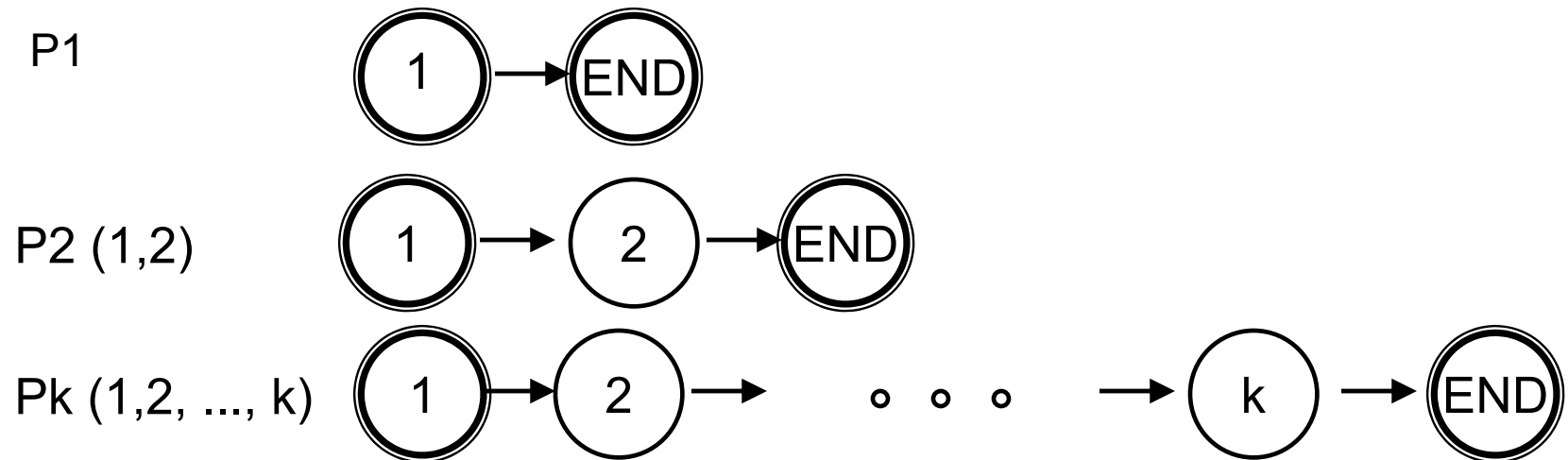
F2: if B then Y



Prime Flowgraph

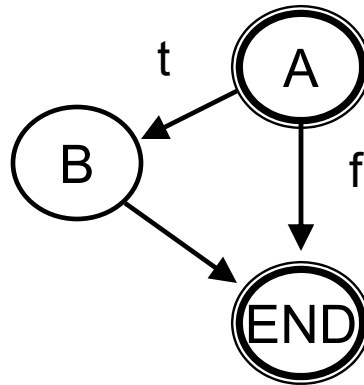
- Cannot be non-trivially decomposed by sequencing or nesting
- Different languages have different prime flowgraphs.
- Common ones:
 - P_k = Sequence of length K
 - D_0 = If ... Then
 - D_1 = If ... Then ... Else
 - D_2 = While ... Do
 - D_3 = Repeat ... Until
 - C_k = Case statement with K cases

Prime Flowgraphs P_k – A simple series of statements



Prime Flowgraphs

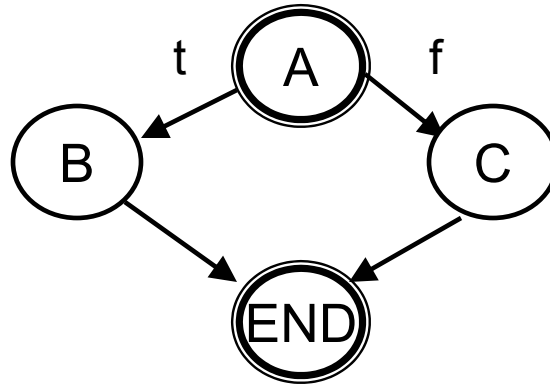
D0 If A then B



D0 (A,B)

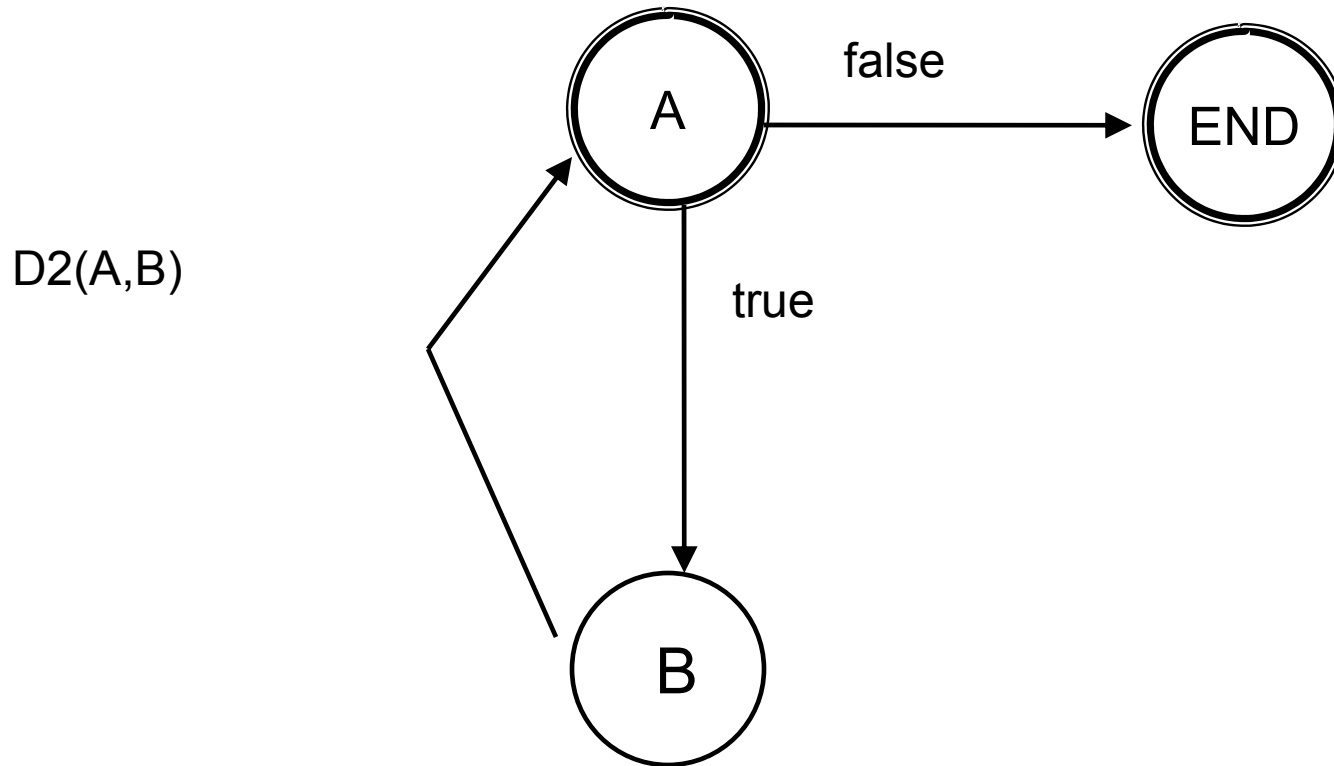
Prime Flowgraphs D1 If A then B else C

D1 (A,B,C)



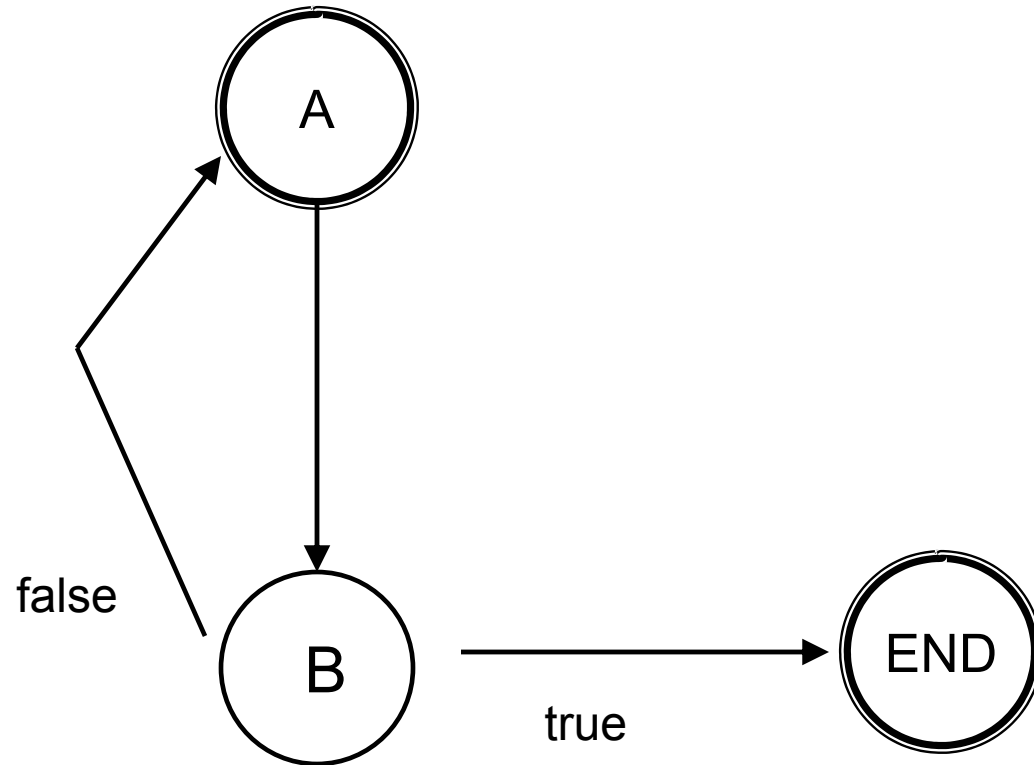
Prime Flowgraphs

D2 while A do B

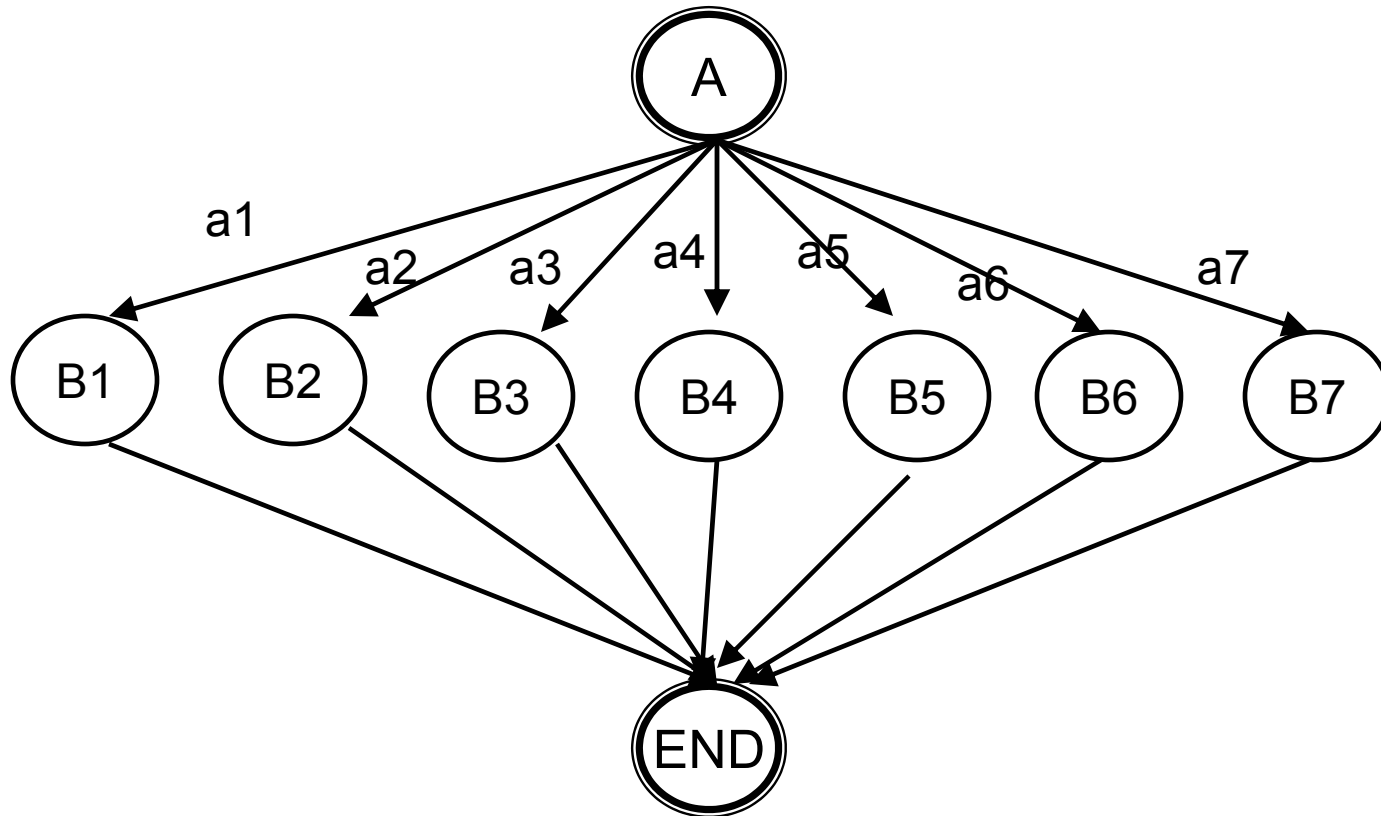


Prime Flowgraphs D3 repeat A until B

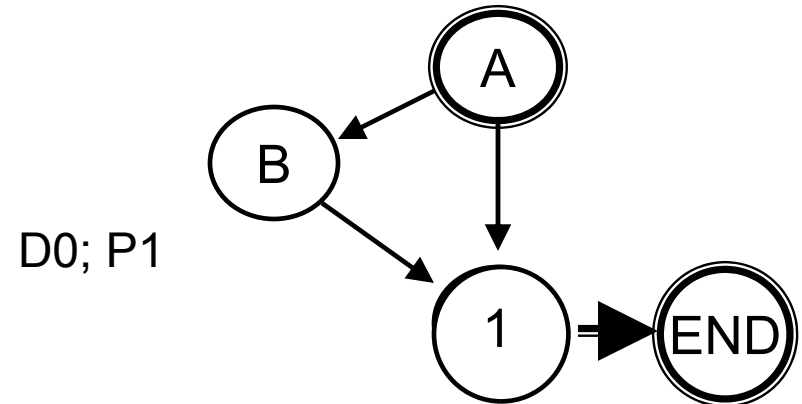
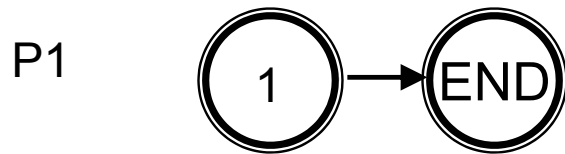
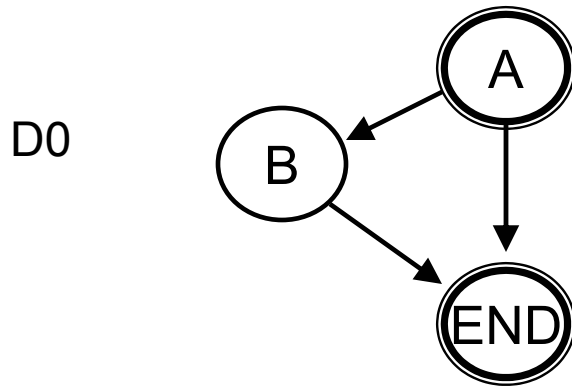
D3(A,B)



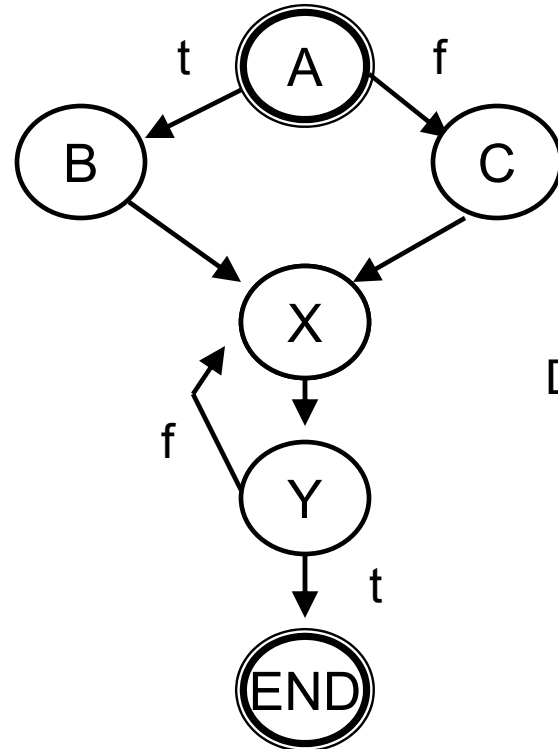
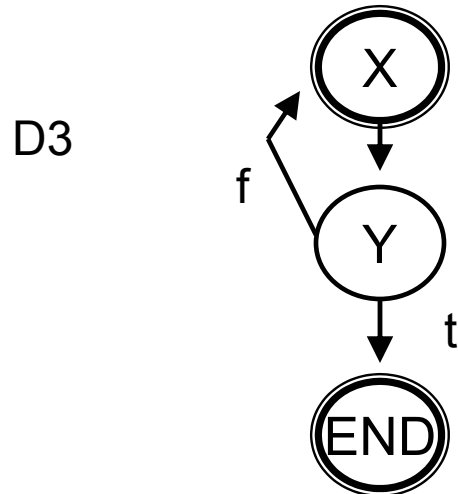
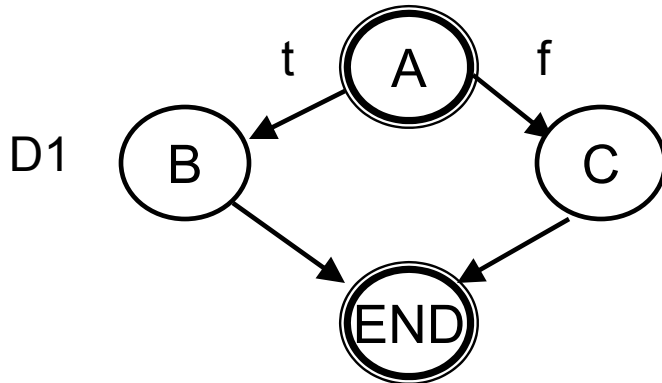
Prime Flowgraphs Ck Case statement



Sequence Practice

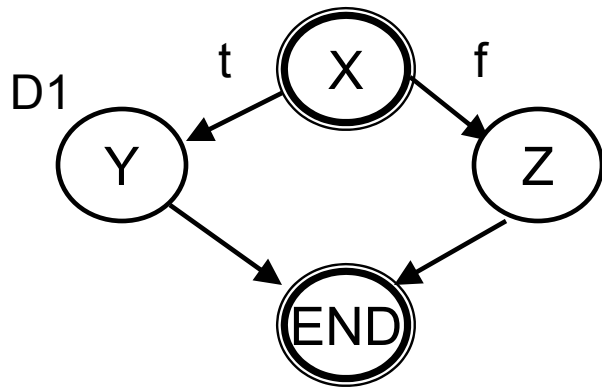
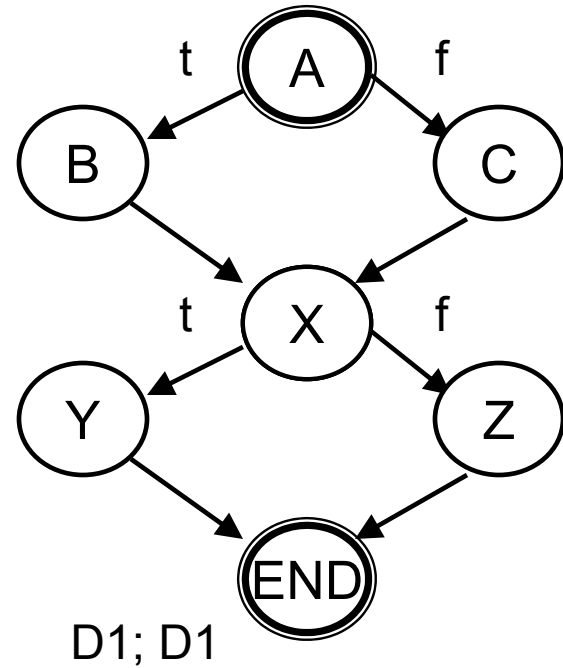
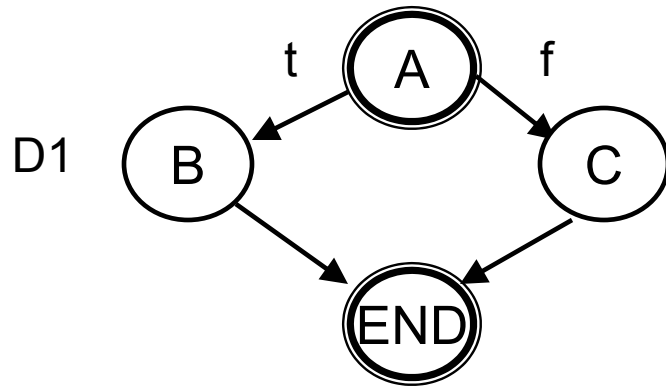


Sequence Practice

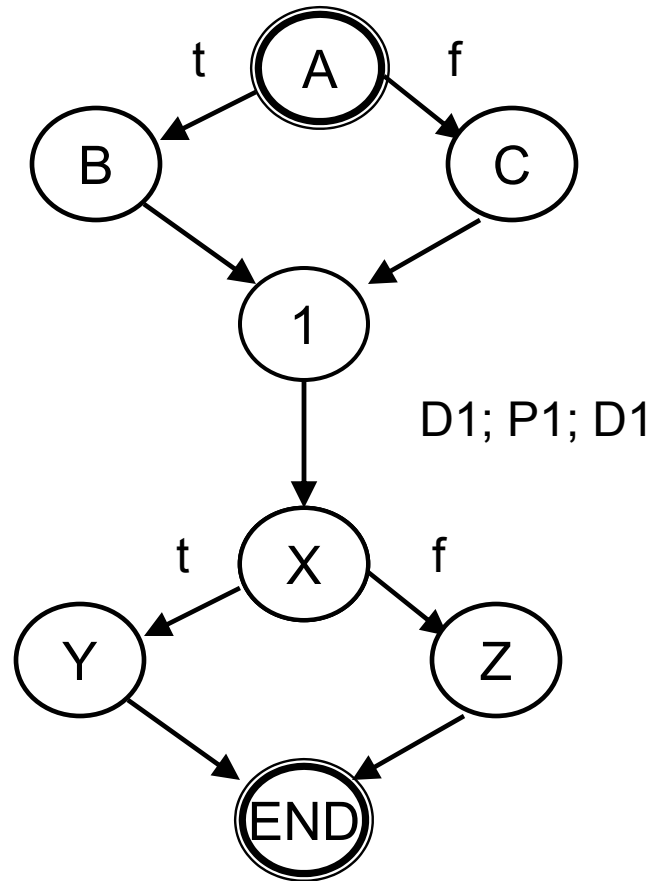
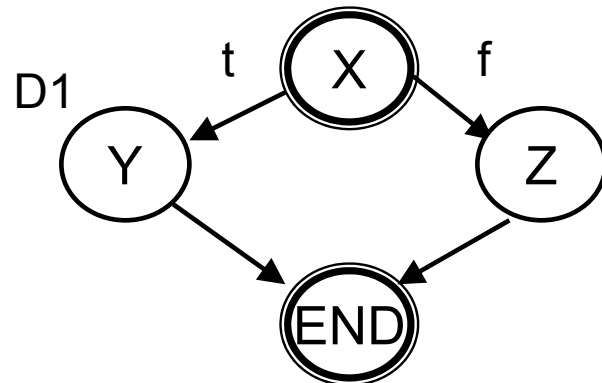
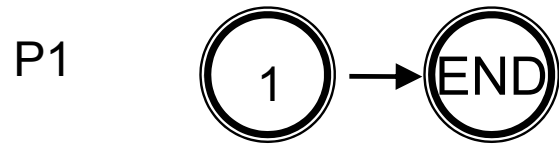
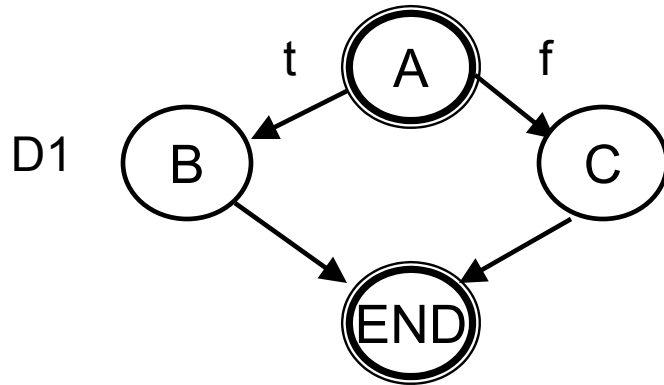


D1; D3

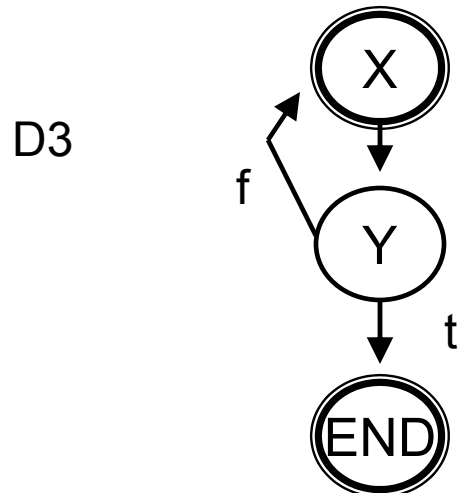
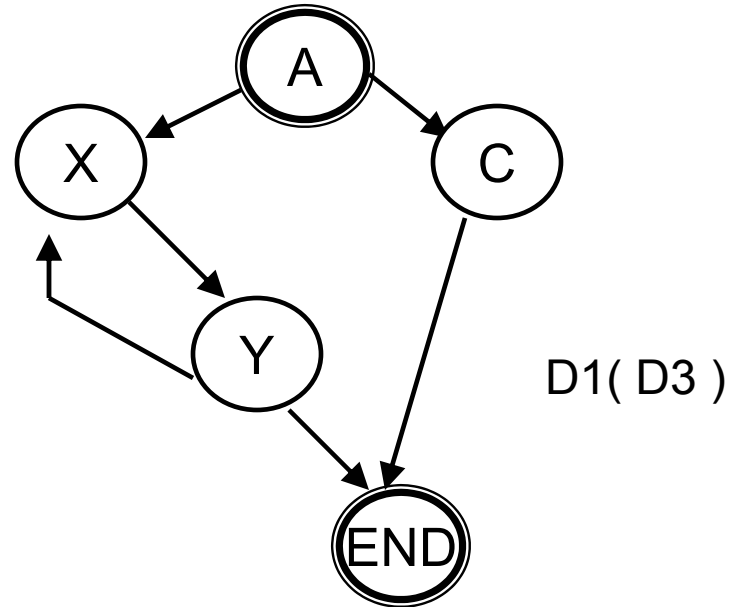
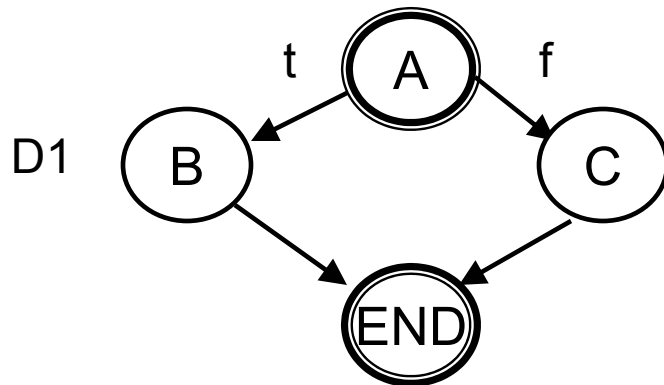
Sequence Practice



Sequence Practice



Nesting Practice

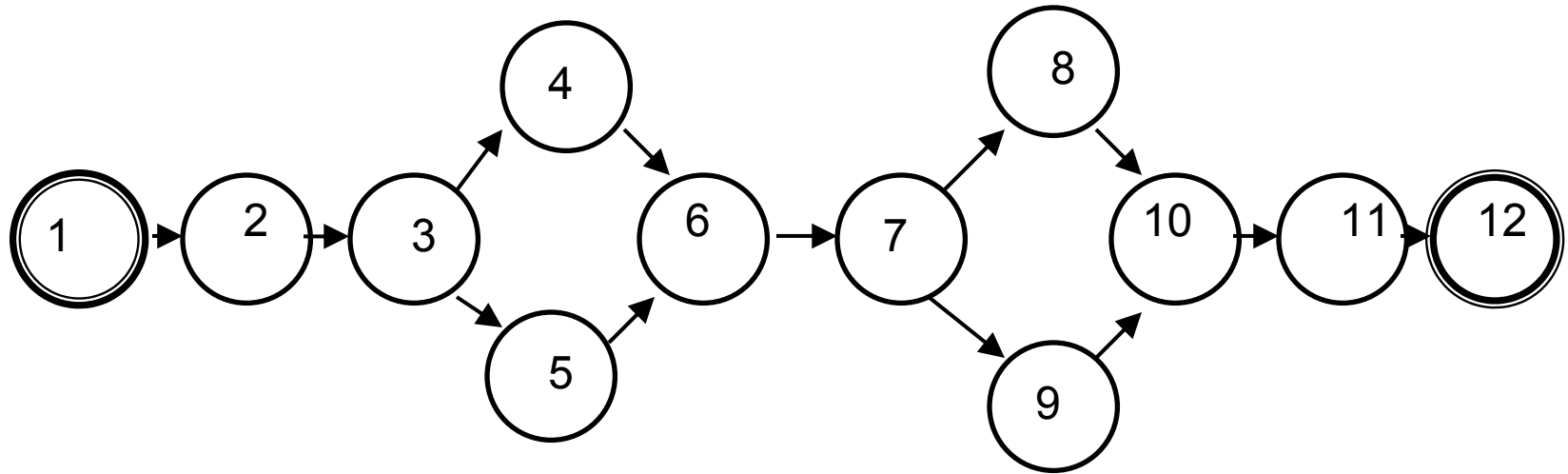


Composition of flowgraphs



- Functions can be modeled as directed graphs, and built up by composition of the basic flowgraphs
- Every flowgraph has a unique decomposition into a hierarchy of primes

Decompose this

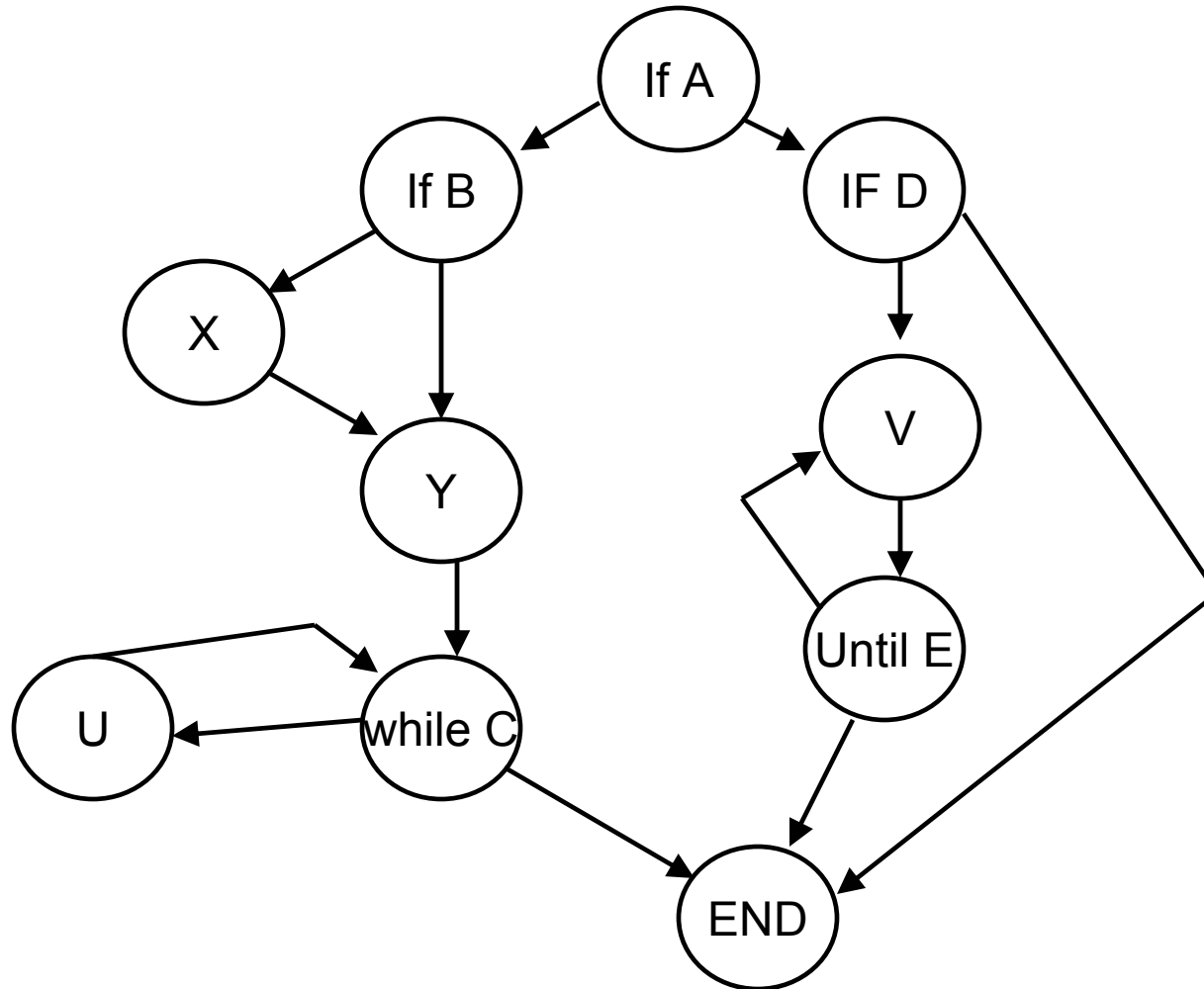


Graph this (From Fenton / Pfleeger)



```
If A
  then
    begin
      If B then do X;
      Y;
      while C do U
    end
  else
    if D
      then do
        repeat V until E
```

Graph this (continued)



Now,
decompose it
into a hierarchy
of primes

Structuredness

- For a family, S , of prime flowgraphs
- A family of graphs is S -structured (or are S -graphs) if it satisfies the following recursive rules:
 - Each member of S is S -structured
 - If $F1$ and $F2$ are S -Structured graphs then so are
 - $F1; F2$
 - $F1 (F2)$ wherever nesting of $F2$ onto $F1$ is defined
 - No flowgraph is an S -structured graph unless it can be shown to be generated by a finite number of applications of the above steps

NOTE: If $SD = \{P1, D0, D2\}$, the set of SD -graphs is the class of “ D -structured” or “structured” graphs. Every algorithm can be encoded as an SD -graph (Bohm & Jacopini)

Hierarchical measures – in general

- A measure M is a hierarchical measure if it can be defined on the set of S-graphs by specifying
 - $M(F)$ for each F in S (rule M1)
 - The sequencing function
 $M(F1;F2;...;Fk)$ (rule M2)
 - The nesting functions for
each F in S (rule M3)
- We can compute a hierarchical measure for a program once we know the rules, M1, M2 and M3 and the decomposition tree.
- (These slides are closely based on Fenton / Pfleeger)

Hierarchical measures: Nesting

- Primes (define as follows)
 - Depth of nesting of P1 is 0
 - Depth of nesting of any other prime is 1
 - $D(P1) = 0$
 - $D(F \text{ prime but } \neq P1) = 1$
- Sequence
 - Depth of nesting of sequence $F1, F2, \dots, Fk$ is maximum of the depth of nesting of the F_i 's.
 - $D(F1; \dots; Fk) = \text{Max}(D(F1), \dots, D(Fk))$
- Nesting
 - Depth of nesting of flowgraph $F(F1, \dots, Fk)$ is max of the depth of nesting of the F_i plus 1 b/c of the extra nesting level in F). So
 - $D(F(F1; \dots; Fk)) = 1 + \text{Max}(D(F1), \dots, D(Fk))$

Hierarchical measures: Length

- M1:
 - $V(P1) = 1$
 - $V(F) = N+1$, where N is number of procedure nodes in F
- M2:
 - $V(F1;F2;...;Fk) = \text{Sum} (V(Fi))$
- M3:
 - $V(F(F1,...,Fk)) = 1 + \text{Sum}(V(Fi))$ for each prime $Fi \triangleleft P1$
- Example: compute $v(D1((D0;P1;D2),D0(D3)))$

Hierarchical measures – number of nodes

- M1:
 - $n(F)$ = number of nodes in F, for each prime F
- M2:
 - $n(F_1; \dots ; F_k) = \sum n(F_i) - k + 1$
- M3:
 - $N(F(F_1, \dots , F_k)) = n(F) + \sum n(F_i) - 2K$ for each prime F
- Try it for
 - P2 (and decompose it to P1;P1)
 - D1 (D3, D1)

Hierarchical measures – number of edges

- M1:
 - $e(F)$ = number of edges in F , for each prime F
- M2:
 - $e(F_1; \dots ; F_k) = \sum e(F_i)$
- M3:
 - $e(F(F_1, \dots, F_k)) = e(F) + \sum e(F_i) - k$ for each prime F
- Try it for
 - P2 (and decompose it to P1;P1)
 - D1 (D3, D1)

Hierarchical measures – cyclomatic complexity

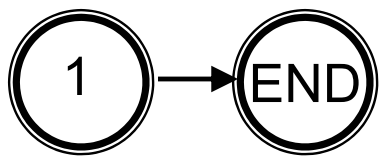
- Cyclomatic complexity (F) = $e(F) - n(F) + 2$
 - $e(F)$ = number of arcs in F
 - $n(F)$ = number of nodes in F
- This is the number of linearly independent paths (*aka basis paths*) through F

Hierarchical measures – cyclomatic complexity

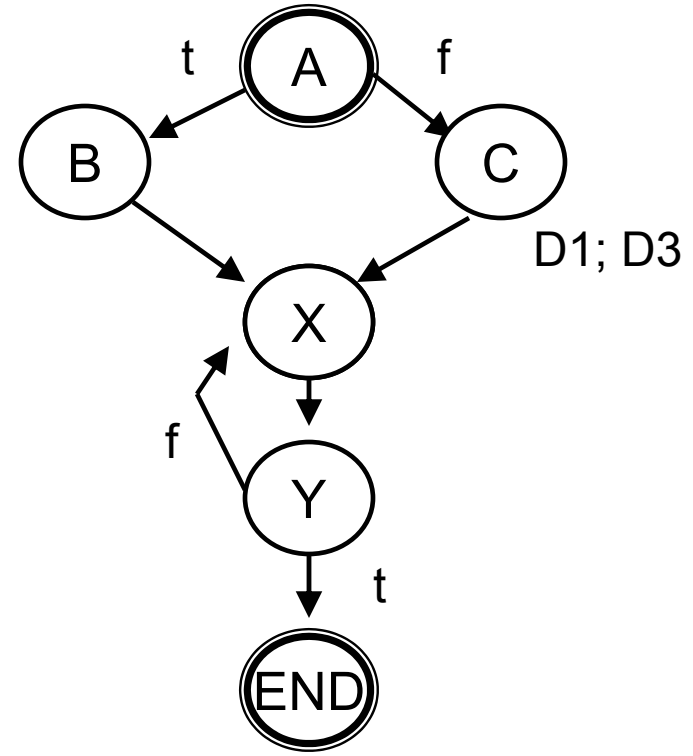
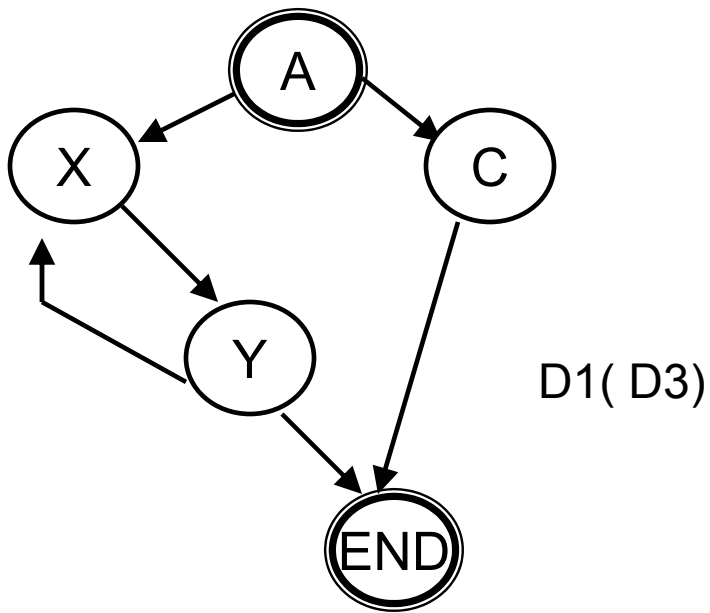
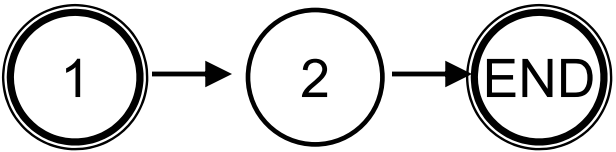
- Cyclomatic complexity $(F) = e(F) - n(F) + 2$
 - $e(F)$ = number of arcs in F
 - $n(F)$ = number of nodes in F
- This is the number of linearly independent paths through F
- M1:
 - $c(F) = 1 + d$ where d is number of predicates in F , prime F
- M2:
 - $c(F_1; \dots; F_k) = \sum c(F_i) - k + 1$, for each prime F_i
- M3:
 - $c(F(F_1, \dots, F_k)) = c(F) + \sum c(F_i) - k$ for each prime F

Cyclomatic complexity, simple examples

P1



P2 (1,2)



Test coverage metrics – basis path coverage



- McCabe's metric counts the number of basis paths through the program, essentially the number of linearly independent paths through the program. If you design your tests to hit every basis path, you will cover every statement and every branch in the program.

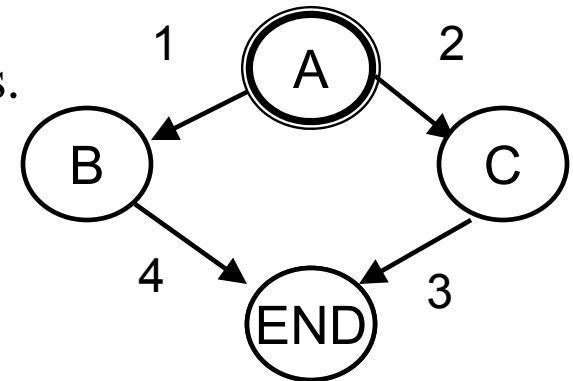
Graph theory underlying basis paths (definitions)

- **Strongly connected graph**: for any node, x , there is a path from x to y and a path from y to x .
- A **circuit** is a path that begins and ends at the same node.
- A **cycle** is a circuit with no node (other than the starting node) included more than once.
- A path, P , is a **linear combination of paths**, P_1, \dots, P_n if there are integers, A_i such that $P = \sum A_i * P_i$.
- A set of paths is **linearly independent** if no path in the set is a linear combination of the others.

Basis paths

- Notation:

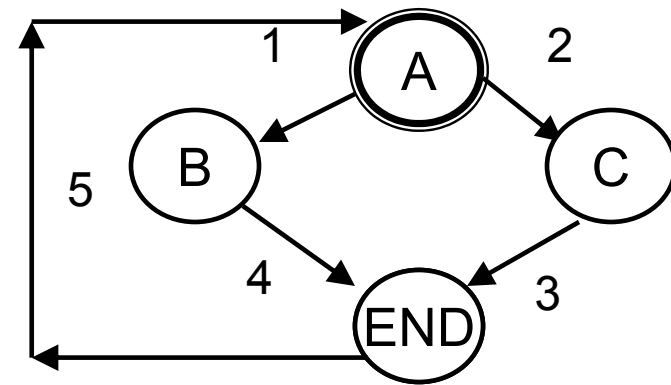
- Rather than labeling nodes, lets label edges.



- We can describe a path as an n-tuple, such as $\langle 2, 3 \rangle$ or $\langle 1, 4 \rangle$
- We can create a path vector that shows the number of times each path is traversed
 - $\langle 2, 3 \rangle = [0 \ 1 \ 1 \ 0]$.
 - $\langle 1, 4 \rangle = [1 \ 0 \ 0 \ 1]$.
 - $\langle 1, 2, 3, 4 \rangle = [1 \ 1 \ 1 \ 1]$ but this is an **infeasible** path.
 - $(1 \ 1 \ 1 \ 1) = [1 \ 0 \ 0 \ 1] + [0 \ 1 \ 1 \ 0]$ (this is basic matrix algebra)
- Cyclomatic complexity = $e(F) - n(F) + 2 = 4 - 4 + 2 =$ size of basis set

Basis paths

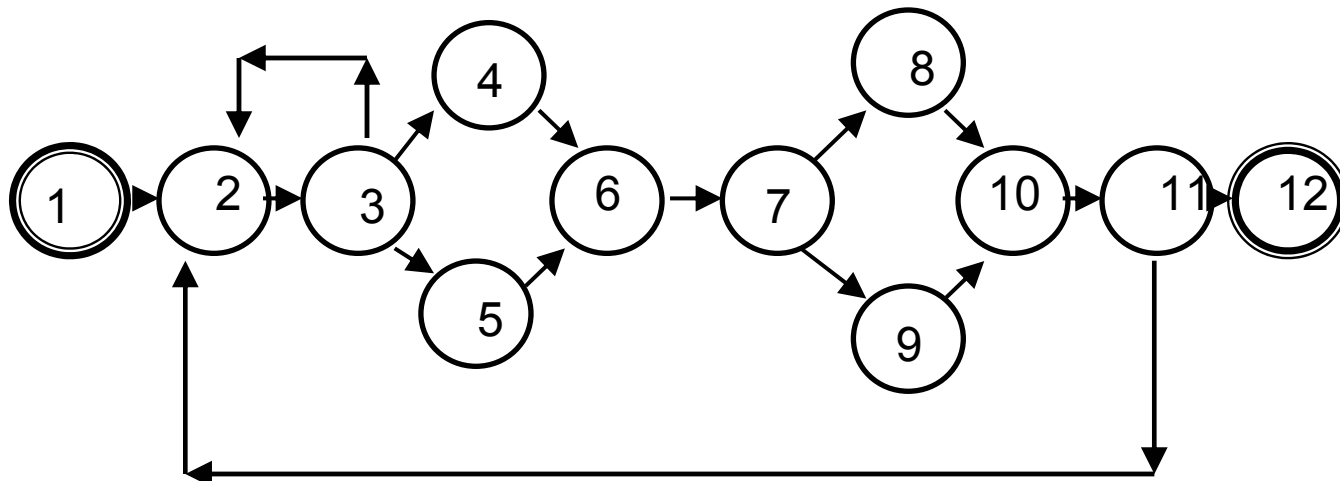
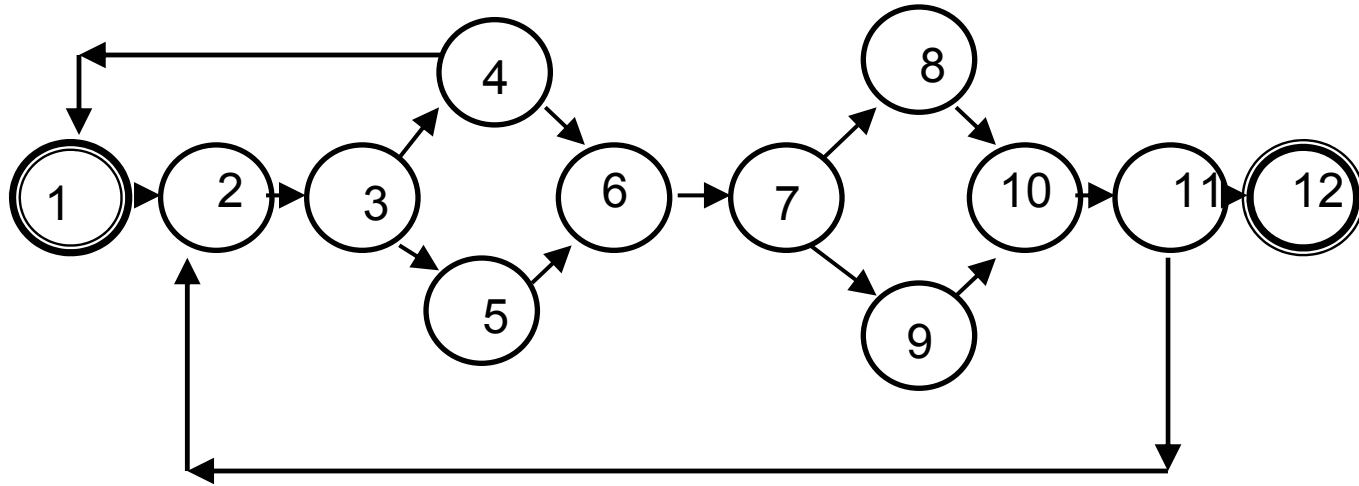
- To create the basis set of cycles, we can turn a directed graph into a strongly connected graph by drawing an arc from the end to the start.
- $[1\ 0\ 0\ 1\ 1]$ is a circuit
- $[0\ 1\ 1\ 0\ 1]$ is a circuit
- $[1\ 1\ 1\ 1\ 2]$ is a circuit (*what is the path?*)
- $[1\ 0\ 0\ 1\ 1]$ and $[0\ 1\ 1\ 0\ 1]$ are cycles but $[1\ 1\ 1\ 1\ 2]$ is not.
- $[1\ 1\ 1\ 1\ 2]$ is a linear combination of $[1\ 0\ 0\ 1\ 1]$ and $[0\ 1\ 1\ 0\ 1]$
- Any other path that you could actually take through the graph is a linear combination of $[1\ 0\ 0\ 1\ 1]$ and $[0\ 1\ 1\ 0\ 1]$
- This is a **basis set** of cycles



Basis paths

- Once we know the basis set of cycles, we eliminate the fictitious branch (from stop to start), reducing the vectors by a column:
 - $[1\ 0\ 0\ 1]$ and $[0\ 1\ 1\ 0]$ is a basis set of linearly independent paths.
 - Basis sets (of cycles or flowgraph paths) are not unique
 - Question: Aren't $[1\ 1\ 0\ 0]$ and $[0\ 0\ 1\ 1]$ linearly independent of the basis paths? Why aren't they usable?

Reality Check: Which is more complex?



Two programs with the same McCabe complexity number can have very different complexity.

A Different Reality Check: Semantic Complexity

- Consider these variable names:
 - **BLUE**
 - **RED**
- Structure complexity metrics are not affected by strange variable names, inaccurate comments, strange data types, or anything that conveys meaning of the program rather than branching structure of the program.

Code coverage

- Coverage measures the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:
 - *100% line coverage* means that you tested for every bug that can be revealed by simple execution of a line of code.
 - *100% branch coverage* means you will find every error that can be revealed by testing each branch.
 - *100% coverage* should mean that you tested for every possible error. This is obviously impossible.


Benefits of coverage

Before I attack coverage measures, let me acknowledge that they are often useful.

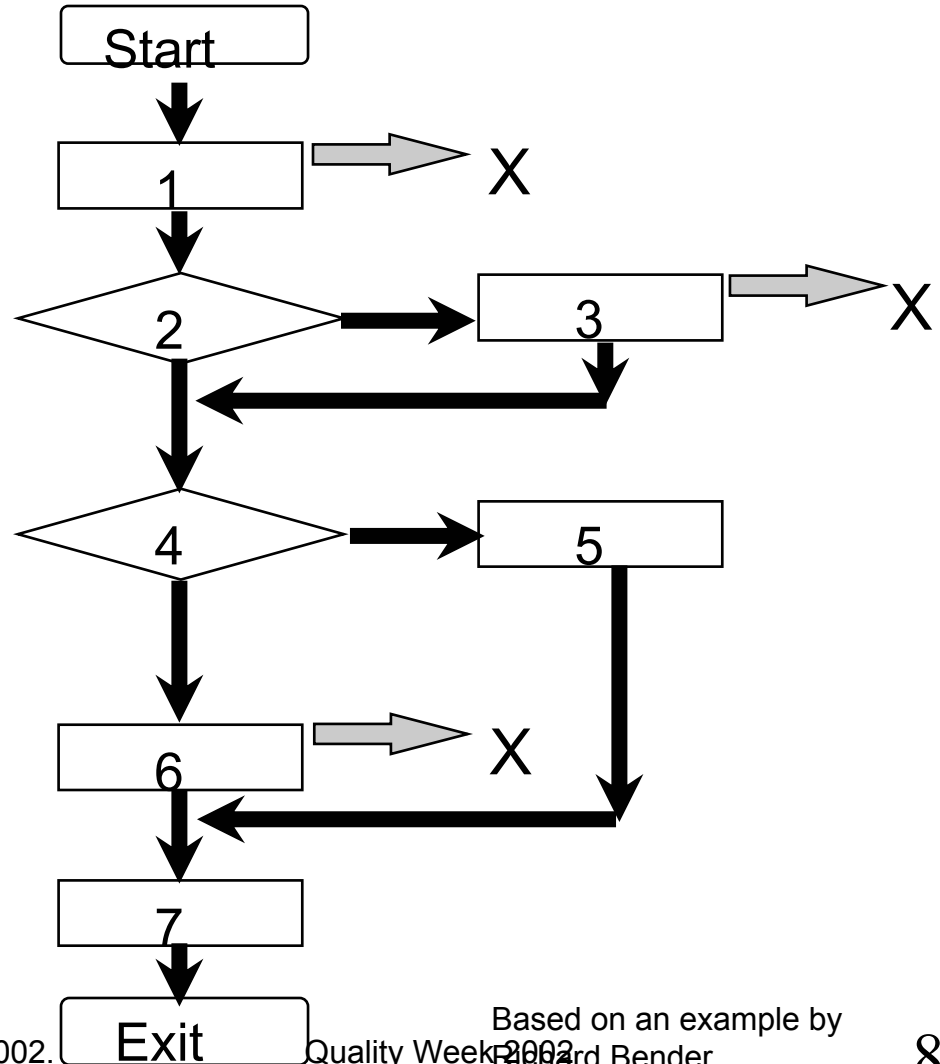
- Many projects achieve low statement coverage, as little as 2% at one well-known publisher that had done (as measured by tester-hours) extensive testing and test automation. The results from checking statement coverage caused a complete rethinking of the company's automation strategy.
- Coverage tools can point to unreachable code or to code that is active but persistently untested.

Coverage tools can provide powerful diagnostic information about the testing strategy, even if they are terrible measures of the extent of testing.

Statement / Branch Coverage and Data Flows

 X
 means this routine
 changes variable X

1 (x) 2 3 (x) 4 5 7
 1 (x) 2 4 6 (x) 7
*Now we have 100% branch
 coverage, but where is 1(x) 7?*
 1 (x) 2 4 5 7



Example: Statement/Branch Coverage

- Attribute** ➤ Extent of testing – *How much of the product have we tested?*
- Instrument** ➤ Count statements and branches tested
- Mechanism** ➤ If we do more testing and find more bugs, does that mean that our line count will increase? *Not necessarily. Example—configuration tests.*
- Side Effect** ➤ If we design our tests to make sure we hit more lines, does that mean we'll have done more extensive testing? *Maybe in a trivial sense, but we can achieve this with weaker tests that find fewer bugs.*
- Purpose** ➤ Not specified
- Scope** ➤ Not specified

Data Flows

- A set-use pair is a dataflow
- A set-use pair with no intervening set is a first-order dataflow.
 - In the example, set the value of X in lines 1, 3, and 6.
 - Path 1(set x) 2 3(set x) 4 5 7 (print x) has
 - a first-order dataflow from line 3 to line 7 and
 - a second-order dataflow from line 1 (through line 3 where x is reset) to line 7
- 100% dataflow coverage (in testing) usually means covering all the first-order dataflows.

Statement / Branch Coverage Just Test the Flowchart

You're not testing:

- » data flow
- » tables that determine control flow in table-driven code
- » side effects of interrupts, or interaction with background tasks
- » special values, such as boundary cases. These might or might not be tested.
- » unexpected values (e.g. divide by zero)
- » user interface errors
- » timing-related bugs
- » compliance with contracts, regulations, or other requirements
- » configuration/compatibility failures
- » volume, load, hardware faults

If we use “coverage”?

- If we improve testing by 20%, does this result in a 20% increase in “coverage”? Does it necessarily result in ANY increase in “coverage”?
- If we increase “coverage” by 20%, does this mean that there was a 20% improvement in the testing?
- If we achieve 100% “coverage”, do we really think we’ve found all the bugs?

Side effects and “coverage”

- Without a mechanism that ties changes in the attribute being measured to changes in the reading we get from the instrument, we have a “measure” that is ripe for abuse.
- ***There are hundreds of different things we can count and thus hundreds of potential coverage measures. Statements, branches, and dataflows are just a few examples.***
- ***The choice of any one or few of them as your “measure” will lead you to optimize on some dimensions and ignore the others.***
- ***This is classic partial supervision, and we should expect the predicted distortion or disfunction as a result.***
- As Brian Marick has often pointed out, in companies that rely on “coverage” as a measure of testing thoroughness, the coverage number will go up, but the quality of testing might well go down.

Statement / Branch Coverage and Data Flows

- How many basis paths are there in this program?

1	2	3	4	5	6	7	Exit
1	1	1	1	1	0	1	1
1	1	0	1	0	1	1	1

- Can we achieve complete basis path coverage without ever hitting the critical data flow?

Basis paths

- By the way, which of these columns is variable?

1	2	3	4	5	6	7	Exit
1	1	1	1	1	0	1	1
1	1	0	1	0	1	1	1
1	1	1	1	0	1	1	1

Basis paths

- So we can reduce the big chart to the simpler small chart. It is much easier to prove that the vectors form a basis path in the simplified matrix.

1	2	3	4	5	6	7	Exit
X	X	1	X	1	0	X	X
X	X	0	X	0	1	X	X
X	X	1	X	0	1	X	X

3	5	6
1	1	0
0	0	1
1	0	1

Goal-Question-Metrics Approach

- To decide what to measure, we should first know why we care about the answer. Given a goal for the measurement, we can work forward to collect information that can help us meet that goal.
- Basic approach
 - Set the goal
 - Identify questions that would give you information that you need in order to meet the goal
 - Determine whether there are (or whether you can create) metrics that can help you answer those questions.

GQM Template for Defining a Goal

- Questions usually look for information like:
 - **Purpose:** TO (characterize, evaluate, predict, motivate, etc.) THE (process, product, model, metric, etc.) IN ORDER TO (understand, assess, manage, engineer, learn, improve, test, etc.)
 - **Perspective:** EXAMINE THE (cost, effectiveness, correctness, defects, changes, product metrics, reliability, etc.) FROM THE POINT OF VIEW OF (the programmer, manager, customer, corporate perspective, etc.)
 - **Environment:** The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc.

Adapted from Basili

Example from Rosenberg



- To PREDICT the SCHEDULE in order to MANAGE it.
 - What are some relevant questions?
 - Which ones might be answerable with metrics?
 - What assumptions or preconditions or challenges are associated with those questions or metrics?

Another example

- To EVALUATE the COSTS AND BENEFITS of CODE INSPECTIONS in order to DETERMINE WHETHER TO CONTINUE THIS PROCESS.
- To analyze this, we have to break it down. What question(s) would we ask about each of these?
 - Evaluate
 - Costs
 - Benefits
 - Code inspections
 - Determine
 - Continue
 - This process

GQM Done Poorly or Well

- The goal has to be one that can be achieved via measurement.
- **Evaluate the questions.** If you could answer them, would you achieve your goal? If not, what other information would you need?
- **Evaluate the metrics.** If you collected them, would they provide you all of the information you need to answer your questions?
- ***GQM in practice is often a rationalization to collect the same old metrics.***
- **Evaluate the metrics.** Apply the 10-factor analysis (or some other careful analysis of validity) to them.

Multi-Dimensional Measurement

- The idea of multi-dimensional measurement is to put together a pattern of information that, collectively, gives a more accurate picture.
- COCOMO is a leading example of this approach. See <http://www.jsc.nasa.gov/bu2/COCOMO.html> and <http://sunset.usc.edu/research/COCOMOII/Docs/stc.pdf>
- Balanced scorecards are a general scheme of this type. (Kaplan & Norton, *The Balanced Scorecard: Translating Strategy into Action*). Rather than reporting a single not-very-representative measure, use:
 - a small number (maybe 5 - 10) of different measures,
 - all of them meaningful to you,
 - none of them perfect,
 - all of them substantially different from each other,
 - selected in a way that distortion caused by attempting to optimize on a single measure will be reflected as a negative in at least one other measure.

For example: Treat “Extent” as a Multidimensional Problem

- We developed the 8 aspects (or dimensions) of “extent of testing” by looking at the types of measures of extent of testing that we were reporting.
- Consider using a combination measure that looks at the 8 dimensions
 - product coverage
 - effort
 - Obstacles
 - quality of testing
 - plan / agreement
 - results
 - risks
 - project history

Project Report / Component Map

Status report used by Elizabeth Hendrickson

Page 1 --- Issues that need management attention

Page 2 --- Component map

Page 3 --- Bug statistics

Component	Test Type	Tester	Total Tests Planned / Created	Tests Passed / Failed / Blocked	Time Budget	Time Spent	Projected for Next Build	Notes

We see in this report:

- Progress against plan
- Effort
- Obstacles / Risks
- Results

Bach's Dashboard

Testing Dashboard				Updated 11/1/00	Build 32
Area	Effort	Coverage Planned	Coverage Achieved	Quality	Comments
File/edit	High	High	Low	☹	1345, 1410
View	Low	Med	Med	☺	
Insert	Blocked	Med	Low	☹	1621

We see coverage of areas, progress against plan, current effort, key results and risks, and obstacles.

One approach: Balanced scorecard



- **For 101 examples of possible coverage measures, that might be suitable for balancing, see “Software Negligence and Testing Coverage” at www.kaner.com. These are merged in a list with over 100 additional indicators of extent of testing in the paper, “Measurement Issues & Software Testing”, which is included in the proceedings.**

Suggested Lessons

- Simple charts can carry a lot of useful information and lead you to a lot of useful questions.
- Report multidimensional patterns, rather than single measures or a few measures along the same line.
- Think carefully about the potential side effects of your measures. Robert Austin criticizes the balanced scorecard approach because it can, and often does, still lead to abuse, especially if the measures don't balance each other out.
- Listen critically to reports (case studies) of success with simple metrics. If you can't see the data and don't know how the data were actually collected, you might well be looking at results that were sanitized by working staff (as a side effect of the imposition of the measurement process).