Intro to Software Testing

chapter 8.3

Applying Logic Coverage

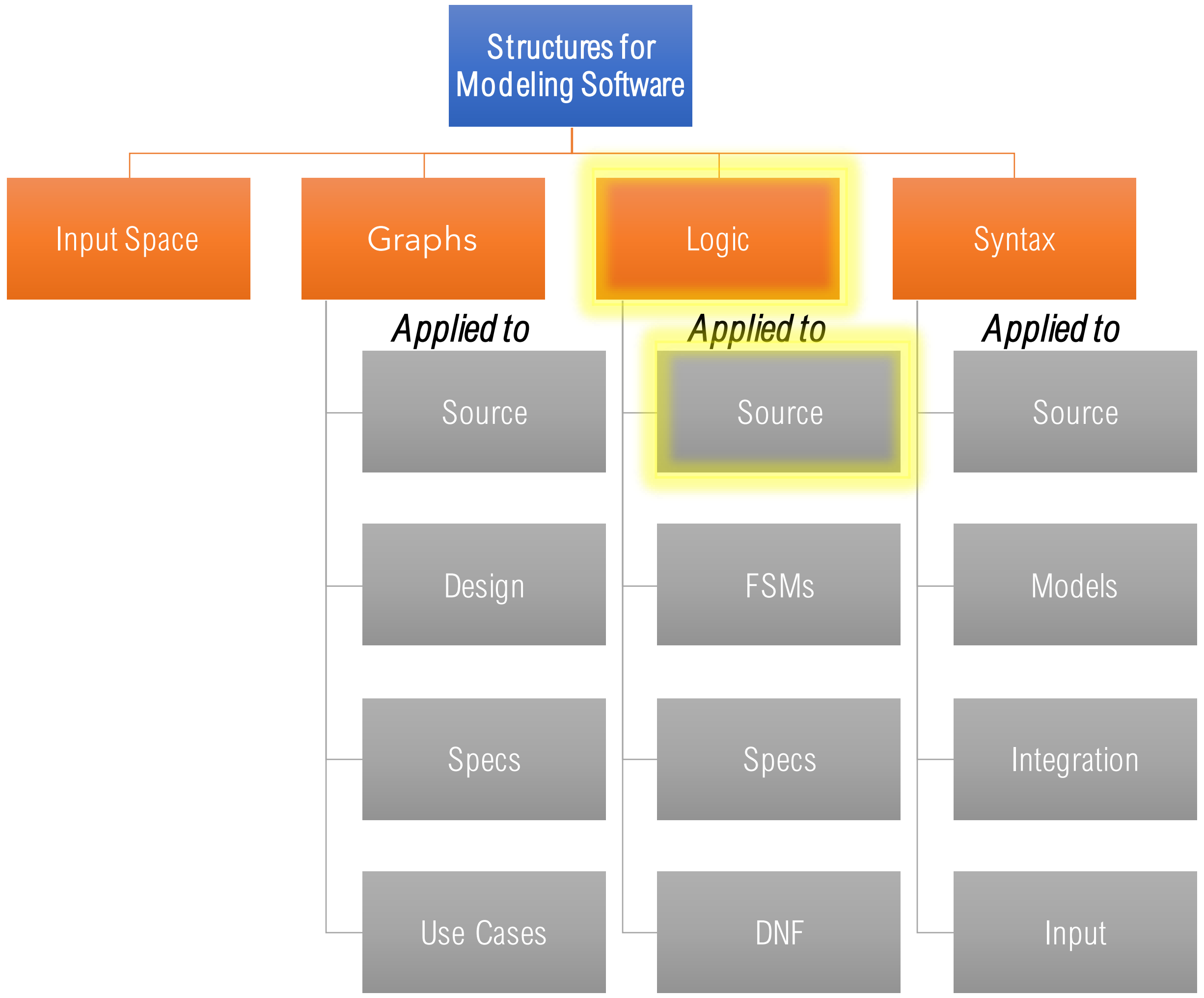
Dr. Brittany Johnson-Matthews

(Dr. B for short)

https://go.gmu.edu/SWE637

Adapted from slides by Jeff Offutt and Bob Kurtz

Logic Coverage

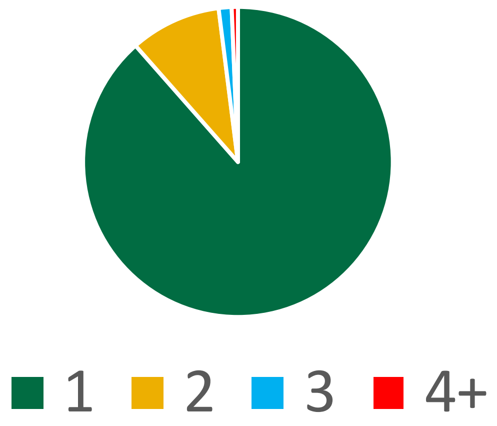


Logic Expressions from Source

Predicates are derived from *decision* statements

In programs, most predicates have *fewer than four* clauses

When a predicate has only one clause, all criteria (CoC, ACC, ICC, CC) collapse to *predicate coverage* (PC)



Logic Expressions from Source

In practice, applying logic criteria to source code is hard because of *reachability* and *controllability*

- **Reachability**: in order to apply criteria to a predicate in a source code statement, we must be able to execute that statement
- **Controllability**: in order to test the desired logic combinations, we must be able to find program input values that indirectly assign the desired logic values to the variables in the predicate *at the time that execution reaches the predicate*

Thermostat Example

```
public class Thermostat
{
   private int curTemp;          // current temperature reading
   private int thresholdDiff;    // temp difference until we turn heater on
   private int timeSinceLastRun; // time since heater stopped
   private int minLag;           // how long I need to wait
   private boolean override;     // has user overridden the program
   private int overTemp;         // overriding temperature
   private int runTime;          // output of turnHea
   private boolean heaterOn;     // output of turnHe
   private Period period;        // morning, day, ev
   private DayType day;          // week day or week

   // Decide whether to turn the heater on, and for
   public boolean turnHeaterOn (ProgrammedSettings p
   {
      int dTemp = pSet.getSetting (period, day);

      if (((curTemp < dTemp - thresholdDiff) ||
           (override && curTemp < overTemp - thresholdDiff)) &&
           (timeSinceLastRun > minLag))
      {  // Turn on the heater
         // How long? Assume 1 minute per degree (Fahrenheit)
         int timeNeeded = curTemp - dTemp;
         if (override)
            timeNeeded = curTemp - overTemp;
         setRunTime (timeNeeded);
         setHeaterOn (true);
         return (true);
      }
      else
      {
         setHeaterOn (false);
         return (false);
      }
   } // End turnHeaterOn
   . . .
```

Consider this predicate…

See the textbook website for the complete source code listing

https://cs.gmu.edu/~offutt/softwaretest/java/Thermostat.java
https://cs.gmu.edu/~offutt/softwaretest/java/ProgrammedSettings.java
https://cs.gmu.edu/~offutt/softwaretest/java/DayType.java
https://cs.gmu.edu/~offutt/softwaretest/java/Period.java

Thermostat Example

```
if (((curTemp < dTemp - thresholdDiff) ||
     (override && curTemp < overTemp - thresholdDiff)) &&
    (timeSinceLastRun > minLag))
```

Simplify the representation

- a = curTemp < dTemp – thresholdDiff
- b = override
- c = curTemp < overtemp – thresholdDiff
- d = timeSinceLastRun > minLag

$p = (a \vee (b \wedge c)) \wedge d$

or $\boldsymbol{f = ad + bcd}$ *if you prefer DNF*

Predicate Controllability

$p = (a \vee (b \wedge c)) \wedge d$

a = curTemp < dTemp – thresholdDiff

```
int dTemp = pSet.getSetting (period, day);
```

Variable **`pSet`** is passed as an argument

Configure **`pSet`** with test code like

```
pSet.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
```

Control the evaluation of **`dTemp`** with test code like

```
setPeriod(Period.MORNING);
setDay(DayType.WEEKDAY);
```

Predicate Coverage: TRUE

$p = (a \lor (b \land c)) \land d = TRUE$

We can force the predicate to be true with (among other possibilities)

$a = TRUE$, $b = TRUE$, $c = TRUE$, $d = TRUE$

That means that we need

curTemp < dTemp – thresholdDiff == true

override == true

curTemp < overtemp – thresholdDiff == true

timeSinceLastRun > minLag == true

How can we do that?

Predicate Coverage: TRUE

a = curTemp < dTemp – thresholdDiff == true

```
// Control curTemp
thermo.setCurrentTemp(62);
// Control dTemp
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
thermo.setPeriod(Period.MORNING);
thermo.setDay(DayType.WEEKDAY);
// Control thresholdDiff
thermo.setThresholdDiff = 5;
```

b = override == true

```
// Control override
thermo.setOverride(true);
```

c = curTemp < overtemp – thresholdDiff == true

```
// Control overtemp (other variables are already set)
thermo.setOverTemp(75);
```

d = timeSinceLastRun > minLag == true

```
// Control timeSinceLastRun
thermo.setTimeSinceLastRun(15);
// Control minLag
thermo.setMinLag(10);
```

CACC Example

DEFINITION

Correlated Active Clause Coverage (CACC) – For each *p* in *P* and each major clause c_i in C_p, choose minor clauses c_j (j!=i) such that c_i determines *p*. *TR* has two requirements for c_i: c_i evaluates to true and c_i evaluates to false. The values chosen for minor clauses c_j must cause p to be true for one value of major clause c_i and false for the other value of c_i.

Let’s look at the CACC requirements for clause ***a***

CACC Example

Calculate when ***a*** determines ***p***

$$
\begin{aligned}
p_a &= p_{a=true} \oplus p_{a=false} \\
&= (\mathrm{T} \vee (b \wedge c)) \wedge d \oplus (\mathrm{F} \vee (b \wedge c)) \wedge d \\
&= \mathrm{T} \wedge d \oplus (b \wedge c) \wedge d \\
&= d \oplus (b \wedge c) \wedge d \\
&= \neg(b \wedge c) \wedge d \\
&= (\neg b \vee \neg c) \wedge d
\end{aligned}
$$

CACC Example

Repeat for ***b***, ***c***, and ***d***

$p_a = (\neg b \lor \neg c) \land d$

We have options here: { TTFT, FTFT } or { TFTT, FFTT } or { TFFT, FFFT } all satisfy p_a, but we'll choose { TTFT, FTFT } because at a glance we think it's a feasible set of tests (in fact I cheated and looked ahead at the test requirements for the other variables)

Det.	a	b	c	d
p_a	T	*T*	*F*	T
	F	*T*	*F*	T
p_c				
p_d				

CACC Example

Repeat for ***b***, ***c***, and ***d***

$p_a = (\neg b \vee \neg c) \wedge d$

$p_b = \neg a \wedge c \wedge d$

Det.	a	b	c	d
p_a	**T**	*T*	*F*	T
	F	*T*	*F*	T
p_b	F	**T**	T	T
	F	**F**	T	T
p_c				
p_d				

CACC Example

Repeat for ***b***, ***c***, and ***d***

$p_a = (\neg b \lor \neg c) \land d$

$p_b = \neg a \land c \land d$

$p_c = \neg a \land b \land d$

Det.	a	b	c	d
p_a	**T**	*T*	*F*	T
	F	*T*	*F*	T
p_b	F	**T**	T	T
	F	**F**	T	T
p_c	F	T	**T**	T
	F	T	**F**	T
p_d				

CACC Example

Repeat for ***b***, ***c***, and ***d***

$p_a = (\neg b \vee \neg c) \wedge d$

$p_b = \neg a \wedge c \wedge d$

$p_c = \neg a \wedge b \wedge d$

$p_d = a \vee (b \wedge c)$

Det.	a	b	c	d
p_a	**T**	*T*	*F*	T
	F	*T*	*F*	T
p_b	F	**T**	T	T
	F	**F**	T	T
p_c	F	T	**T**	T
	F	T	**F**	T
p_d	*F*	*T*	*T*	**T**
	F	*T*	*T*	**F**

CACC Example

Repeat for ***b***, ***c***, and ***d***

$p_a = (\neg b \lor \neg c) \land d$

$p_b = \neg a \land c \land d$

$p_c = \neg a \land b \land d$

$p_d = a \lor (b \land c)$

Det.	a	b	c	d
p_a	T	*T*	*F*	T
	F	*T*	*F*	T
p_b	F	T	T	T
	F	F	T	T
p_c	F	T	T	T
	F	T	F	T
p_d	*F*	*T*	*T*	T
	F	*T*	*T*	F

We have options here: { FTTT, FTTF } or { TFFT, TFFF } or { TTTT, TTTF } all satisfy p_d, but we can see that choosing { FTTT, FTTF } lets us re-use a previous test case for efficiency

CACC Example

Repeat for ***b***, ***c***, and ***d***

$p_a = (\neg b \lor \neg c) \land d$

$p_b = \neg a \land c \land d$

$p_c = \neg a \land b \land d$

$p_d = a \lor (b \land c)$

Det.	a	b	c	d
p_a	T	*T*	*F*	T
	F	*T*	*F*	T
p_b	F	T	T	T
	F	F	T	T
p_c	~~F~~	~~T~~	~~T~~	~~T~~
	~~F~~	~~T~~	~~F~~	~~T~~
p_d	~~*F*~~	~~*T*~~	~~*T*~~	~~T~~
	F	*T*	*T*	F

Five tests needed for CACC

CACC Example: T/F Values

a = curTemp < dTemp – thresholdDiff

b = override

c = curTemp < overtemp – thresholdDiff

d = timeSinceLastRun > minLag

Var.	T/F	curTemp	dTemp	threshold Diff	override	over Temp	timeSince LastRun	minLag
a	T	62	70	5	--	--	--	--
a	F	68	70	5	--	--	--	--
b	T	--	--	--	T	--	--	--
b	F	--	--	--	F	--	--	--
c	T	62	--	5	--	72	--	--
c	F	66	--	5	--	67	--	--
d	T	--	--	--	--	--	12	10
d	F	--	--	--	--	--	8	10

CACC Example: Test 1

Test case: TTFT

```
// Set a=true
thermo.setCurrentTemp(62);
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
thermo.setPeriod(Period.MORNING);
thermo.setDay(DayType.WEEKDAY);
thermo.setThresholdDiff = 5;
// Set b=true
thermo.setOverride(true);
// Set c=false
thermo.setOvertemp(67);
// Set d=true
thermo.setTimeSinceLastRun(12);
thermo.setMinLag(10);
```

CACC Example: Test 2

Test case: FTFT

```
// Set a=false
thermo.setCurrentTemp(68);
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
thermo.setPeriod(Period.MORNING);
thermo.setDay(DayType.WEEKDAY);
thermo.setThresholdDiff = 5;
// Set b=true
thermo.setOverride(true);
// Set c=false
thermo.setOvertemp(67);
// Set d=true
thermo.setTimeSinceLastRun(12);
thermo.setMinLag(10);
```

CACC Example: Test 3

Test case: FTTT

```
// Set a=false
thermo.setCurrentTemp(62);
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
thermo.setPeriod(Period.MORNING);
thermo.setDay(DayType.WEEKDAY);
thermo.setThresholdDiff = 5;
// Set b=true
thermo.setOverride(true);
// Set c=true
thermo.setOvertemp(78);
// Set d=true
thermo.setTimeSinceLastRun(12);
thermo.setMinLag(10);
```

CACC Example: **Test 4**

Test case: FFTT

```
// Set a=false
thermo.setCurrentTemp(62);
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
thermo.setPeriod(Period.MORNING);
thermo.setDay(DayType.WEEKDAY);
thermo.setThresholdDiff = 5;
// Set b=false
thermo.setOverride(false);
// Set c=true
thermo.setOvertemp(78);
// Set d=true
thermo.setTimeSinceLastRun(12);
thermo.setMinLag(10);
```

CACC Example: Test 5

Test case: FTTF

```
// Set a=false
thermo.setCurrentTemp(62);
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 70);
thermo.setPeriod(Period.MORNING);
thermo.setDay(DayType.WEEKDAY);
thermo.setThresholdDiff = 5;
// Set b=true
thermo.setOverride(true);
// Set c=true
thermo.setOvertemp(78);
// Set d=false
thermo.setTimeSinceLastRun(8);
thermo.setMinLag(10);
```

Program Transformation Issues

Reducing the number of clauses in the predicate reduces the number of test cases for each, but is that better?

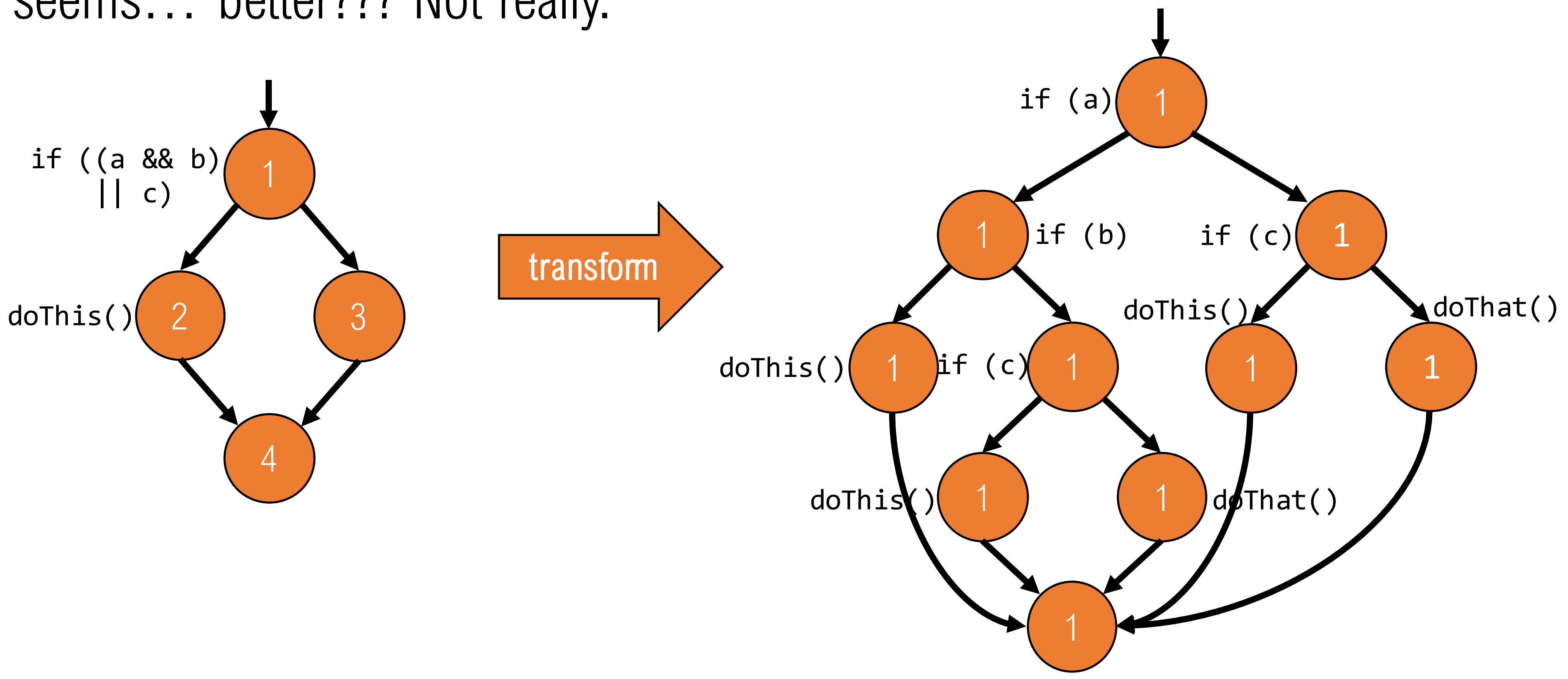
```
P1:
if ((a && b) || c) {
  doThis(…);
} else {
  doThat(…);
}
```

transform

```
P2:
if (a) {
  if (b) {
    doThis(…);
  } else {
    if (c) {
      doThis(…);
    } else {
      doThat(…);
    }
  }
} else {
  if (c) {
    doThis(…);
  } else {
    doThat(…);
  }
}
```

Testing Transformation P2

That seems… better??? Not really.



Problems with Transformation P2

We traded one problem for two problems

- Maintenance is more difficult and expensive
- Coverage actually requires more tests!

a	b	c	(a ∧ b) ∨ c	CACC on P1	PC on P2
T	T	T	T		✓
T	T	F	T	✓	
T	F	T	T	✓	✓
T	F	F	F	✓	✓
F	T	T	T		✓
F	T	F	F	✓	
F	F	T	T		
F	F	F	F		✓

Program Transformation Issues

Moving logic out of the predicates reduces the number of test cases, but is that better?

```
P1:
if ((a && b) || c) {
  doThis(…);
} else {
  doThat(…);
}
```

transform

```
P3:
d = a && b;
e = d || c;
if (e) {
  doThis(…);
} else {
  doThat(…);
}
```

Problems with Transformation P3

We moved complexity into computations

- Test criteria require fewer tests – too few?
- Less effective at finding faults!

a	b	c	(a ∧ b) ∨ c	CACC on P1	PC on P2
T	T	T	T		✓
T	T	F	T	✓	
T	F	T	T	✓	
T	F	F	F	✓	
F	T	T	T		
F	T	F	F	✓	
F	F	T	T		
F	F	F	F		✓

Transformation Undesirable

Logic coverage criteria exist to help us create better software

Artificially circumventing the criteria is not a safe behavior

When possible, simplify the logic itself rather than working around the complexity