Chapter 5:

motivations for data flow testing

memory location for variable is accessed in a desirable way; correctness of data values defined or uses of value produce the desired result

static data flow testing

identify potential defects (data flow anomaly) by analyzing code w/o execution

dynamic data flow testing

execute program and look at paths

anomaly

abnormal way of doing something

three types of anomalies

1. defined and then defined again
2. undefined but referenced
3. defined but not referenced

definition

variable gets value (on the left of the = sign)

c use

computational use (used in an expression on the right of the = sign)

p use

predicate use (used in a conditional statement such as an if or loop condition)

simple path

all nodes, except possibly the first and last, are distinct

loop free path

all nodes are distinct

complete path

entry node to the exit node

du path

variable is defined and is used in a c or p use in the same path

def use association

(variable, defined statement number, used statement number)

feasible path

looking at the path predicate will show a...

undefined variable

this lacks:
1. scope- how it will be used
2. life- allocation in memory

Chapter 6:

fundamental program elements

input domain and program path

input domain

the set of all input data to the program

program path

a sequence of instructions from entry to exit

feasible path

input data that can cause the path to execute

infeasible path

no input data exists to cause the path to execute

computation error

input data triggers the right path execution, but the output value is wrong

domain error

specific input data causes the program to execute a wrong path

domain

set of inputs for which the program performs the same computation for every member of the set

boundary inequalities

constraints on the domain

domain types

closed boundary, open boundary, closed domain, open domain, extreme point, adjacent domain

closed boundary

points on the boundary are included in the domain

open boundary

points on the boundary are not included in the domain

closed domain

all boundaries are closed

open domain

all boundaries are open

extreme point

point where two or more boundaries cross

adjacent domains

two domains with a common boundary inequality

causes of domain errors

incorrectly specified predicate, incorrect assignment to a variable used in a predicate

boundary errors

closure, shift-boundary, tilted-boundary

closure error

a boundary is open or closed when it should be the opposite

shifted boundary error

boundary is parallel to the intended boundary.
example: expected- x+y>4, actual- x+y>5

titled boundary error

constant coefficients of the variables in the predicate defining a boundary are wrong
example: expected- x+y>4, actual- x+0.5y>4

sensitivity

term used to describe the tendency of points closer to the boundary being more vulnerable to domain errors

on point

point on or very close to the boundary

off point

point that lies away from the boundary

open

on- point is in an adjacent domain
off- point is within the domain

closed

on- point is within the domain
off- point is in an adjacent domain

input classifier

name for the fact that a program classifies input into a set of (sub)domains such that the program executes a different path for each domain

domain error test criteria

for each domain and each boundary, select three points ABC in an ON-OFF-ON sequence

Chapter 7

interface error

error outside the local environment of a module that the module uses

intra system testing

low level testing by putting modules together to form a cohesive system

inter system testing

high level testing by interfacing independently tested systems

pairwise testing

two interconnected systems in a larger system are tested at a time to ensure compatibility

incremental testing

modules are added in as more testing is completed

top down testing

break down main module into submodules and test the submodules using stubs for the other dependencies

bottom up testing

test driver integrates lower modules first and we add more up as more testing goes on

big bang testing

individual modules tested first, then all put together and tested as a whole

sandwich testing

top and bottom layers tested first, then the middle layer is integrated

design verification test

hardware testing done before software is integrated

wrapper

code built to isolate underlying components from other system components

glue

functionality to combine different components

tailoring

enhance the functionality of a component

baseline security

can always go back to a previous version in a repository

stub

skeleton code used to emulate a module that is not yet available

off the shelf testing

buy a component from someone and test it

Chapter 8:

basic

system can be setup and brought to an operational state

functionality

comprehensive testing over the full range of requirements

robustness

how well the system recovers from various errors and failures; test invalid values to confirm this

interoperability

how well the system can work with other products

performance

measure the ability of the system under different conditions

scalability

determines how well the program can be expanded or decreased

stress

put the system under pressure

load

prove system can remain stable over a long period of time under stress

reliability

measure how well system can operate without failures

regression

test system stability following changes

documentation

ensure guides are accurate and usable

types of basic tests

boot, down/upgrade, led, diagnostic, CLI

security

test integrity; is the data we get expected?

online insertion and removal

system is running and you lose key component; ex: driving and take out key

mean time between failures

MTBF

Chapter 9:

four key concepts

1. precisely identify the domain of each input and output variable
2. select values from the data domain of each variable having important properties
3. Consider combinations of special values from different input domains to design cases
4. Consider input values such that program under test produces special values from the domains of the output variables

test vector

test data inputs for a program

functionally related

variables that appear in the same assignment statement together or the same branch predicate

pairwise testing

each possible combination is covered in one test case only

equivalence class partitioning

domain is broken up into sub-domains that encompass the entire domain together; union of all subsets give back the original set; intersection of any 2 subsets are an empty set (no overlap between sets)

decision table

Y and N are put into the table for the possible conditions and actions to take which results in test cases

random testing

choosing any value; requires a good test oracle because taking any value might not have a known, expected output

error guessing

based off experience; test areas that tend to be more problematic

category partition

(CPM) divide test vector into categories that represent major characteristics of the domain

Chapter 10:

mealy machine

output determined by state AND inputs

moore machine

output determined only by state

transition tour

(TT) sequential transition from initial state to final state (not necessarily the last state number)

state coverage

cover every state at least once

transition coverage

cover each transition at least once

conceptual components of software

1. flow of control
2. manipulation of data

point of interaction

area where there is an interaction between a system and its users