Contents

Verification 2
- Black box testing 2
- White box testing 2
- **Goals** of verification 2
- Definitions 2
- Test quality 3
- More definitions 3
- Test criteria 3

Testing 5
- Statement coverage 5
- Branch coverage 6
- Condition coverage 6
- Path coverage 7
- Modified condition or decision coverage (MCDC) 7
  - Condition 7
  - Decision 7
- Black box testing 8
  - Tabular expression 8
  - Boundary conditions 8
- Difference between black box and white box testing 9
- Testing in the large 9

Safety 10
- Hazard analysis 10
- Mitigating hazards 11
- Fault tree analysis 12
- Cut sets 12
- Failure modes & Effects analysis (FMEA) 13
- STPA (Systems theoretic process analysis) 14

Verification

- Everything must be verified, every required quality, process, and products
- May not be binary (absolute)
    - Some defects can be tolerated
- May be subjective or objective
- Implicit (hidden) qualities should be verified

Black box testing

- Perform verification when people **don't know** the internal elements of the system
- Used to test the system against external factors responsible for software failures
- Testing about the system

White box testing

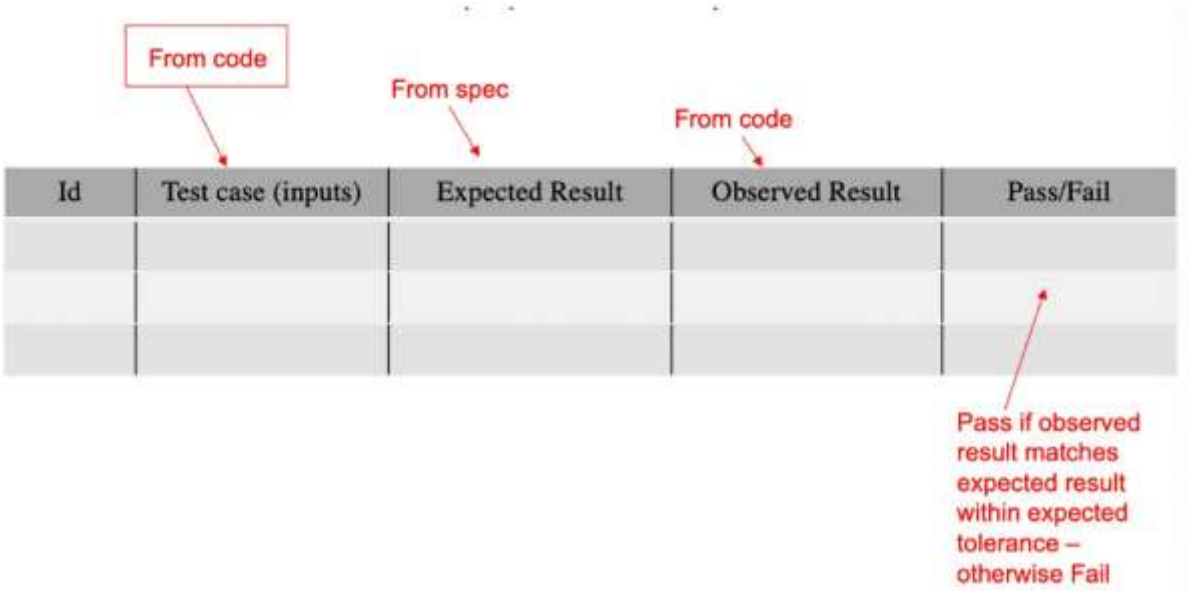
- Perform verification while people **can access and analyze** the internal elements
- Allows testers to inspect and verify the inner workings of a software system
- Testing about the algorithm

Goals of verification

- Ensures the specific software components or subsystems meet their design requirements
    - Check the presence of problematic behaviors → testing
        - Purpose is to find a problem or search for a problem
    - Guarantee the absence of problematic behaviors → model checking
        - Try all possible inputs of the system and no errors
        - Time consuming

Definitions

- P → program
- I → set of possible inputs
- O → set of possible outputs
- IOS → input output of the system
- Complete-coverage principle



Test quality

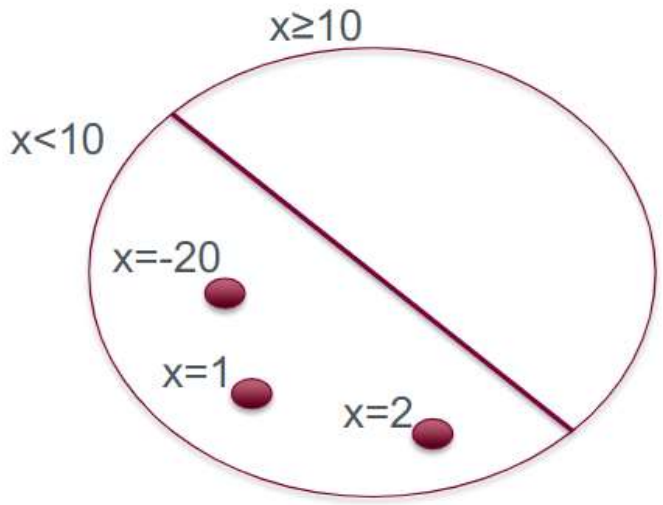
- Should help isolate and replicate error behaviors
- Test should be repeatable
    - Repeating same experiment, we expect to get the same result
    - Not happened all the time due to the execution environment
- Should be accurate
- Failure
    - Find the red dot or incorrect output outside from correct output circle
- Error (defect)
    - Anything that may cause a failure
        - Eg: Typing mistake
- Fault
    - Incorrect intermediate state entered by program
- Testing
    - Random or search based techniques
        - Automatic generate the test cases
    - Systematic procedures
        - Try to manually identity test cases

More definitions

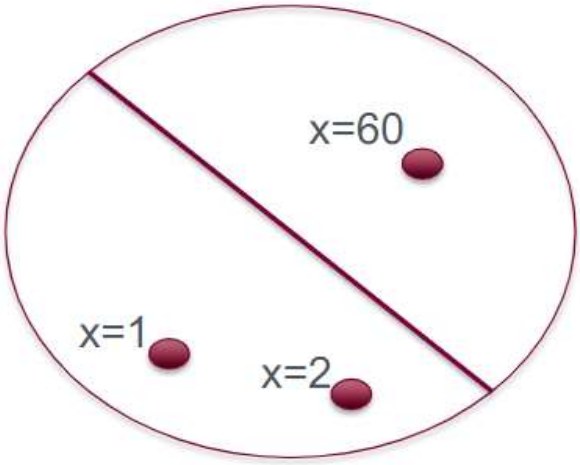
- Test cases t
    - An element of I
- Test set T
    - A finite subset of I
- Test is successful if <i, P(i)> doesn't belong to C
- Test set is unsuccessful if P correct for all i in I

Test criteria

- Define a criteria and splitting input domain into different areas

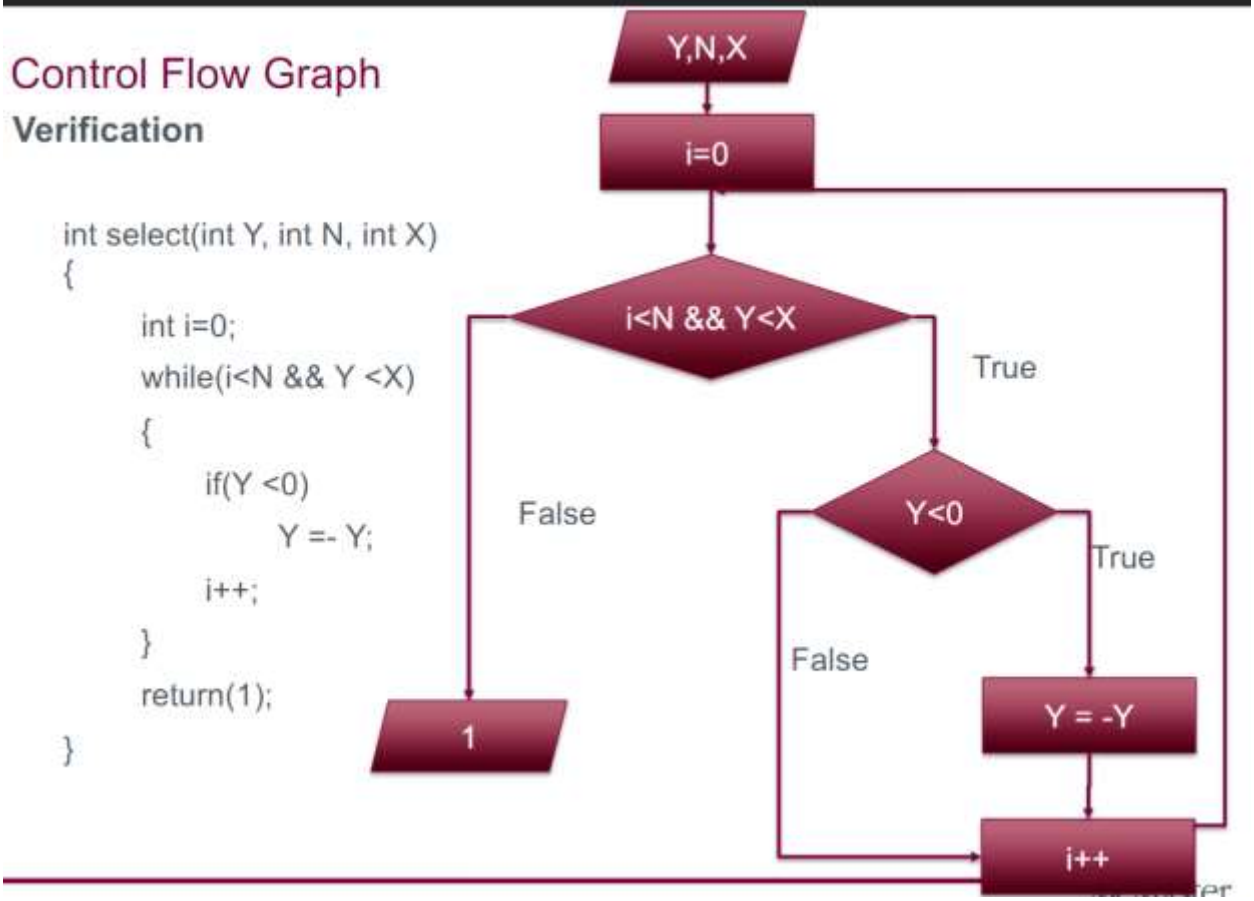


CR1 = {x | x>= 10}
CR2 = {x | x< 10}
T = {x = 1,x=2, x=-20}
This doesn't not satisfy the test criterion



T = {x = 1, x = 2, x = 60}
This satisfies the test criterion

Testing



Statement coverage

Given a set of test cases T such that each statement of the code is executed by at least on test case in T.

- If some instructions were never executed, there might be an error there
- Example
  - N = 1, Y = -7, X = 9
- Statement coverage = $\frac{\#\ of\ statement\ Covered}{\#\ Total\ Number\ of\ statements}$
- Can't always cover 100% of statements
- Can't write an algorithm that let user know if an instruction is reachable

i=0
i<n && Y<X
Y<0
Y'=-Y
i'=i+1
¬(i'<N && Y'<X)

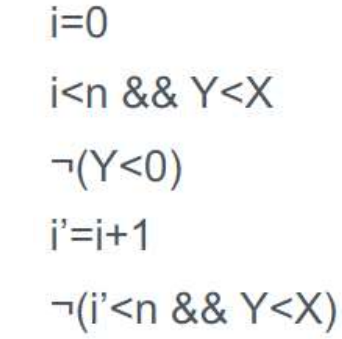
A set of conditions that satisfy the coverage
But doesn't cover all the branches

It covers statements but not branches

Branch coverage

Given a set of test cases T such that each branch of the code is executed by at least one test case in T

- If some branches were never executed, there might be an error there



$$\text{Brach coverage} = \frac{\#\ branch\ coverd}{\#\ total\ number\ of\ branch}$$

Branch coverage will cover all the statement
Statement coverage will NOT cover all branches

Condition coverage

Given a set of test cases T such that each part of the condition of the code is evaluated both as true and as false

- If some conditions were never executed there might be an error there

```
if (i==0 && j==0) {
    do something
} else {
    do something else
}
```

DOES NOT MEAN COVERAGE ALL COMBINATIONS

T = {<i = 0, j = 0> <i = 1, j = 1>} → covers all conditions

Also depends on how complier works

One more test case need to be in T <i = 0, j = 1>

- In some languages (or most of languages), if i = 0, the complier won't even start to check where j = 0 or not.
- So, in order to cover all the condition, if i = 0, we need to check if j = 1 as well

Condition coverage DOES NOT mean branch coverage

- T = <i = 0, j = 1> , <i = 1, j = 0>

Branch coverage DOES NOT imply condition coverage

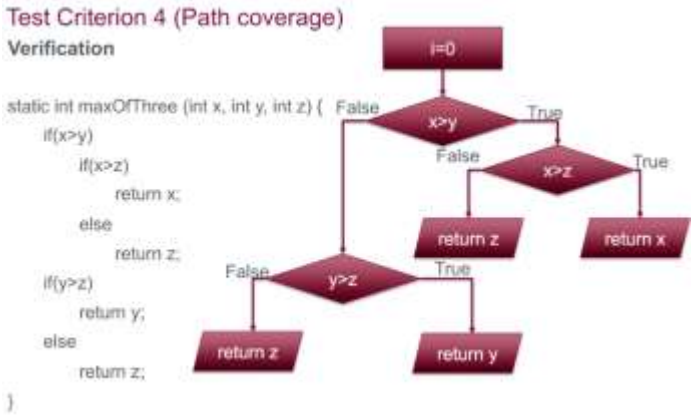
Path coverage

Given a set of test cases T such that each path of code is executed by at least one test cases in T.

- If some paths were never executed, there might be an error there
- Try all the possible ways or all possible combinations that code can be executed

CAN NOT always cover all the path due some special loop condition (while (true) or some for loop) due to infinite amount of test cases

Path coverage IMPLIES branch coverage and instruction coverage



<1,2,3>;<1,2,4>;<1,5,6>;<1,5,7>

Modified condition or decision coverage (MCDC)

- Basically, an extended version of the condition coverage
  - Every entry and exit is covered
  - Each decision exercises every possible outcome range
  - Each condition in a decision takes on every possible outcome range
  - Each condition in a decision must independently change the outcome
- Makes people write better requirements

Condition

- A leaf level Boolean expression
  - a == 0
  - b == 0

Decision

- A Boolean expression composed of conditions and zero or more Boolean operators
- A decision without a Boolean operator is a condition
  - a == 0 && b == 0

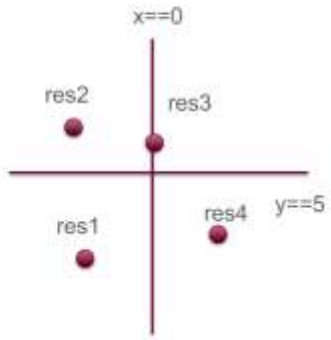
Black box testing

Can't access the source code

- Use requirement to define test cases
- Random generation of test cases
- Use tables representing behaviour at requirement level

Tabular expression

		f_name
x < 0	y ≤ 5	res 1
	y > 5	res 2
x = 0		res 3
x > 0		res 4

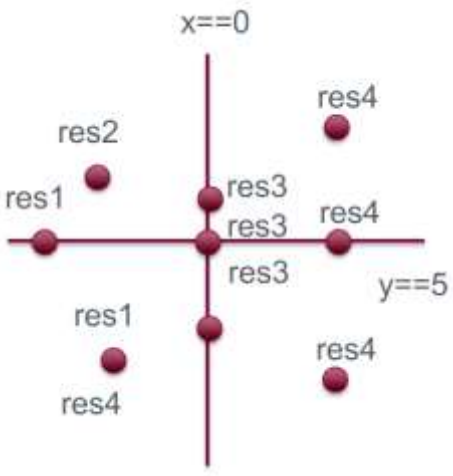


The main idea of this testing approach is to cover each line of the table

Boundary conditions

In addition to cover each row of the table, you want also to cover the boundaries (the place you switch from one row of table to another)

- In boundary condition, more problematic behaviours occur
- Example
    - A square root function, the boundary would be 0
    - Any thing lower than 0 would cause problematic behaviour
    - Another example would be using an array with specific index
    - You can not use index that are outside of an array
        - Depends on programming language (Fxxk MATLAB, it starts from 1)



Difference between black box and white box testing

In practice, both testing techniques are needed → no preference
They do complementary stuff
White box testing is used while developing your code

- Unit testing and develop test cases

Black box

- Try to analyze the system which you don't have access to
- Whether the system behaves correctly with different inputs

Testing in the large

Types of testing

1. Module or unit testing
- Testing a single unit of code
- Stubs
    - Modules used by the test cases
    - A small piece of code that takes the places of another component during testing
    - For example
        - If you are building a user interface in pacemaker project
        - All you care will be if the UI system works well
        - No need for the Simulink board
        - You can send a fake signal (stab) to UI to check if it's working correctly
- Driver (harness)
    - Module activating the model under test
    - Something you need to create to enable the test cases
    - A piece of code so that it becomes easier to call another programs or modules
2. Integration testing
    - Integration of units and subsystems
    - Basically, combine all the models and subsets then do the test
    - To make sure, after each unit join together, they can act together
3. System testing
    - Testing the entire system
    - To guarantee that the whole system is working
4. Acceptance testing
    - Performed by the customer

Oracle

- It tells you what the correct answer for test cases is
- The drawing beside the tabular expression table is an oracle

Debugging

- The activity of finding errors
- It can start once a failure has been detected
- Example
    - Put a breakpoint and test model step by step

Analysis

- Performed by a small group of people
- Focus on finding errors, not fixing them
- Experience shows that most errors are discovered by the designer during the presentation

Safety

Safety

- Safety is freedom from harm
- Being protected from things that could unintentionally harm you
- Safety is not absolute
- Safety is NOT correctness
- Need security to be safe

Security

- Deliberate protection against threats
- The measures when against threats
- Being protected from things that are meant to harm you

Hazard analysis

- A hazard is a system state or set of conditions that will lead to a loss
- Hazard analysis is a process by which we identify hazard and their causes, and then specify ways in which we can eliminate them or decrease their effect

Hazard analysis

- Designed to
    - Identity hazards in the system
    - Determine the consequences of each hazard
    - Determine the likelihood of the hazard
    - Determine how to eliminate each hazard or at least mitigate the consequence of each hazard
- The method to eliminate or mitigate a hazard result in one or more safety requirements for a system
- Typical methods in use in the medical domain are
    - HAZOP → hazard and operability analysis
        - Comes from insurance domain
        - Identify the risks for employer
            - Checklist
            - What-if
        - Forming a HAZOP team
        - Identifying the elements of the system
        - Considering possible variations in operating parameters
        - Identifying any hazards or failure points
    - FMEA & FMECA → Failure modes and effects analysis
        - From military domain (US)
        - Used in automotive domain
        - Define the system, ground rules and assumptions
        - Construct system boundary diagrams and parameter diagrams
        - Failure analysis activities
        - Typically bottom-up → start from identity failure → analyze the effect
    - FTA → Fault tree analysis
        - Top – down
        - Identify hazard

        - Obtain understanding of the system being analyzed
        - Create the fault tree
        - Identify the cut sets
        - Mitigate the risk
    - STPA → system theoretic process analysis
        - Treat accidents as control problem
        - Identify the potential for inadequate control of the system leading to hazardous state

Four types of unsafe control actions

- Control commands required for safety are not given
- Unsafe ones are given
- Potentially safe commands but given too early, too late
- Control action stops too soon or applied too long

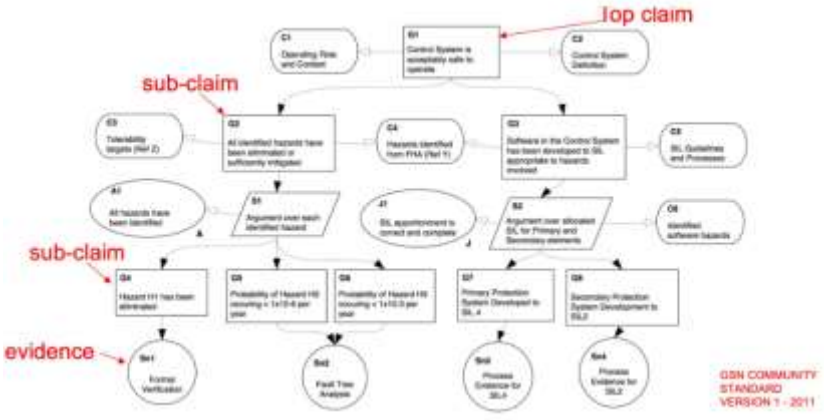
Safety integrity levels (SILs)

- A societal judgement on what constitutes tolerable risk in the specific domain
- ASIL D in automotive is most critical
- ASIL A is the least critical
- In medical devices SIL 3 is the most critical
- SIL 1 is the least

Mitigating hazards

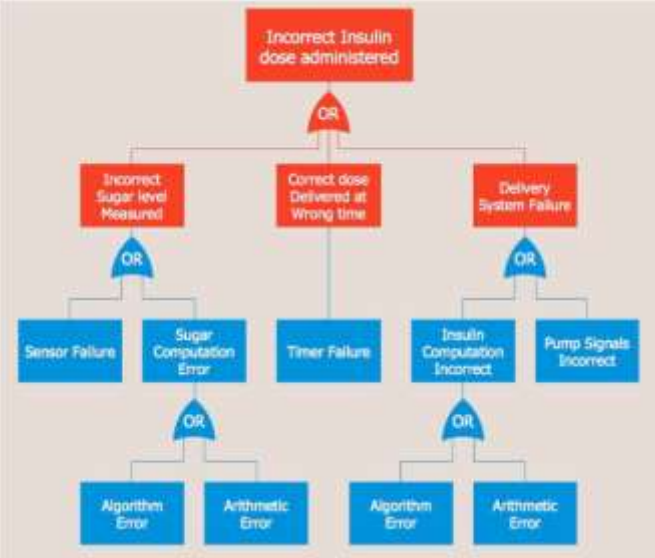
- Redundancy
    - The duplication of components or functions of a system to increase reliability
    - Copy components to avoid one component goes wrong
    - These components are typically clines or very close to clones
    - The idea is to use probabilistic reasoning to improve our chances of success
    - It is NOT used to prevent software failures
    - It is there to mitigate against hardware failures
- Diversity
    - Realization of a desired outcome by different means
    - Diversity is mitigation against systemic errors
- Assurance case → generalization of a safety case
    - Basically, show an argument and provide evidence to show the hazard has been mitigated

Goal structing notation
Decompose high level goals to sub goals
Also provide solution and evidence



Fault tree analysis

- Determine the root causes and probability of occurrence of undesired events
- Determine the combinations of basic events that can cause the specified undesired event

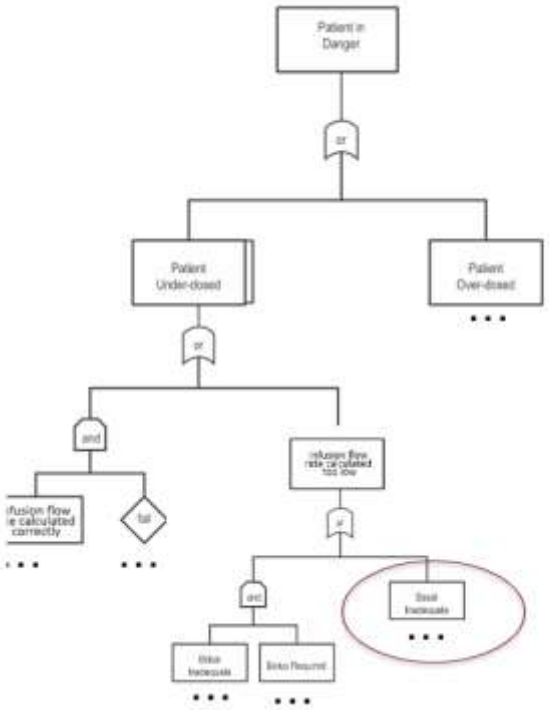


An example of FTA

From high level to real low level

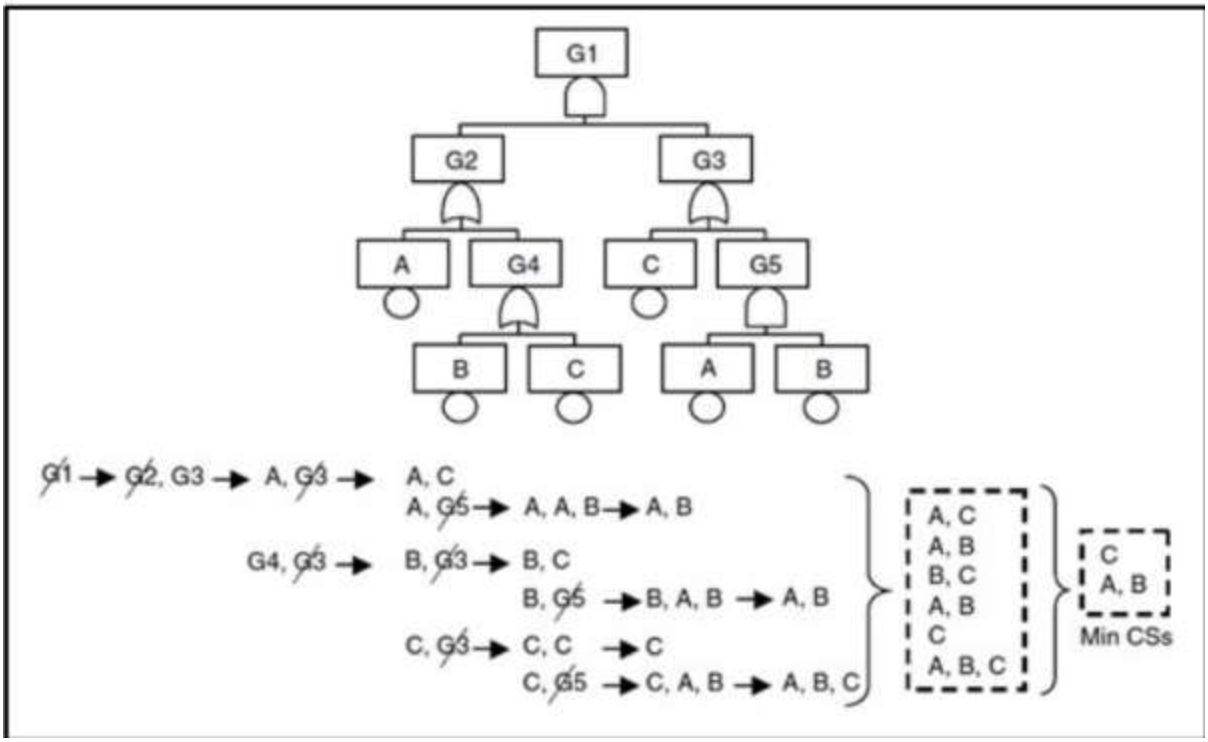
Cut sets

- A set of basic events whose occurrence ensures that the TOP event occurs
- In other words, if cut set is true, high level event is also true
- Minimal cut set
    - A cut set that cannot be reduced without losing its status as a cut set



cut set:
Infusion flow rate calculated correctly
Bolus inadequate
Bolus required
Basal inadequate

Minimal cut set:
Basal inadequate



To determine where a cut set is a minimum

- Check if it has other cut set
- For example, A, C is not a minimum because it has C

Failure modes & Effects analysis (FMEA)

- Developed for the US military
- It is a disciplined bottom up method
- Determine all possible failure modes for systems, subsystems, or components

Component	Failure Mode	Failure Rate	Causal Factors	Immediate Effect	System Effect	RPN

- Failure
  - Departure of an item from its required or intended operation, function, or behavior problems that users encounter.
- Failure mode
  - Describes the way the failure occurs and its impact on equipment operation
- Failure cause
  - Basic reason for failure
- Failure effect
  - The consequence a failure mode has on the operation, function, or status of an item. It can be divided into immediate effect and system effect
- RPN (risk priority number)

- Risk ranking index for reliability
- RPN = (probability of occurrence) (severity ranking) (detection ranking)
- Detection ranking ranks the ability of planned tests and inspections to detect failure modes in time

Failure Mode and Effects Analysis

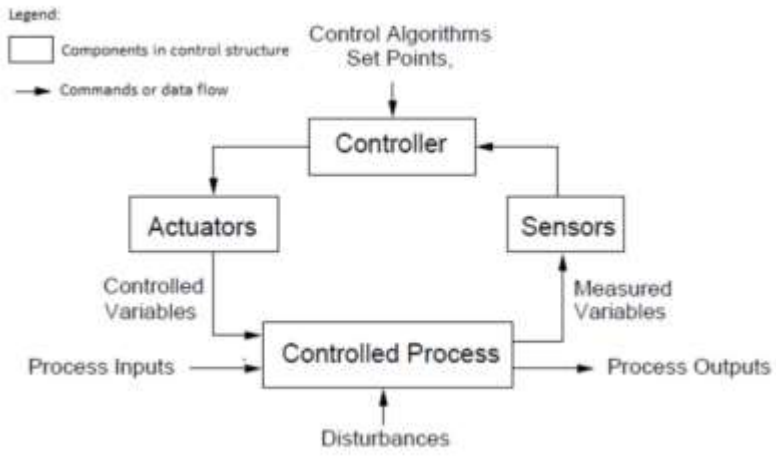
System: Generic Insulin Infusion Pump
Subsystem: N/A
Phase/Mode: System Requirements

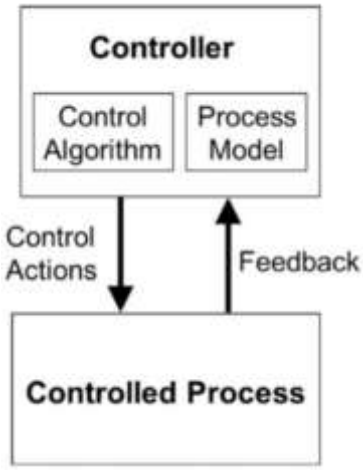
Design Function	Failure Modes	Effects of Failure	Causes of Failure	Detection	Occ	Risk	Recommended Action	SR	Ref
Deliver insulin	Does not deliver enough or fails to deliver any insulin	Hyperglycemia in patient	a. Insulin delivery mechanism failure; b. Infusion delivery control failure; c. Incorrect basal or bolus bounds settings; d. Incorrect basal or bolus profile settings; e. Incorrect profile activation; f. Controller failure; g. Not enough insulin available;				a. Include flow meter in system design to measure insulin output flow and check against controlled output, report system error in case of malfunction. b. Each component should verify the validity of the signal it is receiving. c. Refer to H2. d. Refer to H3. e. Refer to H4. f. Require system to detect controller failure (e.g. use watchdog to detect system lockup). g. Require device to check insulin levels before and during insulin delivery and notify user when insulin levels are low.	a:SR-1 b:SR-2 f:SR-3 g:SR-4	H1-1
	Delivers too much insulin	Hyperglycemia in patient	a. Insulin delivery mechanism failure; b. Infusion delivery control failure; c. Incorrect basal or bolus bounds settings; d. Incorrect basal or bolus profile settings; e. Incorrect profile activation; f. Controller failure.				a. Same as H1-1a. b. Same as H1-1b. c. Refer to H2. d. Refer to H3. e. Refer to H4. f. Same as H1-1f.	a:SR-1 b:SR-2 f:SR-3	H1-2
	Takes too long to deliver insulin	Hyperglycemia in patient	a. Insulin Delivery Mechanism Failure; b. Infusion delivery control failure; c. Incorrect basal or bolus bounds settings; d. Incorrect basal or bolus profile settings; e. Incorrect profile activation; f. Controller failure.				a. Same as H1-1a. b. Same as H1-1b. c. Refer to H2. d. Refer to H3. e. Refer to H4. f. Same as H1-1f.	a:SR-1 b:SR-2 f:SR-3	H1-3

SR → safety requirement

STPA (Systems theoretic process analysis)

- Consider a system as a control problem rather than a failure problem
- Lead to mitigating hazards at a higher abstraction level
- Highly dependent on the control diagrams
- Prevent accidents by constraining the component behavior and interactions





- Identify accidents and hazards
- Draw the control structure
- Step 1 → Identify unsafe control actions
- Step 2 → Identify causal scenarios

Failure occurs with following four different reasons

- Unsafe control actions (UCA) are given
- Control actions required for safety are not given
- Control actions stops too soon or applied to long → duration of the action
- Control actions commands but given too early, too late → the moment of action

Control Action	Not providing causes hazard	Providing causes hazard	Too early, too late, out of order	Stopped too soon, too late
Action 1	UCA-1: text	UCA-2: text	UCA-3: text	UCA-4: text
	UCA-5: text	UCA-6: text	UCA-7: text	UCA-8: text
Action 2	UCA-9: text	UCA-10: text	UCA-11: text	UCA-12:text