วิศวกรรมซอฟต์แวร์

Software Engineering

สมเกียรติ วังศิริพิทักษ์

somkiat.wa@kmitl.ac.th

ห้อง 518 หรือ ห้อง 506 (MIV Lab)

PART II

Quality Management

Testing Conventional Applications

Testing Conventional Applications

- We will discuss **techniques** for software **test-case design** for **conventional applications**.
  - Test-case design focuses on a set of techniques for the creation of test cases that **meet** overall testing **objectives** and the testing **strategies**.

Software Testing Fundamentals

- You should **design** and **implement** a computer-based system or a **product with** “**testability**” **in mind**.
  - The **tests** themselves must exhibit a set of characteristics that **achieve the goal** of **finding** the **most errors with** a **minimum of effort**.

คนออกแบบ ต้อง ออกแบบให้ SW
มัน เทสง่าย ด้วย

Software Testing Fundamentals
Testability

- Definition for testability:
  - = "Software testability is simply how easily [a computer program] can be tested."

- The following characteristics lead to testable software.

Software Testing Fundamentals
Testability

- Operability—it operates cleanly
  - "The better it works, the more efficiently it can be tested."

- Observability—the results of each test case are readily observed
  - "What you see is what you test."

- Controllability—the degree to which testing can be automated and optimized
  - The better we can control the software, the more the testing can be automated and optimized.

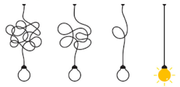
แยกส่วน เพื่อให้ทดสอบแยกกัน , ง่ายขึ้น

- Decomposability—testing can be targeted
  - "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting."
  - The software system is built from independent modules that can be tested independently.

Software Testing Fundamentals
Testability

ค.ง่าย

- Simplicity—reduce complex architecture and logic to simplify tests
  - "The less there is to test, the more quickly we can test it."



มีแค่ function ที่ test ที่จำเป็น

  - *functional simplicity* (e.g., the feature set is the minimum necessary to meet requirements);
  - *structural simplicity* (e.g., architecture is modularized to limit the propagation of faults), and

ทำโค้ดเป็นตามมาตรฐานการเขียน

  - *code simplicity* (e.g., a coding standard is adopted for ease of inspection and maintenance).

Software Testing Fundamentals
Testability

เสถียร

- Stability—few changes are requested during testing
  - "The fewer the changes, the fewer the disruptions to testing."

เข้าใจง่าย

- Understandability—of the design
  - "The more information we have, the smarter we will test."

What is a “Good” Test?

มีโอกาสเจอ error ได้สูง

- A good test has a **high probability** of **finding an error**
- A good test is **not redundant.**
  - Testing time and resources are limited.
  - No point in conducting a test that has the same purpose as another test.

Equivalence Partitioning

เจ๋งที่สุด

- A good test should be “**best of breed**” คือ test case ที่ มันเจอ error ง่าย
  - Those tests that has the highest likelihood of uncovering a whole class of errors
- A good test should be **neither too simple nor too complex**
  - *Combining a series of tests into one test case* may mask errors.
  - In general, each test should be executed separately.

Internal and External Views of Testing

- Any engineered product (and most other things) can be tested in one of **two ways**:

Functional testing

External view
“Black-box testing”

เน้นข้าง
นอก

(1) Knowing the **specified function** that a product has been designed to perform,

- tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;

ทำงานข้างใน

(2) Knowing the **internal workings** of a product,

- tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised.

เน้นข้างใน

Internal view
“White-box testing”

Structural testing

Pliche

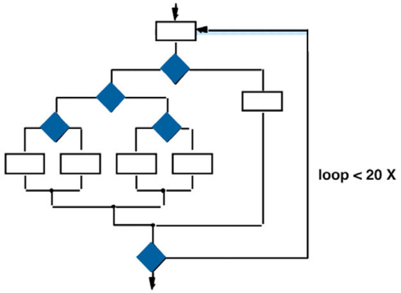
100 line + nested loop program.

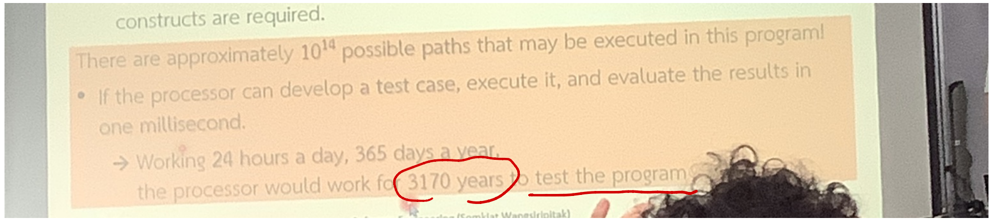
การจะเทส เคสทดสอบทุกกรณีเป็นไปได้ไหม

ตอบ absolutely not

Exhaustive Testing

- Consider a 100-line program in the language C.
- After some basic data declaration, the program contains two nested loops that execute from 20 ครั้ง 1 to 20 times each, depending on conditions specified at input.
- Inside the interior loop, four if-then-else constructs are required.



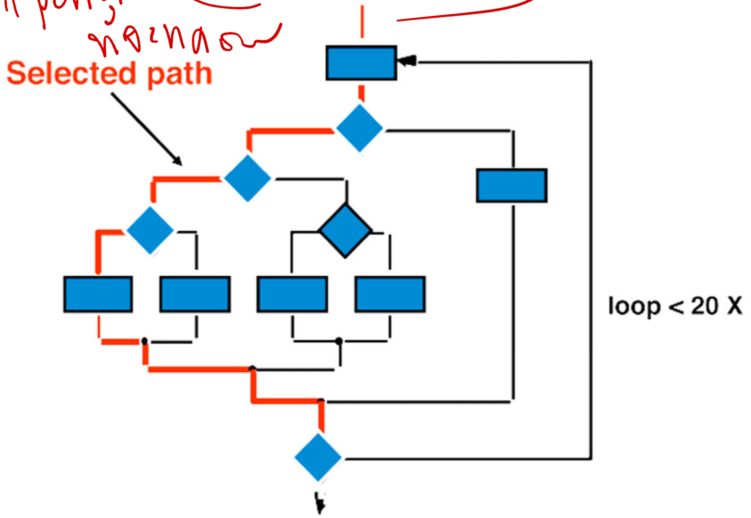

constructs are required.

There are approximately 10^{14} possible paths that may be executed in this program!

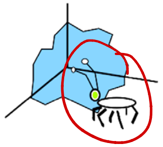
- If the processor can develop a test case, execute it, and evaluate the results in one millisecond.
  → Working 24 hours a day, 365 days a year, the processor would work for 3170 years to test the program

Selective Testing

เลือก path ที่จะทดสอบ



Test Case Design



bugs อยู่ที่มุม

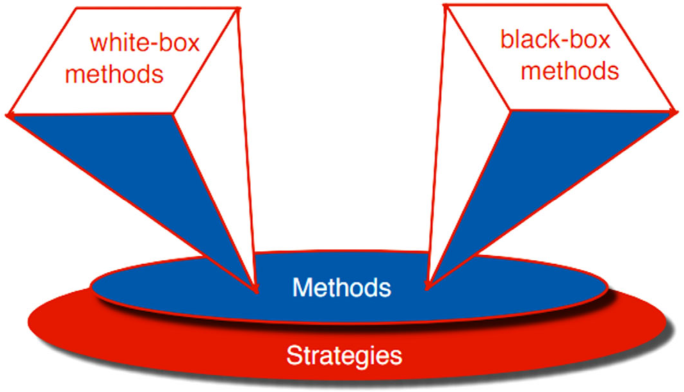
- "Bugs lurk in corners and congregate at boundaries." -- Boris Beizer

- **Objective** : To uncover error มุ่งเน้น error
- **Criteria** : In a complete manner ครอบคลุม
- **Constraint** : with a minimum of effort and time

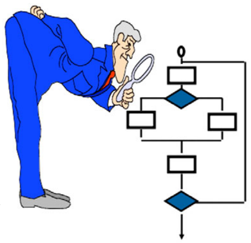
ทำได้ภายใน

เวลา + effort ที่จำกัด

Software Testing



White-Box Testing



- White-box testing, sometimes called **glass-box testing** or **structural testing**, is a test-case design philosophy that ***uses*** the ***control structure*** described as part of component-level design ***to derive test cases***.

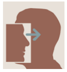
ส่วที่จะทดสอบ

- Using white-box testing methods, you can derive **test cases** that ...

(1) guarantee that all independent paths within a module have been **exercised at least once**,

(2) exercise all logical decisions on their true and false sides,

(3) execute all loops at their boundaries and within their operational bounds, and



(4) exercise internal data structures to ensure their validity

Basis Path Testing

ทดสอบเส้นทางพื้นฐาน

- **Basis path testing** is a white-box testing technique first proposed by Tom McCab.
- The basis path method enables the test-case designer to ...
  - **derive** a **logical complexity measure** of a procedural design and
  - **use** this measure **as a guide** for **defining** a basis set **of** execution paths.
- Test cases derived to exercise the basis set are **guaranteed** to **execute every statement** in the program **at least one time** during testing.

ทุก statement

ต้องได้รับการทดสอบ

อย่างน้อย 1 ครั้ง

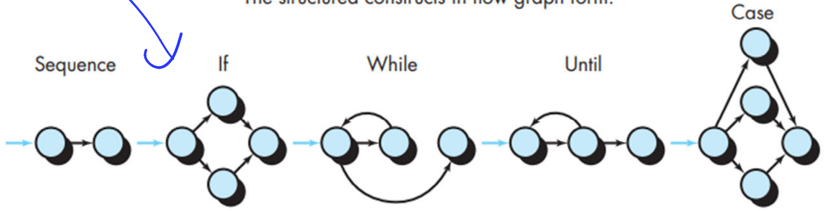
Basis Path Testing

Flow Graph Notation

A flow graph should be drawn **only when** the logical **structure** of a component is **complex**.
The flow graph allows you to trace program paths more readily.

- To understand the basis path method, a simple **notation** for the representation of control flow, called a **flow graph** (or **program graph**) must be introduced.
- The flow graph depicts logical control flow using the notation illustrated in the figure.

The structured constructs in flow graph form:



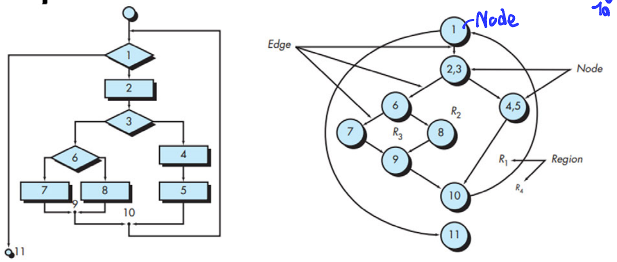
Where each circle represents one or more nonbranching PDL or source code statements

Basis Path Testing

Flow Graph Notation

1 วงกลม สามารถมี statement>1 ได้

- Example of using a flow graph to represent the procedural design in the figure.
- Left figure: a **flowchart** is used to depict program control structure.

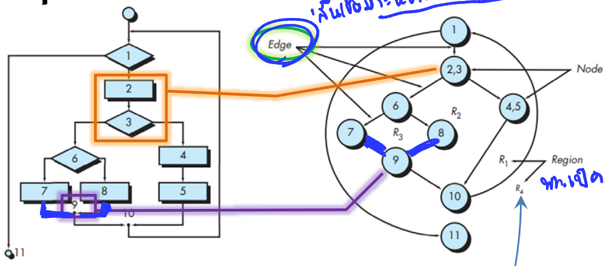


- Right figure: Map the flowchart into a corresponding **flow graph** (assuming that no compound conditions are contained in the decision diamonds of the flowchart).
  - Each **circle**, called a **flow graph node**, represents one or more procedural statements.

Basis Path Testing

Flow Graph Notation

'เส้นเชื่อมระหว่าง Node ต่างๆ'

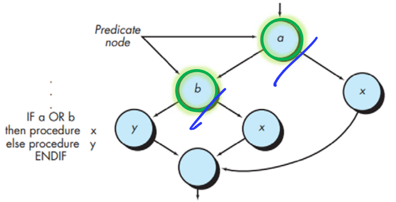


- A sequence of process boxes and a decision diamond can map into a single node.
- The arrows on the flow graph, called **edges** or หรือ **links**, represent flow of control and are analogous to flowchart arrows.
  - An edge must **terminate at a node**, even if the node does not represent any procedural statements (e.g., see the flow graph symbol for the if-then-else construct).
- Areas bounded by edges and nodes are called **regions**.
  - When counting regions, we include the area outside the graph as a region.

รวม พท.เปิดด้วย

Basis Path Testing

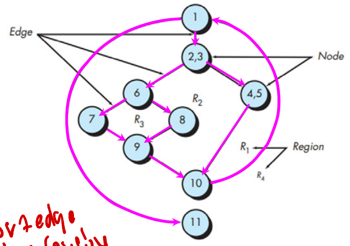
Flow Graph Notation



- Compound conditions in a procedural design **make** the generation of a flow graph **slightly more complicated**.
- A compound condition occurs when **one or more Boolean operators** (logical OR, AND, NAND, NOR) is present in a conditional statement.
- Referring to the **figure**, the program design language (PDL) segment translates into the flow graph shown.
- Note that a **separate node** is created **for each of the conditions** `a` and `b` in the statement `IF a OR b` compound condition → แยก node
  - Each **node** that **contains** a **condition** is called a **predicate node** and is characterized by two or more edges emanating from it.

2+ edges

Independent Program Paths



- An **independent path** is any path through the program that introduces at least one new set of processing statements or a new condition.
  - In terms of a flow graph, an independent path must *move along* ***at least one edge*** that → อย่างน้อย 1 edge ที่ไม่เคยโดนผ่าน has ***not been traversed before*** the path is defined.
- For example, **a set of independent paths** for the flow graph illustrated in this figure is …

รอบที่1 → **Path 1:** 1-11 ยังไม่ได้ไปไหน เดินครั้งแรก

Path 2: 1-2-3-4-5-10-1-11

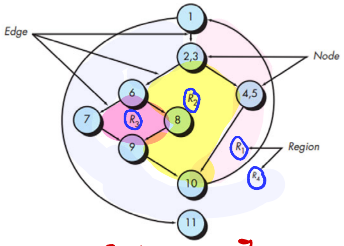
Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

Note that each new path introduces a new edge.

The path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 is **not** considered to be an **independent path**

Independent Program Paths

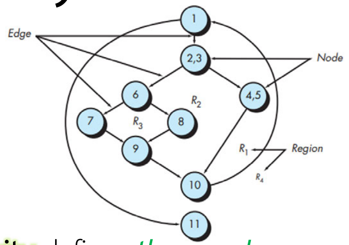


Region = พท.ปิด

- Paths 1 through 4 constitute a **basis set** for the flow graph in this figure.
  - If you can design **tests** to **force execution** of **these paths** (a basis set),
    - **every statement** in the program will have been **guaranteed** to **be executed at least one time** and
    - **every condition** will have been **executed** on its **true** and **false** sides.
- Noted that the basis set is **not unique**.
  ↳ ที่แต่ละคนหาอาจไม่เหมือนกันก็ได้
  - A number of different basis sets can be derived for a given procedural design.
- How do you know **how many paths** to look for?
  - The computation of **cyclomatic complexity** provides the answer.
    ตัวกำหนดว่า basisset ควรมีทั้งหมดกี่

Cyclomatic Complexity

ใช้คำนวณค่า

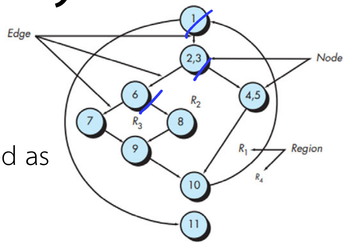


- Cyclomatic complexity is a software metric that provides a quantitative measure of the **logical complexity** of a **program**.
- When used *in the context of the basis path testing* method, the value computed for **cyclomatic complexity** defines *the number of independent paths* in the **basis set** of a program and provides you with an **upper bound** for *the number of tests* that **must be conducted** to **ensure** that **all statements** have been **executed at least once**.
- Complexity is computed in one of **three ways**:

3 วิธี

Cyclomatic Complexity

จำนวน regions



1. The **number of regions** of the flow graph corresponds to the cyclomatic complexity.
2. Cyclomatic complexity $V(G)$ for a flow graph G is defined as
   $V(G) = E - N + 2$
   where E is the number of **flow graph edges** and
   N is the number of **flow graph nodes**.
3. Cyclomatic complexity $V(G)$ for a flow graph G is also defined as
   $V(G) = P + 1$
   มีเส้นออกมา 2 เส้น
   where P is the number of **predicate nodes** contained in the flow graph G.

- The cyclomatic complexity of the flow graph above is:
  1. The flow graph has ? regions. = 4
  2. V(G) = ? 11-9+2 = 4
  3. $V(G)$ = ? 3+1 = 4
- Therefore, the cyclomatic complexity of the flow graph in this figure is ?

∴ 4

Cyclomatic Complexity

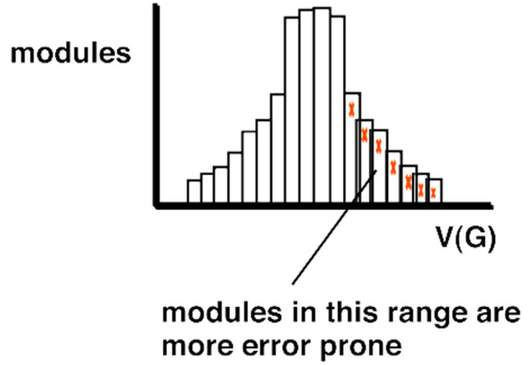
- More important, the value for ***V(G)*** provides you with
  - an **upper bound** for the number of independent paths that form the basis set and, by implication,
  - an **upper bound** on the number of tests that must be designed and executed to guarantee coverage of all program statements.

จแนวโน้มที่จะเกิด error

- Cyclomatic complexity is a useful metric **for predicting** those **modules that are likely to be error prone**.
- Use it **for test planning** as well as **test-case design**.

Cyclomatic Complexity

A number of industry studies have indicated that the higher V(G), the higher the probability or errors.


modules
V(G)
modules in this range are more error prone