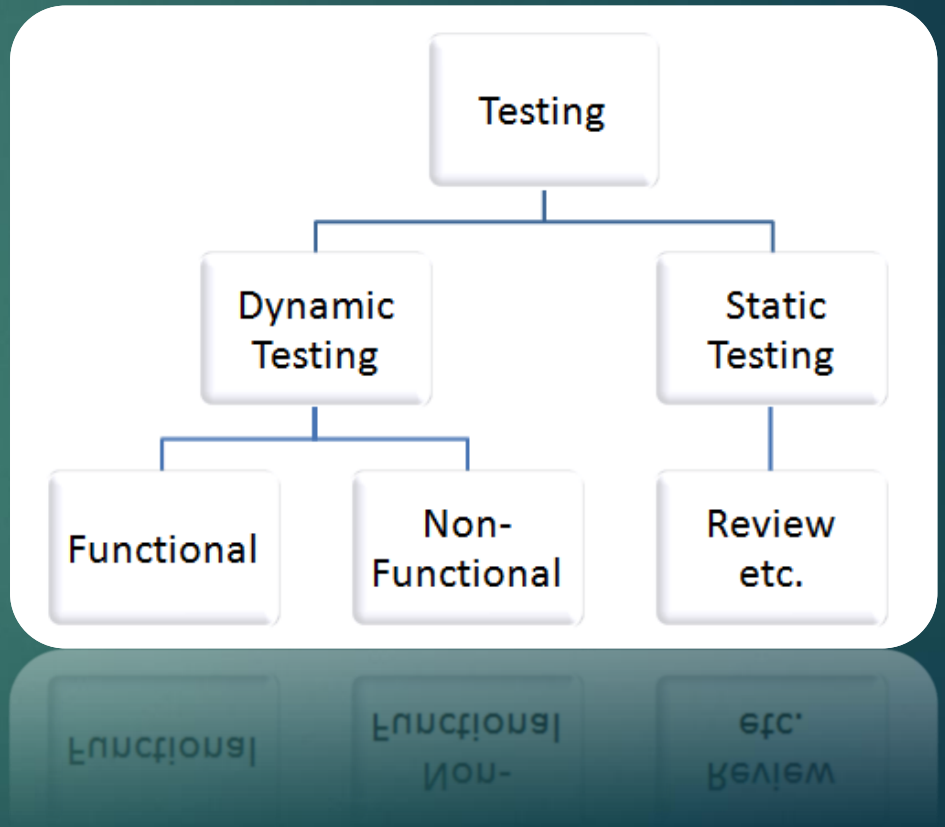
Software Quality Assurance & Testing

LECTURE 4: STATIC TEST TECHNIQUES

Prepared By:
Md. Minhaj Hosen, Lecturer, DIU

Outlines

1. Static Techniques and the Test Process
2. Review Process
3. Static Analysis by Tools



STATIC TECHNIQUES

- **Basic Approach**
- **Static** testing techniques summarize various methods, that **do not execute** the component or the system that is being tested.
  - Static testing include:
    - **reviews** (manual activity)
    - **static analysis** (mostly tool based activity)
- Static testing techniques complement dynamic test methods
  - static testing find causes of failures **(defects)** rather than **failures**
  - **concepts** are inspected as well, not only executable code
  - defects / deviations are found in an **early phase**, before they are implemented in the code
  - **Static** testing might find defects not found in **dynamic** testing
- **High quality documents** lead to high qualify product
  - even if the reviewed specifications do not contain any errors, **interpreting** the specification and creating the design could be **faulty**

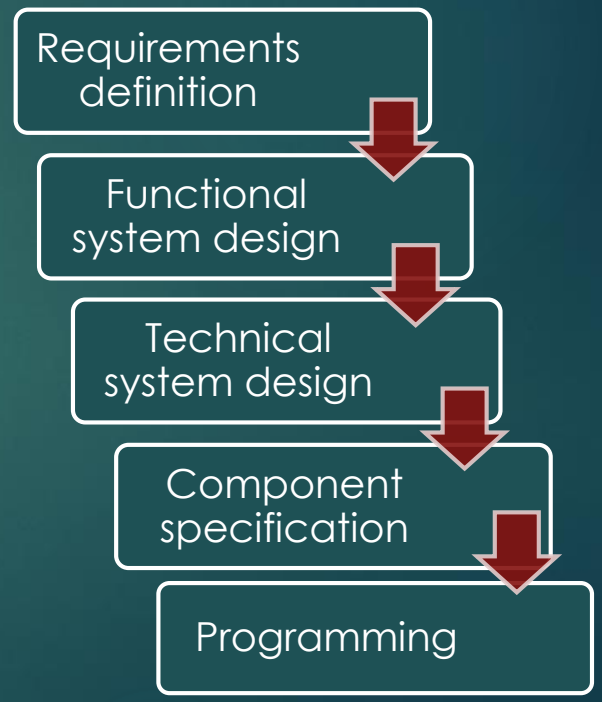
STATIC TECHNIQUES

- **Review Objectives**
  - Reviews are done in order to improve product quality
    - Reviews are used to verify the correct **transition** from one **phase** to the **next phase**, as defined in the **left** half of the **V-model.**
  - Detecting defects **early** saves **costs**
  - During reviews, the **following**

defects might be detected:

  - defects in the **specifications**
  - defects in the **design** and architecture
  - defects in the **interface** specifications
  - insufficient **maintainability** (e.g. comments)
  - deviations from agreed standards

(e.g. programming guides)



STATIC TECHNIQUES

- **Advantages and Drawbacks of Reviews**
- **Advantages**
  - **Lower costs** and a relatively **high savings** potential
  - **Defects in documentation** are detected and corrected early
  - High **quality documents** improve the development process
  - Increased rate of **communication** / exchange of know-how
- **Drawbacks**
  - Stress may arise if the **author is confronted** directly
  - **Experts** involved in reviews need to attain specific product **knowledge**, good preparation is necessary
  - Considerable time investment **(10%-15%** of the overall budget)
  - **Moderator** / participants influence review **quality** directly

Outlines

1. Static Techniques and the Test Process
2. Review Process
3. Static Analysis by Tools

REVIEW PROCESS

- **Activities of a Formal Review / 1**
- **Planning**
  - Defining the review criteria (checklists, type of review)
  - Selecting the personal (reviewers, moderator, ...)
  - Allocating roles and time in project schedules (who is doing what)
  - **Defining the entry and exit criteria for formal review types**
    - Selecting which parts of documents to review (depending on **the importance or complexity**)
- **Kick-off**
  - Distributing documents (to the reviewers)
  - Explaining the objectives, process and documents (checklists)
- **Individual preparation**
  - Reviewers inspect objects, note items in need of clarification

REVIEW PROCESS

- **Activities of a Formal Review / 2**
- **Review meeting**
  - Meeting of review members, reviewers present their results
  - Discussion or logging, with documented results
  - Noting detects, making recommendations regarding handling the defects, making decisions about the defects
  - **Examining/Evaluation/Recording**
    - During any physical meetings/tracking any group electronic communications
- **Rework**
  - Author fixes any defects addressed by inspectors
  - Recording updated status of defects
- **Follow up**
  - Checking that defects have been addressed/gathering metrics
  - Decision to have a second review meeting if necessary
  - **Checking on exit criteria** (formal review types) to give the OK

REVIEW PROCESS

- **Roles and Responsibilities / 1**
- (Project-) **Manager**
  - initiates the review, decides on participants
  - allocates time in project schedules
  - determines if the review objectives have been met
- **Moderator**
  - leads the meeting / the discussion, mediates, summarizes
  - planning the review / running the review / following-up afterwards
  - upon whom rests the success of the review
- **Author**
  - writer or with chief responsibility for the review-object
  - exposes himself to the criticism, performs recommended changes

REVIEW PROCESS

- **Roles and Responsibilities / 2**
- **Reviewers** (also: inspectors or checkers)
  - individuals with a specific technical or business background
  - discover defects, deviations, problem areas, etc.
  - they represent different perspectives and roles in the review process
  - should take part in any review meetings

- **Scribe** (also: recorder)
  - documents all issues, problems and open points that were identified during the meeting
  - for important reviews the protocol will be prepared together perhaps with a projector, you will have a verified protocol afterwards

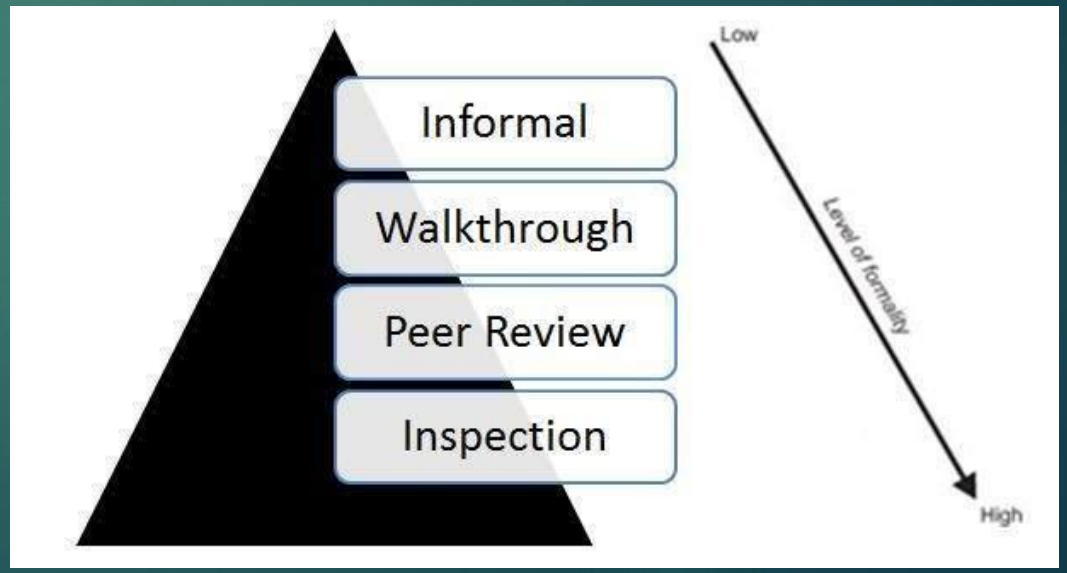
Reviews are looking at software products or relates work products from different perspectives. Checklists can make reviews more efficient. A checklist with the typical problems may help to uncover previously undetected issues.

REVIEW PROCESS

- **Roles and Responsibilities / 3**
- **Checklist - Example** (depends on the review-object)
  - Identify Target Audience
  - Create a Logo for your software
  - Create Website to promote the software
  - Define Minimum System Requirements
  - Optimize Website for Search Engines (SEO)
  - Set up Order Pages on your site
  - Create Key Benefits List
  - The terms are correct and unique
  - The requirements are complete and unique
  - The comments are "speaking" and in the expected language
  - All variables in the coding begins with a "V"
  - All constants in the coding will be named in capitals
  - The interfaces are consistent and in the same metric
  - ....

REVIEW PROCESS

- **Types of reviews (IEEE 1028) / 1**
- The basic process for a review - as outlined here - applies to the following **variants of reviews:**
  - Inspection, walkthrough, technical review, informal review
  - These variants differ in a few aspects from the general outlined base practice



REVIEW PROCESS

- **Types of reviews**
- **a) Informal Review –** No formal process, Undocumented, Pair programmer or a technical lead reviewing designs and code, inexpensive way to get some benefit.

- **b) Walkthrough –** Meeting led by the author, 'scenarios, dry runs, peer group', open-ended sessions., A step by step presentation by the author in order to gather information and to establish a common understanding, Main purpose: learning, gaining understanding, defect finding.
  - Example for using walkthroughs
    - Walkthrough of documents
    - Walkthrough of drafts for user interfaces
    - Walkthrough of business process modeling data

REVIEW PROCESS

- **Types of reviews**
- **c) Technical Review –** Documented, Defined defect detection process, Ideally lead by trained moderator, Involved peers and technical experts, No management participation, Purpose: discuss, make decisions, find defects, solve technical problems and check conformance to specifications and standards
- **d) Inspection –** Lead by trained moderator not the author, Uses entry and exit criteria, Usually peer examination. defined roles, includes matrices, formal process, pre-meeting preparation, formal follow-up process, Inspection Led by trained moderator (not the author), usually peer examination, defined roles, includes metrics, formal process, pre-meeting preparation, formal follow-up process. Main purpose: find defects.
  - Reviewers inspect the object under review using **checklists and metrics** (e.g. problems per page)
  - A trained, independent **moderator** is leading the review
- Note: walkthroughs, technical reviews and inspections can be performed within a peer group-colleague at the same organization level. This type of review is called a "peer review"

REVIEW PROCESS

- **Success Factors of a Review / 1**
  - Reviews are to be performed **goal oriented**, i.e. deviation in the reviewed object should be stated in an unbiased manner
  - The author of the reviewed object should be motivated in a **positive manner** by the review („your document will be better still" instead of „your document is of poor quality")
  - Systematic usage of the introduced **techniques** and templates
  - Using **check lists** or roles will improve the efficiency of a review
  - Sufficient **budget** is needed to perform proper reviews **(10% to 15%** of the overall development cost)
  - Make use of the lessons learned effect, use feed back to implement a continuous **improvement** process
  - Training in review techniques especially for the more formal
  - **Management** supports a good review process
  - The review is conducted in an atmosphere of trust
  - Testers are valued reviewers who contribute to the review and also learn about the product which enables them to prepare tests earlier
  - The right people for the review objectives are involved
  - There is an emphasis on learning and process improvement

Outlines

1. Static Techniques and the Test Process
2. Review Process
3. Static Analysis by Tools

STATIC ANALYSIS BY TOOLS

- **Terminology and Definitions**
- **Static analysis (Definition):**
  - Static analysis is the task of analyzing a test object (e.g. source code, script, requirement) without executing the test object.
  - Possible **aspects to be checked** with static analysis are:
    - programming rules and **standards**
    - program design **(control flow** analysis)
    - use of data **(data flow** analysis)
    - **complexity** of the program structure

    (metrics, e.g. the cyclomatic number)

STATIC ANALYSIS BY TOOLS

- **General aspects /1**
- All test objects must have a **formal structure**
  - This is especially important when using testing tools
  - Very often documents are not generated formally
  - In practice, modelling, **programming** scripting **languages** comply to the rule, some diagrams as well
- The tool-based static analysis of a program is performed with less effort than an inspection
  - Therefore, very often a static analysis is performed before a review takes place
  - To find missing and erroneous logic (potentially infinite loops)
  - To discover overly complicated constructs, security vulnerabilities, inconsistent interfaces etc.

STATIC ANALYSIS BY TOOLS

- **General aspects /2**
- The value of static analysis is the prevention of defects:
  - To find defects as early as possible before test execution
  - To warn about suspicious aspects of the code
  - To detect discrepancy in the design by the calculation of metrics like high complexity measure
  - This defects may b not find easily by dynamic testing
  - Detecting inconsistencies and dependencies
  - To check the maintainability of code or design

STATIC ANALYSIS BY TOOLS

- **General aspects /3**
- Tools to be used are compilers and analyzing tools (analyzers)
  - Compiler
    - Detect syntax errors inside a program source code
    - Create reference data of the program (e.g. cross reference list, call hierarchy, symbol table) of the program
    - Check for consistence between types of variables
    - Find undeclared variables and unreachable (dead) code
  - Analyzer address additional aspects, such as
    - Conventions and standards
    - Metrics of complexity
    - Object coupling

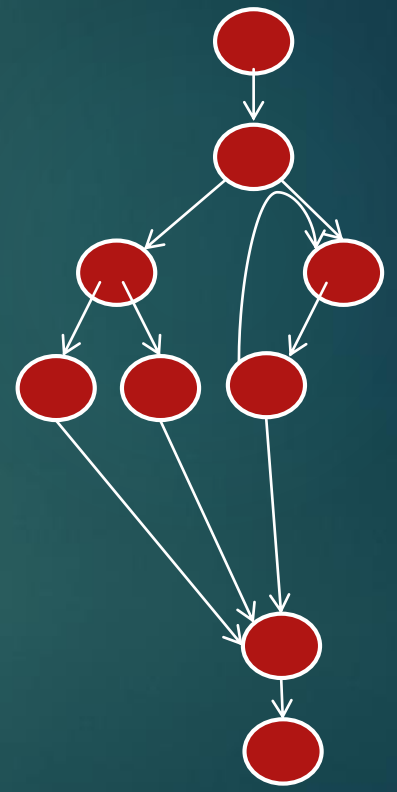
STATIC ANALYSIS BY TOOLS

- **Control flow analysis /1**
- **Aim**
  - To find defects caused by wrong construction

of the program code (dead branches, dead code etc.)

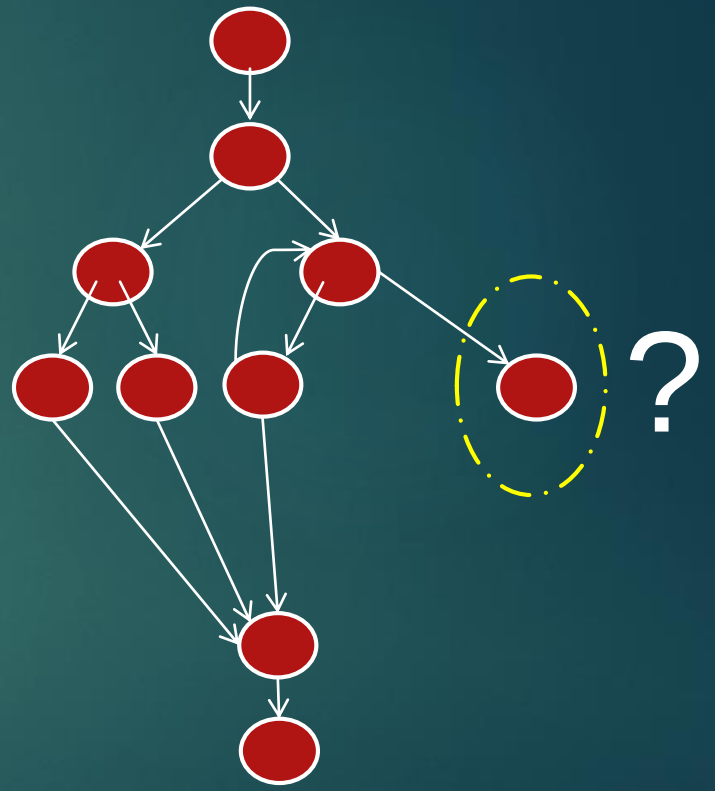
- **Method**
  - Code structure is represented as a control

flow graph
  - Directed graph
    - Nodes represent statements or sequences of statements
    - Edges represent control flow transfer, as in decisions and loops
    - Tools based construction



STATIC ANALYSIS BY TOOLS

- **Control flow analysis /2**
- **Results**
  - easy understandable overview or program code
  - anomalies* can easily be detected, defects stick out
    - loops exited by jumps
    - dead branches
    - multiple returns
  - a control flow graph is just a simplified version of a flow chart



*Anomaly: an irregularity or inconsistency

STATIC ANALYSIS BY TOOLS

- **Data flow analysis /1**
- **Aim:**
  - To discover data flow anomalies with help of control flow graphs and sensible assumption of data flow sequences
- **Benefits:**
  - Reliable detection of data flow anomalies
  - Exact locating of errors can be determined easily
  - A good supplement for other testing methods
- **Drawbacks:**
  - Limited to a narrow range of error types

STATIC ANALYSIS BY TOOLS

- **Data flow analysis - Method**
  - A variable x may have the following different states during program execution:
    - x is undefined **(u):** no value is assigned to x
    - x is defined (**d**)**:** a value is assigned to x (e. g. x = *1*)
    - x is referenced (**r**)**:** a reference is taken, the value of x does not change (e.g. if (x > *0*) or a=b+x)
  - The data flow of a variable can be expressed as a sequence of the states: **d**, **u**, and **r,** e.g. x -> **u d r d r r u**
  - If one of these sequences contains a sub sequence which does not make sense, a data flow anomaly is identified:
    - **ur -** anomaly: undefined value gets referenced
    - **du** - anomaly: defined value gets undefined before reading
    - **dd-** anomaly: defined value gets defined again before reading

STATIC ANALYSIS BY TOOLS

- **Data flow analysis /3**
- **Data flow anomaly example**

For this example, the values of two variables are exchanged via an auxiliary variable if they are *not* sorted by value:

```
void MinMax (int Min, int Max)
      {
      int Help;
            if (Min>Max)
                  {
                        Max = Help;
                        Max = Min;
                        Help = Min;
                  }
       End MinMax.
```

The data flow analysis shows:

- Variable Help is still undefined when it gets referenced: **ur**-anomaly
- Variable Max gets defined twice without any reference in between: **dd**-anomaly
- Defined value for Help gets undefined (program ends) without reference: **du**-anomaly

This example can be corrected easily:

```
Void MinMax (int Min, int Max)
      {
       int Help;
            If (Min>Max)
                  {
                        Help = Max;
                        Max = Min;
                        Min = Help;
                  }
       End MinMax.
```

STATIC ANALYSIS BY TOOLS

- **Metrics and their computation**
  - Using metrics, certain aspects of program quality can be measured
    - The metric has only relevance for the measured aspect
  - The static complexity of the program can be measured
    - Currently, there are about 100 different metrics available
  - Different metrics address different aspects of program complexity
    - Program size (e. g. Lines of Code - LOC)
    - Program control structures (e.g. cyclomatic number)
    - Data control structures (e.g. Halstead-Metric)
  - **It is difficult to compare different metrics**, even if they address the same attribute of the program!

STATIC ANALYSIS BY TOOLS

- **Metrics and their implication / 1**
- **Cyclomatic number v(G)**
  - Metric to measure the static complexity

  of a program based on its control flow graph
  - Measures linear independent program

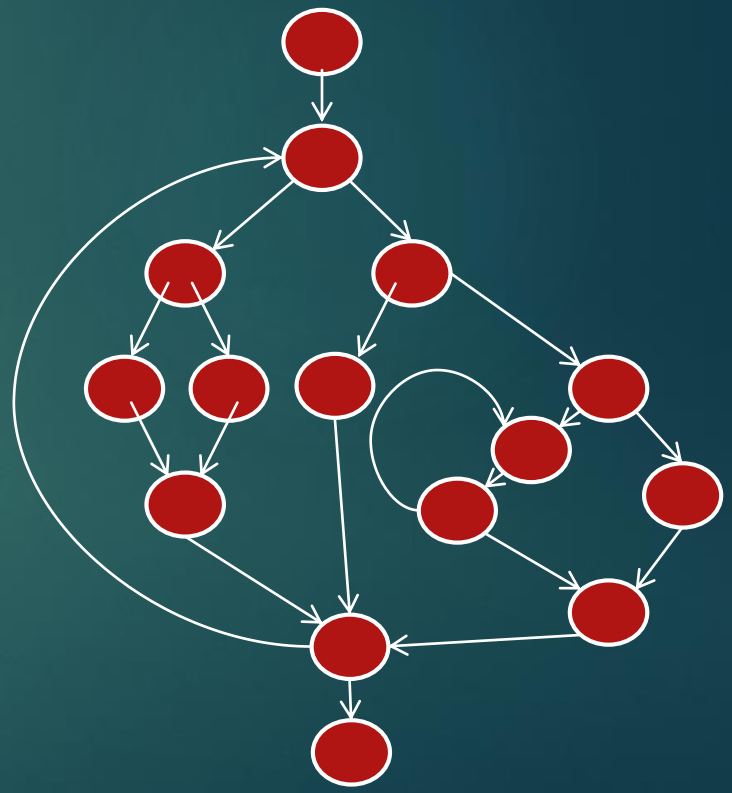
  paths, as an indication for testability

  and maintainability
  - The cyclomatic numbers made up of
    - Number of edges e
    - Number of nodes n
    - the number of disconnected parts of the graph/ independent program

    points p (mostly 1)
  - Values up to 10 are acceptable. Beyond this,

  code should be reworked/improved (best practice, McCabe)

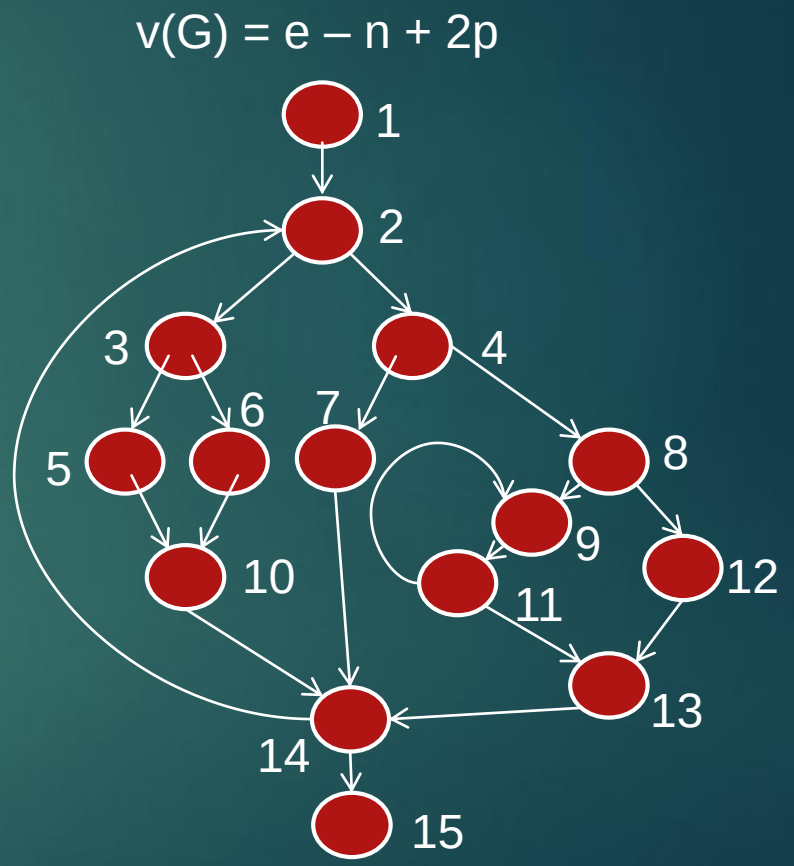
v(G) = e – n + 2p



STATIC ANALYSIS BY TOOLS

- **Metrics and their implication / 2**
- **Example III**

  (Cyclomatic number)
  - The example to the right has
    - 1 program part: p = 1
    - 15 nodes: n = 15
    - 20 edges: e= 20
  - This leads to a cyclomatic number of
    - *v(G) = e-n + 2p*
    - ***v(G) = 7***



STATIC ANALYSIS BY TOOLS

- **Metrics and their implication / 3**
- **Cyclomatic number (by McCabe) - implication**
  - The cyclomatic number can be used as a target value for code reviews
- Alternatively one may calculate Cyclomatic Complexity using decision point rule, Decision points +1. If both ways of calculation give different results, it may be due to
    - A superfluous branch
    - A missing branch
  - The cyclomatic number also gives an indication for the number of necessary test cases (to achieve decision coverage)

STATIC ANALYSIS BY TOOLS

- **Summary**
- **Static Analysis**
  - Using tools for static analysis (compiler, analyzers), program code can be inspected **without** being executed.
  - Using **tools,** the **static analysis** of a program can be performed with **less effort** than an **inspection.**
- **Analysis results**
  - The **control flow diagram** shows the program flow and allows for the detection of "dead branches" and unreachable code
  - Data anomalies are found using **data flow analysis**
  - **Metrics** can be used to assess the structural complexity, leading to an estimation of testing efforts