Chapter-III
Static Techniques

Chapter III – Static Testing Techniques

- III/01 Reviews and the test process
- III/02 Review process
- III/03 Tool based static analysis

Chapter III – Static Testing Techniques

- **III/01 Reviews and the test process**
- III/02 Review process
- III/03 Tool based static analysis

Basic Approach

- **Static** testing techniques summarize various methods, that **do not execut e** the component or the system that is being tested.
- Static tests include:

  - **review** (manual activity)

  - **static analysis** (mostly tool based activity)
- Static techniques component dynamic methods

  - static tests find **defects** rather than **failures**

  - **concepts** are inspected as well, not only executable code

  - defects / deviations are found in an **early phases**, before they are impl emented in the code

  - Static tests might find defects not found in **dynamic** testing
- **High quality documents** lead to high quality product

  - Even if the reviewed specifications do not contain any errors, interp reting the specification and creating design could be faulty

Review Objectives

- Review are done in order to improve product quality
  - Review are used to verify the correct transition from on e phase to the next phases, as defined in the left half of t he V-model.
- Detecting errors early saves costs
- During reviews, the following error must be detected:

  - errors in the specification

  - errors in the design and architecture

  - errors in the interface specifications

  - deviations from agreed standards

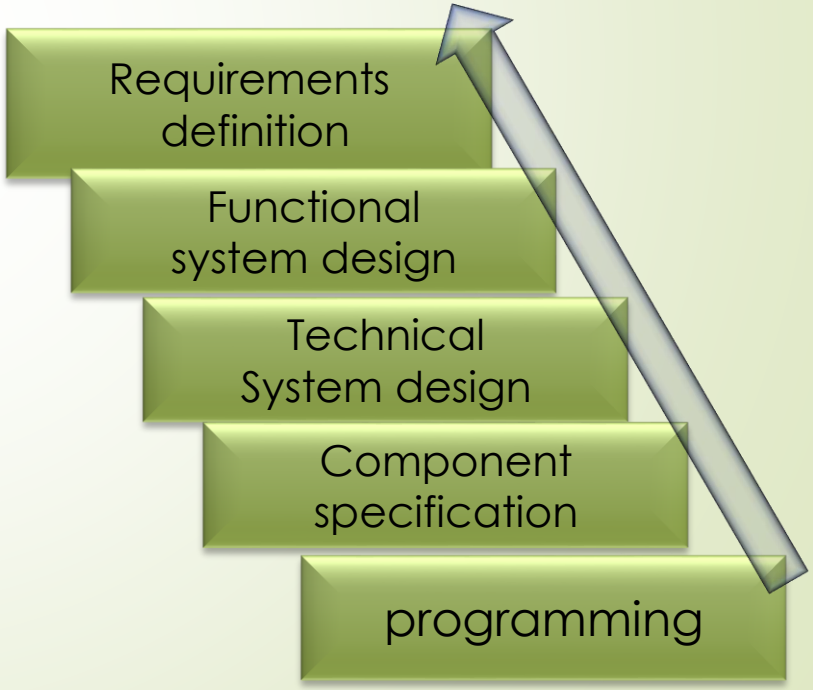
    (e.g. programming guides)

Review Objectives

- Review are done in order to improve product quality
  - Review are used to verify the correct transition from one phase to th e next phases, as defined in the left half of the V-model.
- Detecting errors early saves costs
- During reviews, the following error must be detected:

  - errors in the specification
  - errors in the design and architecture
  - errors in the interface specifications
  - deviations from agreed standards

  (e.g. programming guides)



Advantages and Drawbacks of Reviews

- **Advantages**
  - **Lower costs** and a relatively **high saving** potential
  - **Errors in documentation** are detected and corrected early
  - High **quality documents** improve the development process
  - Increase 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 **know ledge**, good preparation is necessary
  - Considerable time investment (**10% - 15%** of the overall budget)
  - **Moderator** / participants influence review **quality** directly

Chapter III – Static Testing Techniques

- III/01 Reviews and the test process
- **III/02 Review process**
- III/03 Tool based static analysis

Phases of a Review

1. **Planning phases**
   Organizing the review, select and supplementary information
2. **Organizational preparation** (and-kick-off)
   Hand out of review objects and supplementary information
3. **Individual preparation**
   Reviewers inspect objects, note item in need of clarification
4. **Review meeting**
   Meting of review members, reviewers present their results
5. **Rework**
   Author fixes any defects addressed by inspectors
6. **Follow up**
   The review meeting protocol is distributed to the manager, stating object under test, participants, roles, recommendation

Roles and responsibilities

- (Project-) Manager
  - Initiates the review, decides on participants and allocates resources
- Moderator
  - Runs the meeting / the discussion, mediates. sum marizes
- Author
  - exposes himself to the criticism, performs recom mended changes
- Reviewer (also: inspector or checkers)
  - Discover defects, deviations, problem areas etc
- Scriber (also: recorder)
  - Documents all issues, problems and open points that were identified

Types of reviews (EEE 1028) /1

- The basic process of 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

- A further distinction of review is mode depending on the nature of the reviewed object: product or process

- **SW- development process** or project process

  - CMMI, IEC 12207, IPI are terms relating to **process improvement**

  - Also called **management review**, these review do not directly interface with the testing process, they are **not part of this lecture**

- Documents / **products** of the development process

  - These reviews are addressed here

Types of reviews (EEE 1028) /2

- Inspection: Key characteristics

- Formal process based on rules, uses **defined roles**
- Review inspect the object under review usin g **checklists and metrics** (e.g. problems per page)
- A trained, independent **moderator** is leadin g the review
- Review-ability of the object is assessed prior to the review
- **Formal process** for preparation, execution, documentation and follow up activities.

Types of reviews (EEE 1028) /3

- **Inspection: advantages and drawbacks**

  - Well organized formal session with clear roles
  - Needs intensive preparation and clear roles
  - Moderator and scribe are necessary
  - Main purpose: finding defects using a structured method

Types of reviews (EEE 1028) /4

Walkthrough: key characterizes

1. There is an optional pre-meeting preparation of the reviewers
2. The meeting is led by the author, he explains the object under review
3. A separate moderator is not necessary (the author moderates)
4. During preparation by the author, the reviews try to locate deviations and / or problem areas

Example for using walkthroughs

1. Walkthroughs of documents
2. Walkthroughs of drafts for user interfaces
3. Walkthroughs of business process modeling data

Types of reviews (EEE 1028) /5

Walkthrough: advantages and disadvantages

1. Little effort in preparing a review session, but it is an open-end session
2. Session can be initiated on short term notice
3. Author has a grate influence on the outcome: since she/ he moderates the review, there is a danger of domination by the author (critical points are not addressed in depth)
4. Little control possible, since author is also in charge of any follow-up actives

Types of reviews (EEE 1028) /6

Technical review: key characteristics

1. Target of examination is a technical aspect of the object under
2. review: is it fit for use?
3. Experts are needed, preferably external experts
4. The meeting might take place without the author
5. The review is done using technical specifications and other documents
6. A unanimous vote of recommendation is given by the expert panel
7. Intensive preparation is needed

Types of reviews (EEE 1028) /7

Informal review: key characteristics

1. Simplest form of reviews
2. Often initiated by the author
3. Only reviewers (one or more) will be involved
4. No separate meeting necessary
5. Results may be recorded in form of an action list
6. Often performed in such a way that a colleague is asked to review a document
7. Also called: peer review

Types of reviews (EEE 1028) /8

Informal Review: advantages and drawbacks

1. Easy to perform, even on short term notice
2. Cost effective
3. No protocol needed

Success Factors of a Review

1. Reviews are to be performed **goal oriented**, i.e. deviation in the reviewed object should be started in an unbiased manner
2. 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")
3. Systematic usages of the introduced **technique** and templates
4. Using **check lists** will improve the efficiency of a review
5. Sufficient **budget** is needed to perform proper review (**10% to 15%** of the overall development cost)
6. Make use of the lesson learned effect, use feed back to implement a continuous **improvement** process
7. Duration of review meeting (inspection): **2 hours maximum**

Summary

- During **static testing**, the test object is not executed
- Review can take place during the **early phases** of the development process, they complement/ extended the methods of dynamic testing

- **Phases of a review:**

  Planning- preparation – individual preparation – meeting – follow up

- **Roles and takes for the Review**:

  Manager – Moderator – Author – Reviewer – Scribe

- **Types of reviews:**

  Inspection – Walkthrough – Technical review – Informal review

Chapter III – Static Testing Techniques

- III/01 Reviews and the test process
- III/02 Review process
- **III/03 Tool based static analysis**

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 aspect to be checked with static analysis are**:

  1. programming rules and **standards**
  2. program design (**control flow** analysis)
  3. use of data (**data flow** analysis)
  4. **complexity** of the program structure (metrics, e.g. the cycloramic number)

General aspects /1

- **All test objects must have a formal structure**
  1. this is especially important when using testing tools
  2. very often documents are not generated formally
  3. in practice, modeling, **programming** and scripting **languages** comply to the rule, some diagrams as well

- **The tool-based static analysis of a programmed with lees effort than an inspection.**
  - therefore, very often a static analysis is performed before a review takes place

General aspects /2

Tools to be used are compliers and analyzing tools (analyzer)

- **Compiler**

1. detect syntax errors inside a program source code
2. create reference data of the program (e.g. cross reference list, call hierarchy, symbol table) of the program
3. check for consistence between types of variables
4. find undeclared variables and unreachable (dead) code

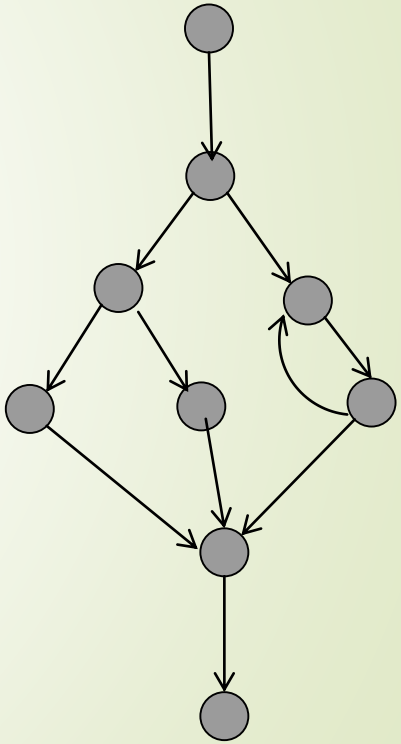
- **Analyzer address additional aspects, such as**

  - conventions and standards
  - metrics of complexity
  - object coupling

Control flow analysis /1

- **Aim**
  - To find defects caused by wrong construction of the program code (dead branches, dead code etc.)

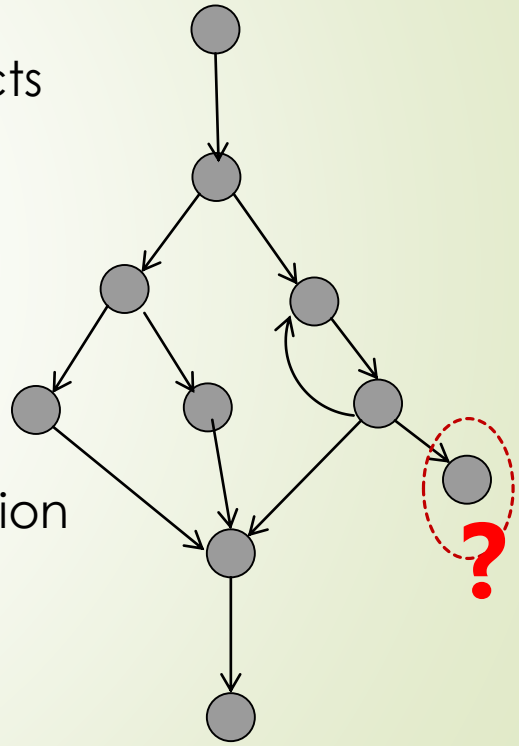
- **Method**
  1. Code structure is represented as a control flow graph
  2. Directed graph
     a. Nodes represent statements or sequences of statements
     b. Edges represent control flow transfer, as in decisions and loops
     c. Tools based construction



Control Flow analysis /2

Result

- easy understandable overview of 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 flow chart



* Anomalies: an irregularity of inconsistency

Data flow analysis /1

Aim

1. To discover data flow anomalies with help of control flow graph and sensible assumption of data flow sequences

Benefits

1. Reliable detection of data flow anomalies
2. Exact locating of errors can be determined easily
3. A good supplement for other testing methods

Drawbacks

1. Limited to a narrow range of error types

Data flow analysis – Method

- A variable x may have the following different stages 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 sequence contain 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

Data flow analysis /3

- Data flow analysis example

For this example the values of two variables are exchanged via an auxiliary variable, if they are not stored 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
}
```

Metrics and their computation

- Using Metrics, certain aspect 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 structure (e.g. cyclomatic complexity)
  - Data control structures (e.g. Halstead complexity)

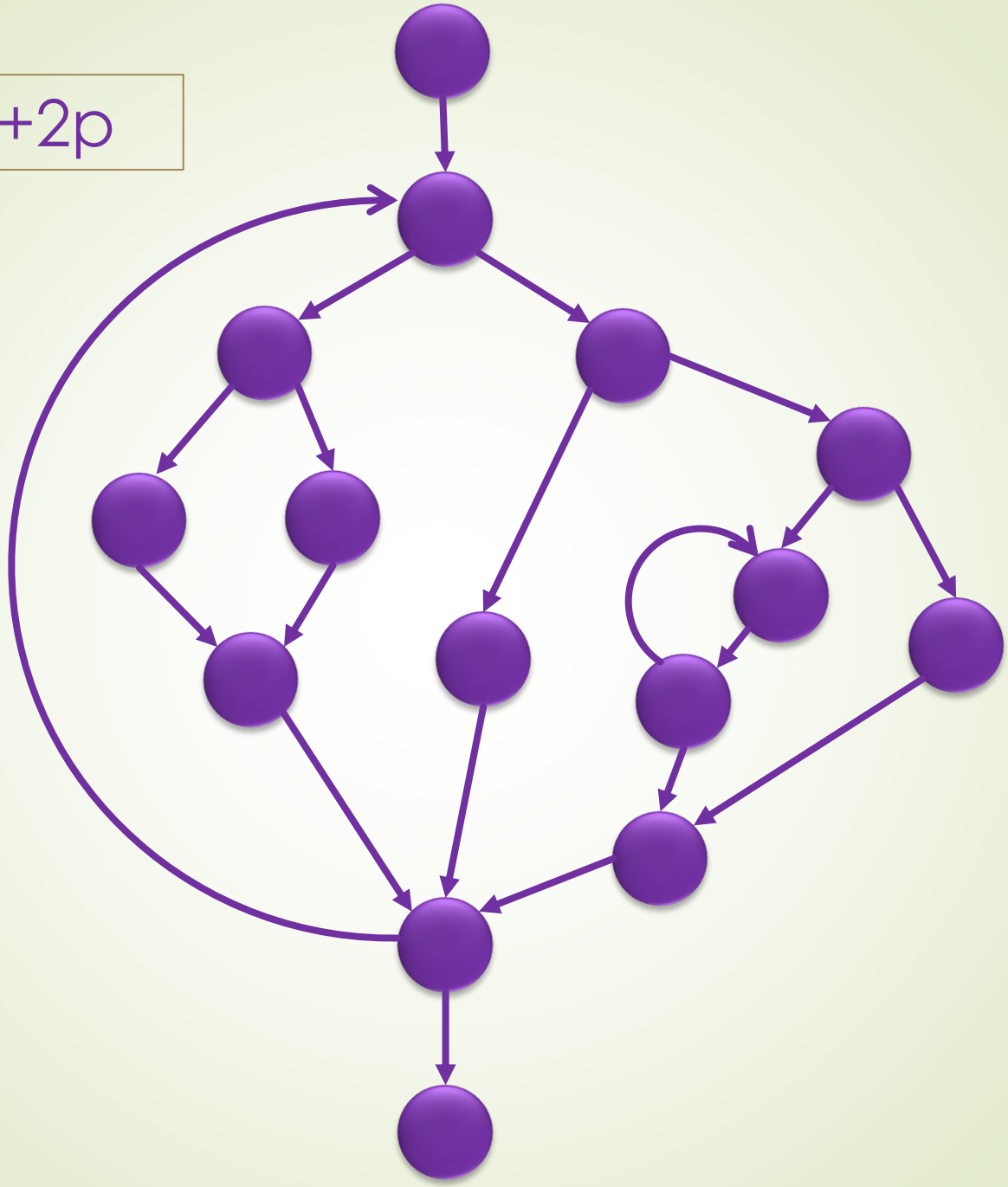
- It is difficult to compare different metrics, even if they address the same attribute of the program !

Metrics and their implementation /1

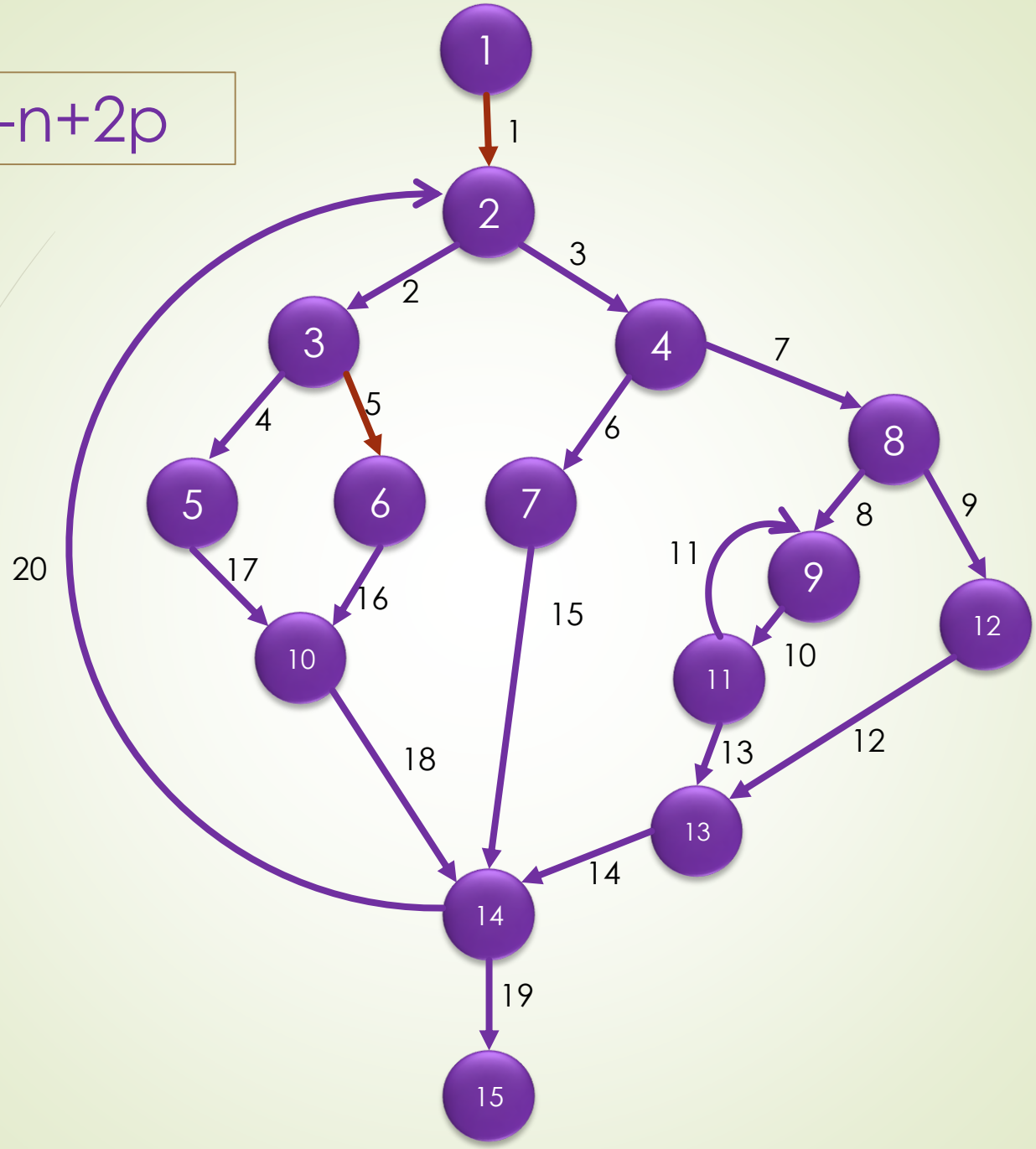
Cyclomatic number v(G)

- Metric to measure the static complexity of a program based on its control flow graph
- Measure linear independent program paths, as an indication for testability and maintainability
- The cyclomatic number is made up of
  - Number of edges e
  - Number of nodes n
  - Number of inspected independent program parts p (mostly 1)
- Values up to 10 are acceptable. Beyond this, code should be reworked/improved (best practice, McCabe)

v(G)=e-n+2p



v(G)=e-n+2p


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Metrics and their implementation /2

Cyclomatic Complexity

1 program part : p = 1
15 nodes : n = 15
20 edges : e = 20

The number to a cyclomatic complexity
v (G) = e-n+2p
v(G)=7

Metrics and their implementation /3

Cyclomatic Complexity (by Thomas J. McCabe) - implication

- The cyclomatic complexity can be used as a target value for code reviews
- The cyclomatic complexity can also be calculated as the number of independent decisions plus one. If both ways of calculation give different results, it may be due to
  - ✓ A superfluous branch
  - ✓ A missing branch
- The cyclomatic complexity also gives as indication for the number of necessary test cases (to achieve decision coverage)

Summary

Static Analysis

- Using tools for static analysis (complier, 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 access the structural complexity, leading to an estimation of testing efforts to expect

Thank You

mail4rashed@gmail.com - fb.me/rashedkarim.info - skype: rashed-karim