Software Construction and User Interfaces (SE/ComS 319)

Ali Jannesari

Department of Computer Science

Iowa State University, Spring 2019

TESTING

Administrative

- Final exam:
- https://www.registrar.iastate.edu/students/exams/springexams
- **LAGOMAR W0142 (lecture room)**

Thursday	**May 9th**	**9:45-11:45 a.m.**

Outline

- Testing (review)
- Statement coverage
- Branch coverage
- Path coverage
- GUI Testing

Testing – Motivation

- Software artifacts always contain errors
- The later a mistake is found, the more expensive it is to fix it
- The goal is to find mistakes as early as possible

Error detection is the goal of testing (1)

> *„Testing shows the presence of bugs, not their absence.“*
>
> (Edsger W. Dijkstra)

- Why do we need testing?
  - Catch bugs (defect testing)
  - Check if we follow all the requirements (Validation testing)
  - **Providing “documentation” in TDD/XP process**
- How much resources are spent in testing?

Error detection is the goal of testing (2)

- Complete testing of all **combinations** of all **input values** is **not possible** except for trivial programs (astronomical number of test cases).
- Correctness is possible only with **formal** correctness proof (correspondence of specification and program); this is only possible for small programs today.
- Central question: When can one stop testing to look for errors? (**Test completeness criteria**)
  - When is sufficient?
  - Exhaustive testing

Error detection is the goal of testing (3)

- Attention:

Differences:

- Testing procedure → Detect error
- Verifying procedure → Prove correctness
- Analyzing procedure → Determine properties of a system component

There are 3 types of errors ...

- A **failure** (*fault)* is the deviation of the behavior of the software from the specification (an event).
- A **defect** (*bug)* is a deficiency in a software product, which can lead to failure (a condition).
- It is said that a defect manifests itself in failure.
- A **mistake** is a human action that causes a defect (a process).

Error

Test doubles (Types of test auxiliary/assistant)

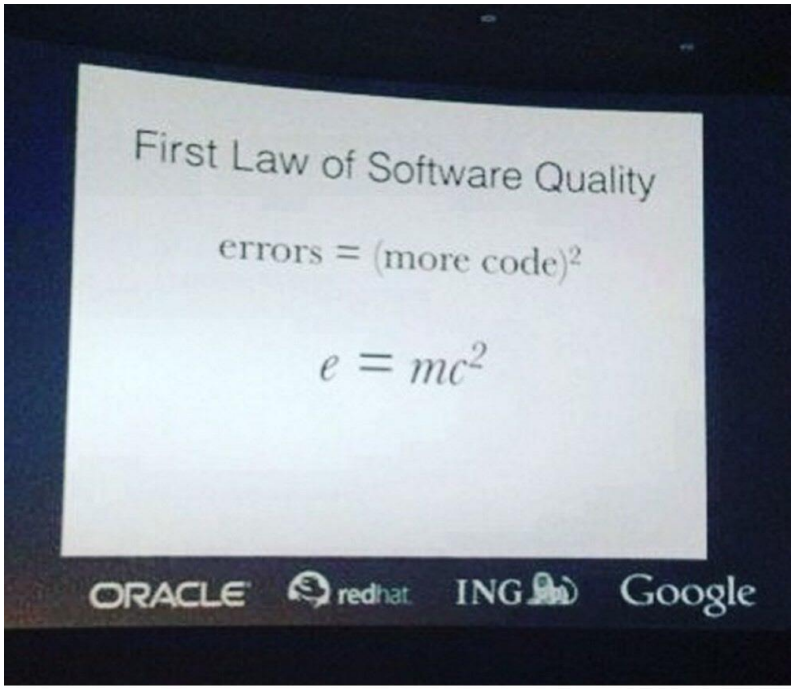
- A ***stub*** is a rudimentary part of the implemented software and serves as a placeholder for yet unreacted functionality.
- A ***dummy*** simulates the implementation for test purposes
- A ***mock*** *object* is a dummy with additional functionality, such as setting the reaction of the imitation to certain inputs or checking the behavior of the "client"
- More details later (if we get time…).

Error classes (1)

- **Requirement errors** (defect in the requirements)
  - Incorrect information of user requests
  - Incomplete information about functional requirements, performance requirements, etc.
  - Inconsistency of different requirements
  - impracticability (not feasible)
- **Design errors** (defect in the specification)
  - Incomplete or incorrect implementation of the requirement
  - Inconsistency of specification or design
  - Inconsistency between requirement, specification and design

Error classes (2)

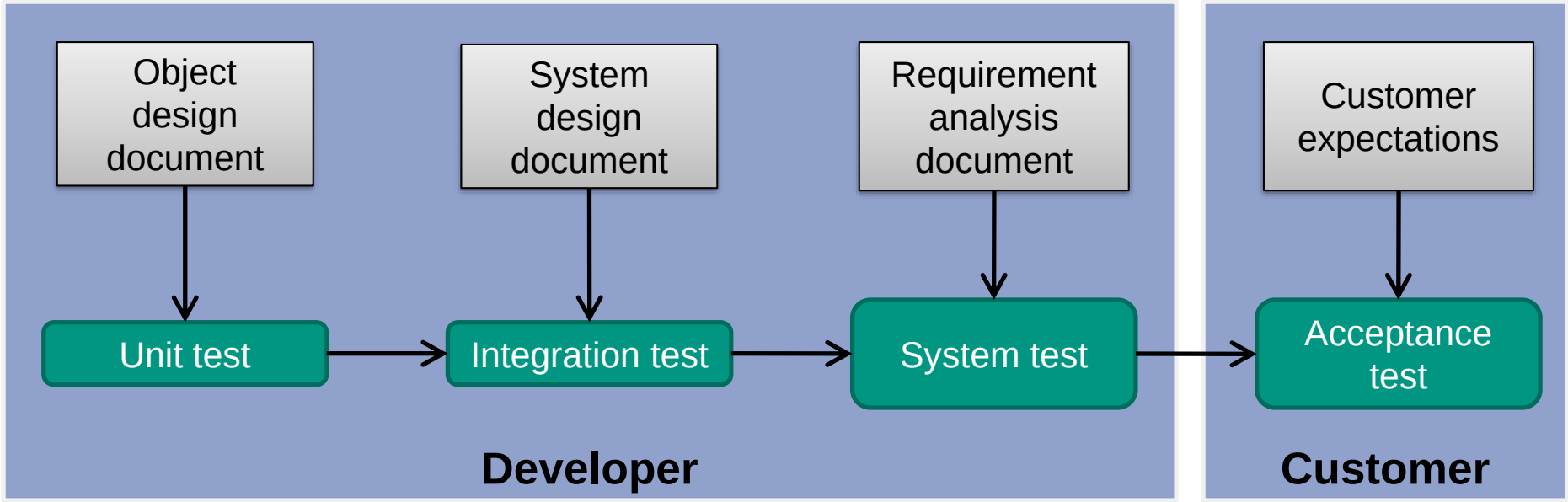
- **Implementation error** (defect in the program)
  - Incorrect implementation of the specification in a program



Module / Software testing process

- A **software test**, test for short, executes a single software component or a configuration of software components under known conditions (inputs and execution environment) and verifies their behavior (outputs and responses).
- The to be tested SW-component or configuration is called **test object** (component under test, CUT; function under test, FUT).
- A **test case** consists of a set of data for the execution of a part or the whole test object.
- A **test driver** (**test framework)** supplies test objects with test cases and initiates the execution of the test objects (interactively or automatically).

What are the different stages of testing?



The **unit test** checks the function of a single module/function by observing the processing of test data.

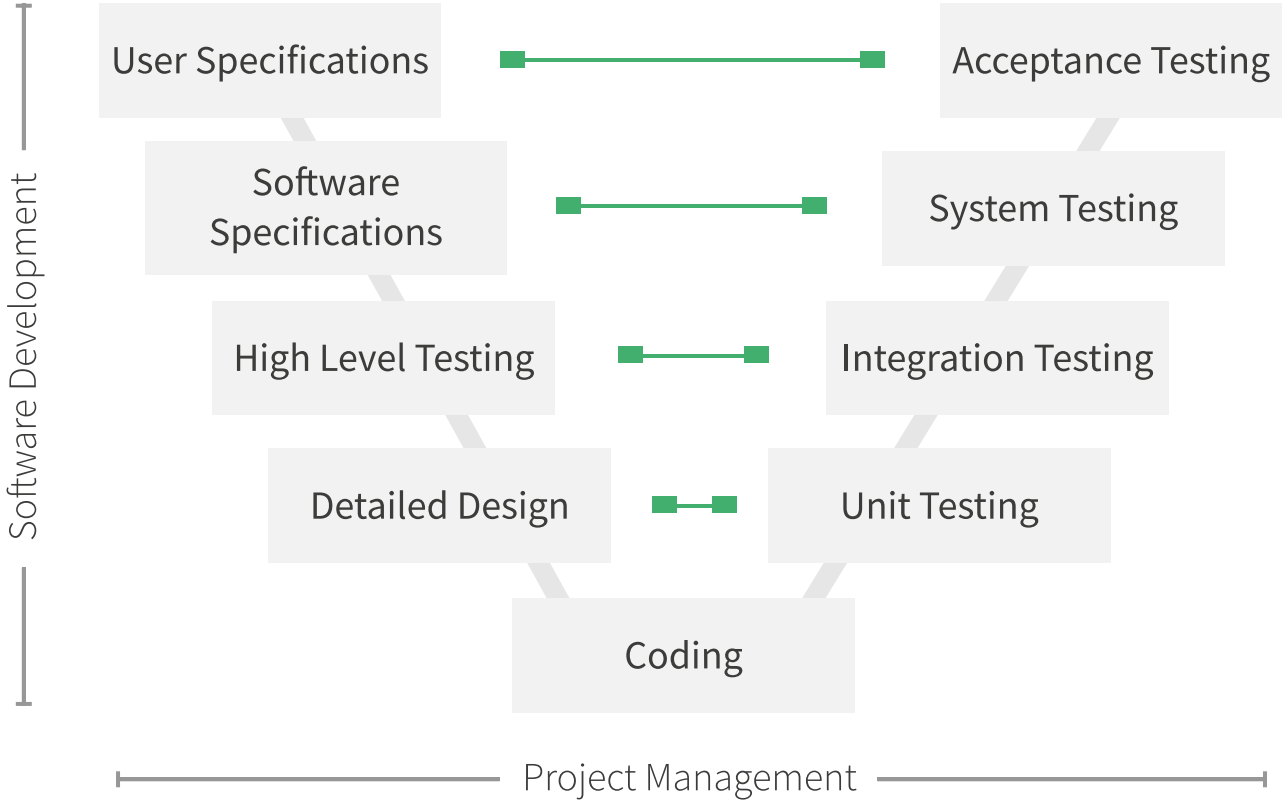
The **integration test** progressively checks the error-free interaction of system components that have already been individually tested.

The **system test** is the contractor's final test in a real (or realistic) environment without customers.

The **acceptance test** is the final test in real environment under observation, participation and / or leadership of the customer.

What are the different stages of testing?

- The V-model of software development for each stage of development:



Source: https://www.ranorex.com/resources/testing-wiki/gui-testing

Classification of testing methods (1)

- Dynamic methods

  - **Structural** tests (white / glass box testing)
    - Control flow-oriented tests
    - Data flow-oriented tests
  - **Functional** tests (black box testing)
    - Internal structure of the test object is not considered (unknown to the tester)
  - **Performance** tests (also black box)
    - Load test: Tests the system/component for reliability and compliance with the specification **within the allowed limits**
      - Can the system serve the required number of users?
    - Stress test: Tests the behavior of the system when **exceeding the defined limits**

Classification of testing methods (2)

- Static methods

  - Manual test methods (inspection, review, walkthrough)
  - Test programs (static analysis of programs)

Classification of testing methods (3)

- Dynamic methods
  - The compiled, executable program is tagged and executed with certain test cases
  - The program is tested in the real (realistic) environment
  - Sampling method: Correctness of the program is **not proven!**
- Static methods
  - The program (the component) is not executed, but the source code is analyzed

Classification of testing methods (2)

- Dynamic methods
  - The compiled, exec able program is tagged and executed with certain test cas
  - The program is tes n the rea alistic) environment
  - Sampling method: ctness of t gram is **not proven!**
- Static methods

White Box Testing:
Determining the values **with knowledge** of control and / or data flow

Black Box Testing:
Determining the values **without knowledge** of control and / or data flow; just out of specification

Overview Matrix

Phase						
Acceptance test						
System test						
Integration test						
Unit test						
Method	Control flow oriented	Data flow oriented	Functional Tests	Performance tests	Manual testing methods	Testing programs

Control flow oriented (CFO) test methods

- Statement Coverage
- Branch Coverage
- Path coverage

How to judge the quality of a test suite? – Test coverage criteria

- Completeness criteria for CFO test method are defined by "control flow graphs"
  - Definition of an intermediate language (IL)
  - Definition of the transformation into the intermediate language
  - Definition of the control flow graph
- Then: definition of the testing method and their corresponding coverage

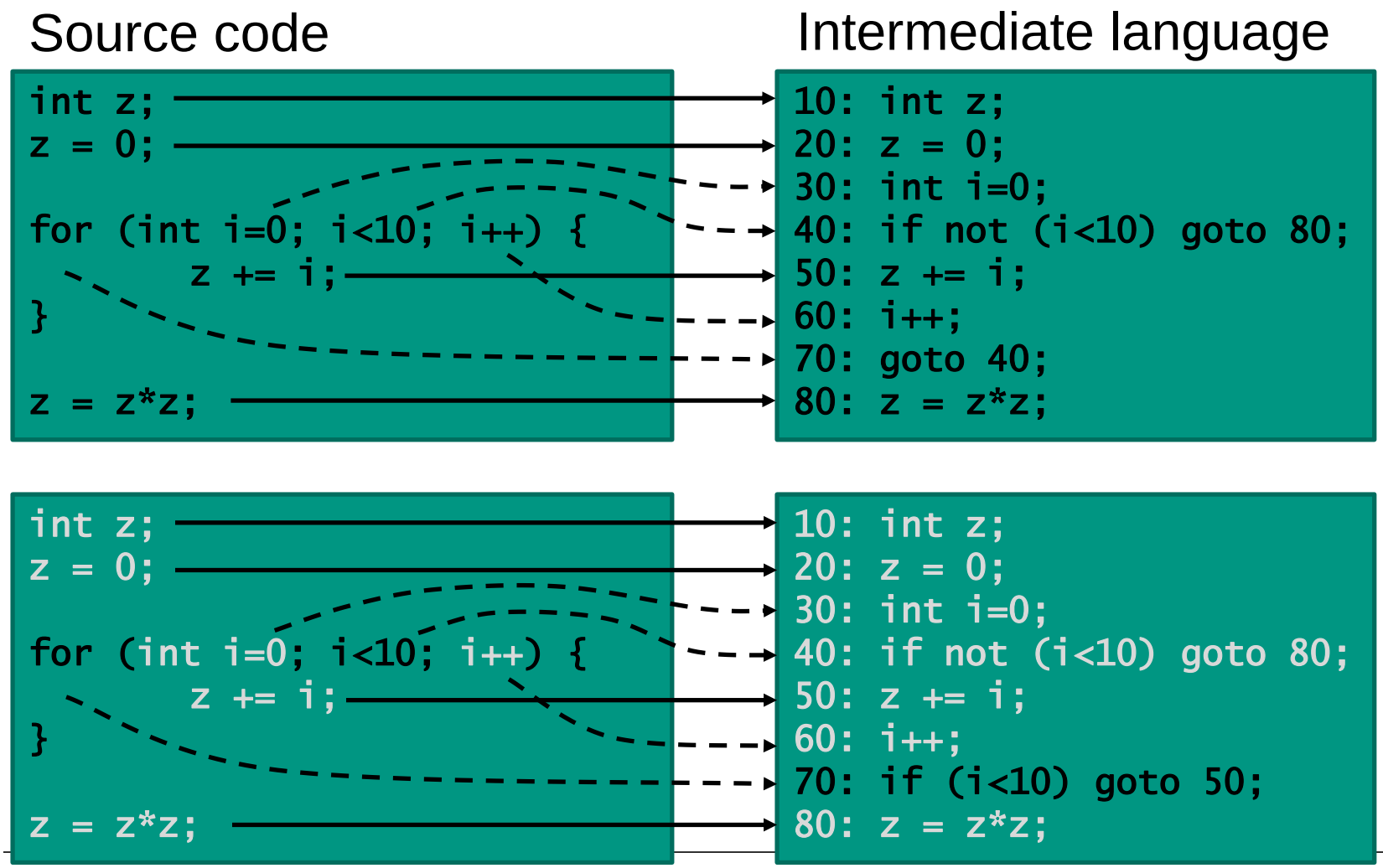
Definition: Intermediate Language (IL)

- We define an **intermediate language** consisting of:
  - arbitrary commands except those that affect the execution order (such as conditional statements jumps, loops, etc.),
  - conditional/ unconditional jump instructions (**goto**) to arbitrary but fixed positions of the instruction sequence,
  - any number of variables.
- The intermediate language is based on what is commonly understood by "assembler code".
- The implementation (especially the glossary of the commands) of this intermediate language is irrelevant here.

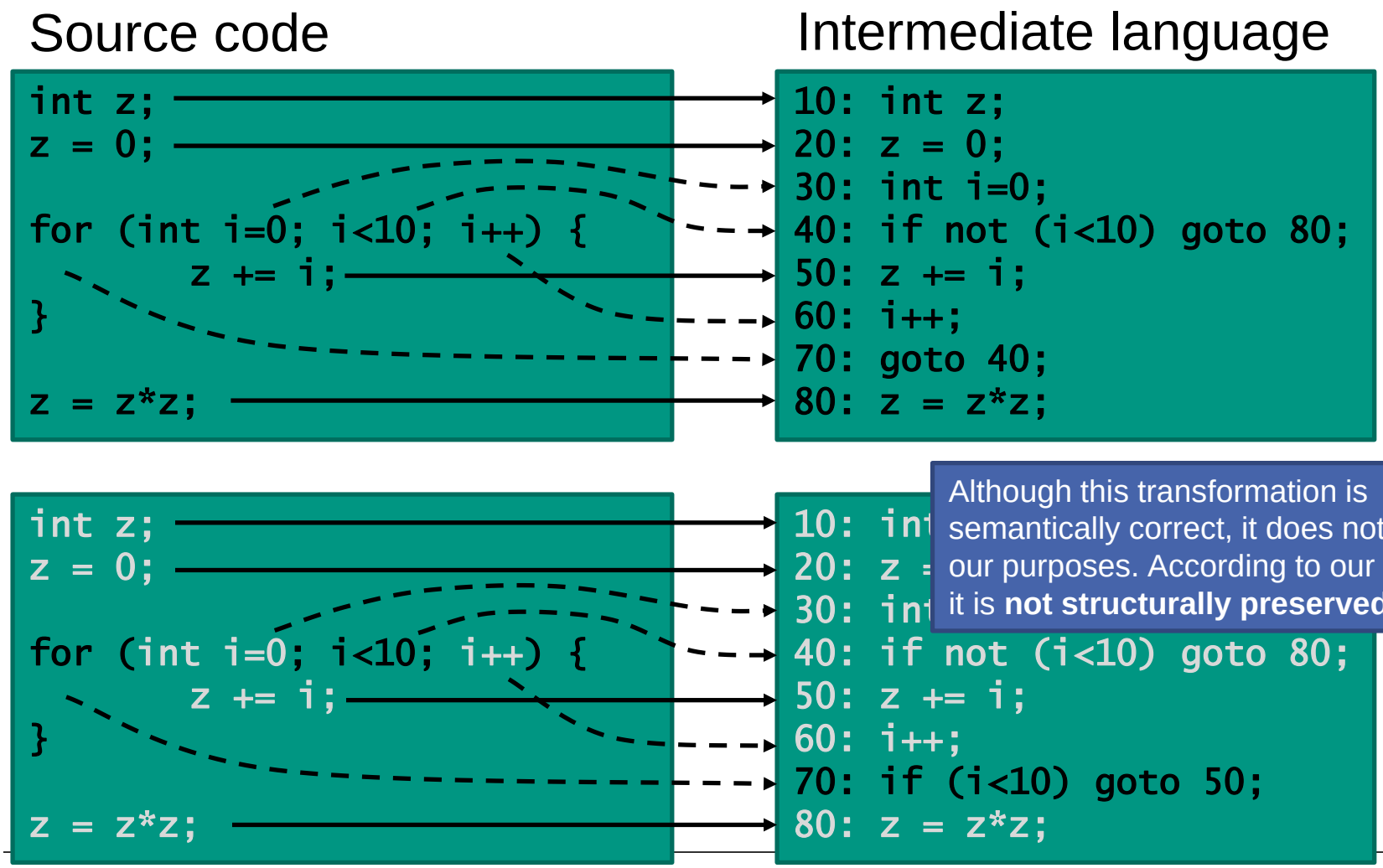
Definition: Structure-preserving transformation

- We speak of a **structure-preserving transformation** of a source language (e.g. Java) into the intermediate language, if
  - (exclusively) the commands affecting the execution order are replaced by intermediate language command sequences, where
    - the order of execution of the other commands remains the same with the same parameterization with that in the source language!
- All other commands are taken over unchanged
- Transforms are avoided where statement sequences or conditional jumps are replicated (no loop unrolling, no optimizations.)

Transformation – Example



Transformation – Example



Definition: Basic block (BB)

- A **basic block** denotes a maximum length of consecutive statements of the intermediate language,
  - in which the control flow occurs only at the beginning and
  - which contains no jump instructions except at the end.
  - In other words, has **only one entry point and one exit point**!

Example:

```
a = 10;
b = c / a;
if b > d goto basicBlock x;
m = 3 * b;
...
```

1 basic block

next basic block

Definition: Control flow graph

- A **control flow graph** of a program P is a directed graph G,

$$G = (N, E, n_{start}, n_{stop})$$

where:

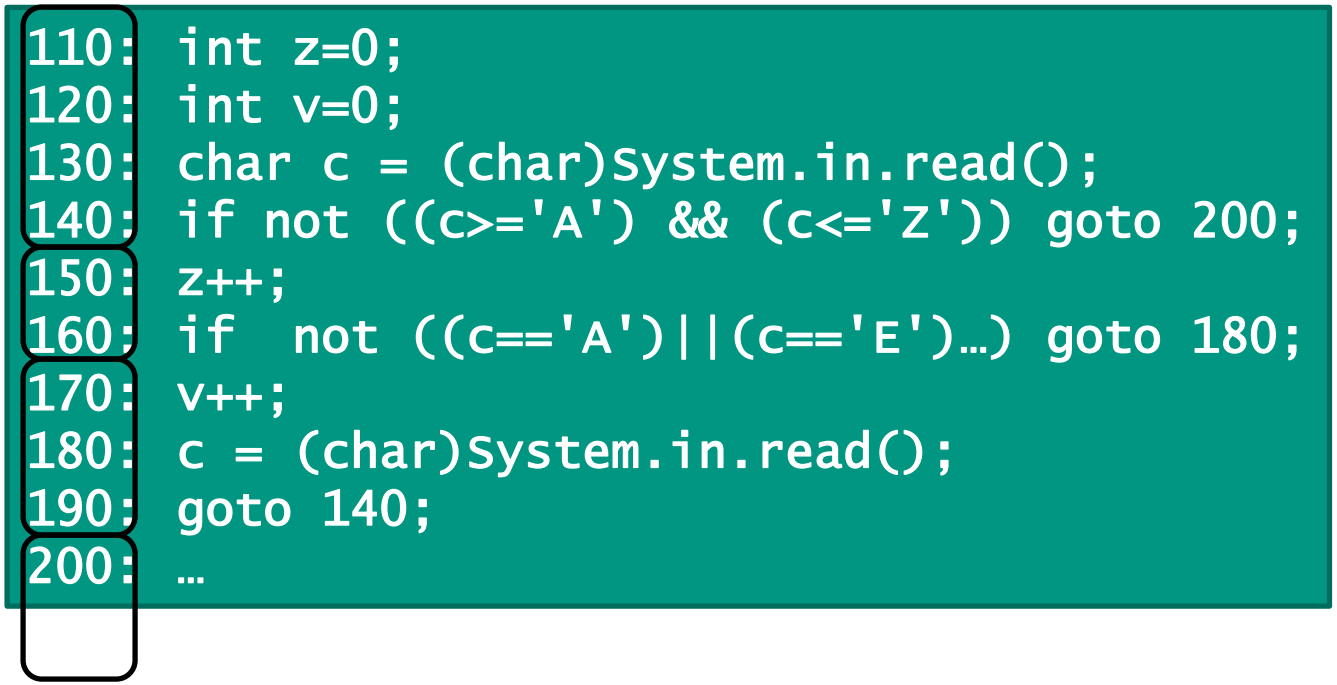
- N is the set of basic blocks in P,
- $E \subseteq N \times N$ the set of edges, where the edges indicate the execution order of two basic blocks (sequential execution or jumps)
- n_{start} the starting block and
- n_{stop} the stop block.

Find control flow graph – Example

```
…
int z=0;
int v=0;
char c = (char)System.in.read();
while ((c>='A') && (c<='Z‘)) {
  z++;
  if ((c=='A')||(c=='E')||(c=='I')||(c=='O')||(c=='U‘)) {
    v++;
  }
  c = (char)System.in.read();
}
…
```

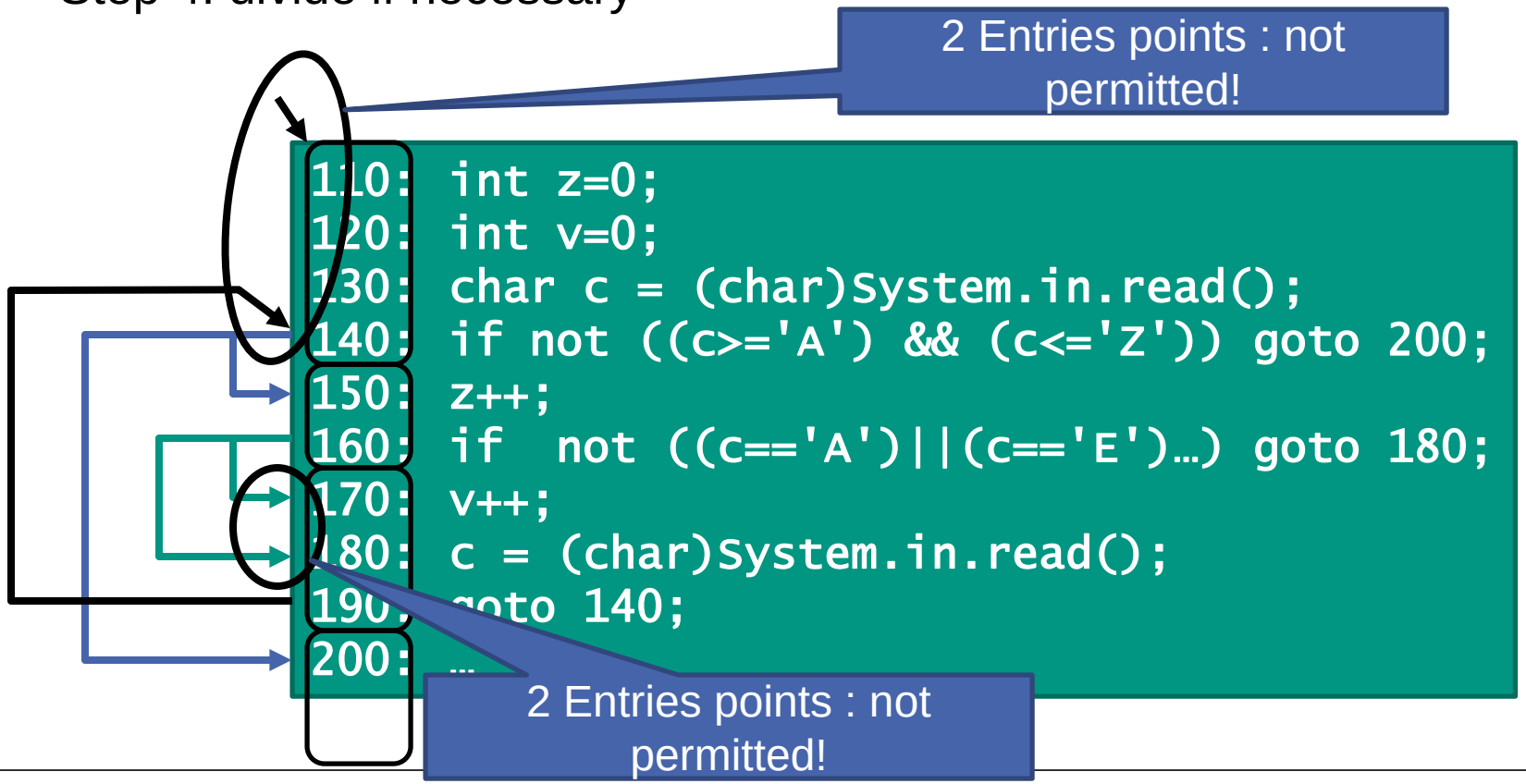
Find control flow graph – Example

- Step 1: Transform in intermediate language
- Step 2: Summarize all sequences ending in a jump into a basic block



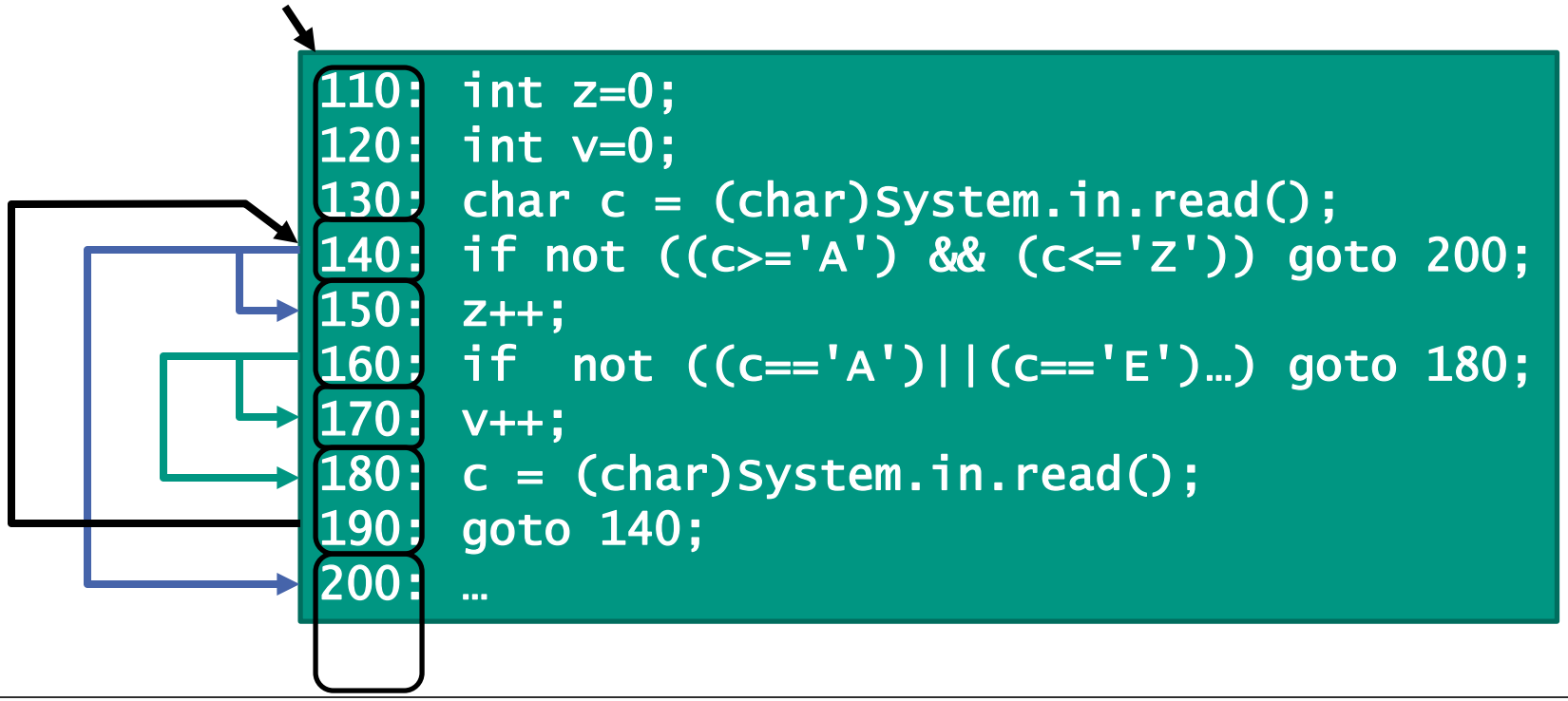
Find control flow graph – Example

- Step 3: Check if entry is only at the beginning
- Step 4: divide if necessary



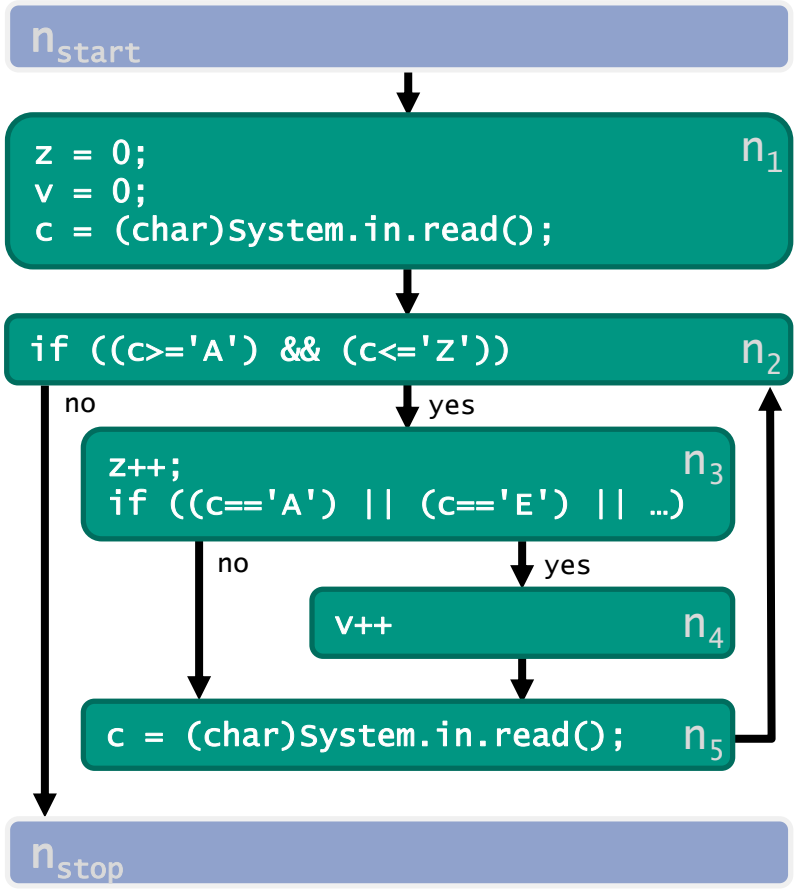
Find control flow graph – Example

- Step 3: Check if entry is only at the beginning
- Step 4: divide if necessary



Find control flow graph – Example

- Control flow graph (CFG):

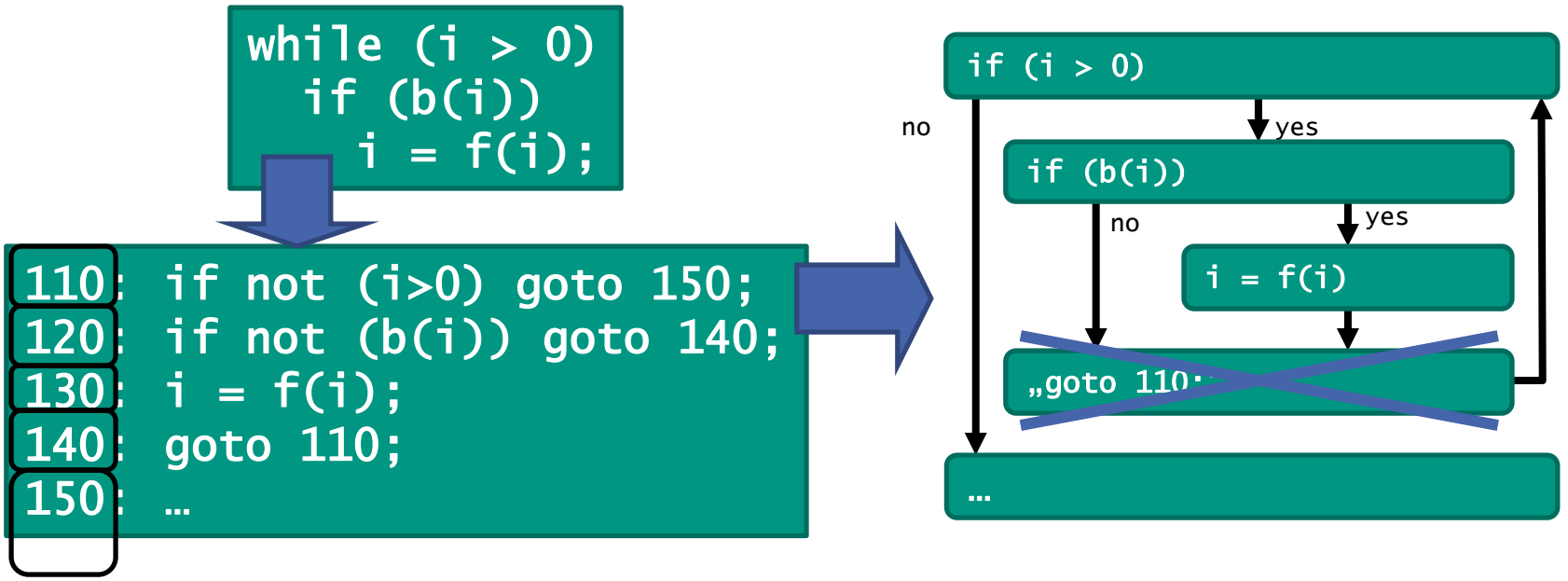


Definitions: branch, full paths

- An edge $e \in E$ in a CFG G is called a **branch**. Branches are basically directed.
- Paths in the CFG that start with the start node n_{start} and stop at the stop node n_{stop} are called **full paths**.

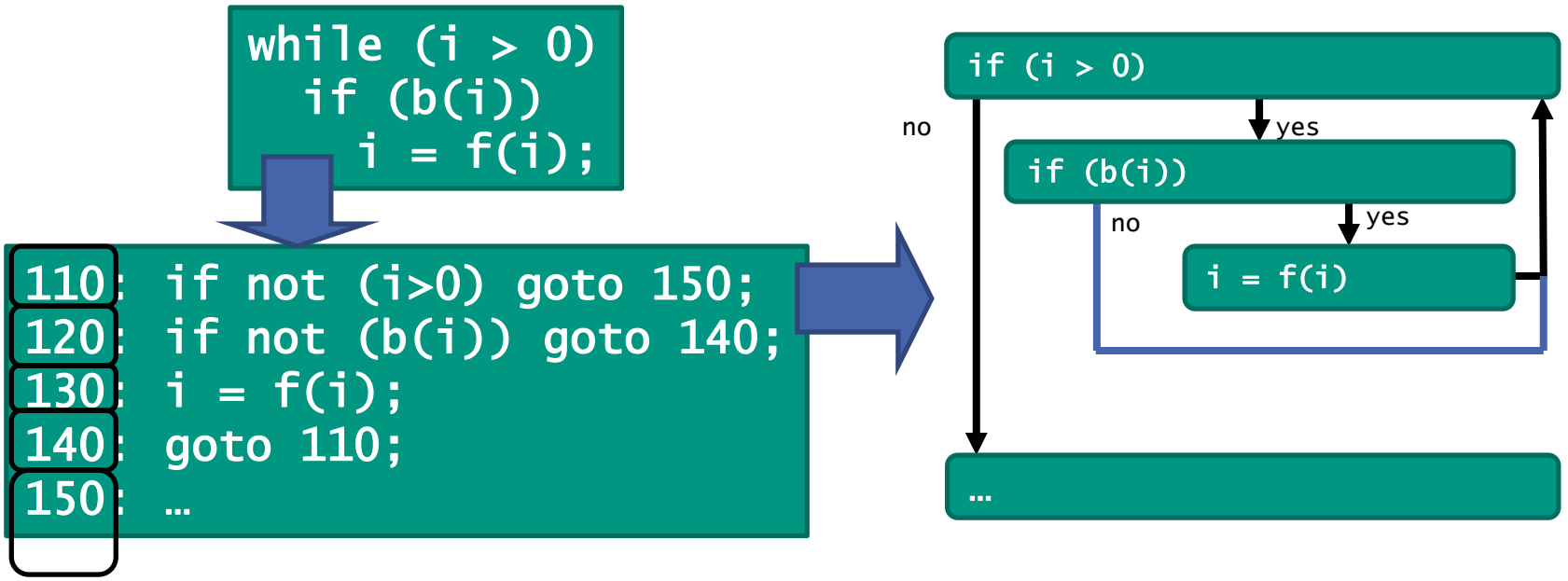
Simplify control flow graph

- If a basic block, which only contains an **unconditional jump** command, is created it can be removed (reduction of the BB).



Simplify control flow graph

- If a basic block, which only contains an unconditional jump command, is created by splitting it can be removed (reduction of the BB).



Definition: Statement coverage

- The test strategy **statement coverage** $C_{Statement}$ requires the execution of **all basic blocks** of the program P.
  - C stands for coverage
  - Metric, also called C_0

$$C_{Statement} = \frac{\text{Number of instructions passed}}{\text{Number of all instructions}}$$

  - Insufficient test criterion
  - Non-executable program parts can be found
  - Missing program parts are not detected
- Also called statement capture (a test completion criterion)

Definition: Branch coverage

- **Branch coverage** C_{branch} requires the traversal of **all branches** in the CFG.
- Metric, also called C_1

$$C_{branch} = \frac{\text{Number of traversed branches}}{\text{Number of all branches}}$$

  - Branches that are not executed can be detected
  - Neither combination of branches (paths) nor complex conditions considered
  - Loops are not tested sufficiently
  - Missing branches not testable, not detected.
- Also called branch capture.

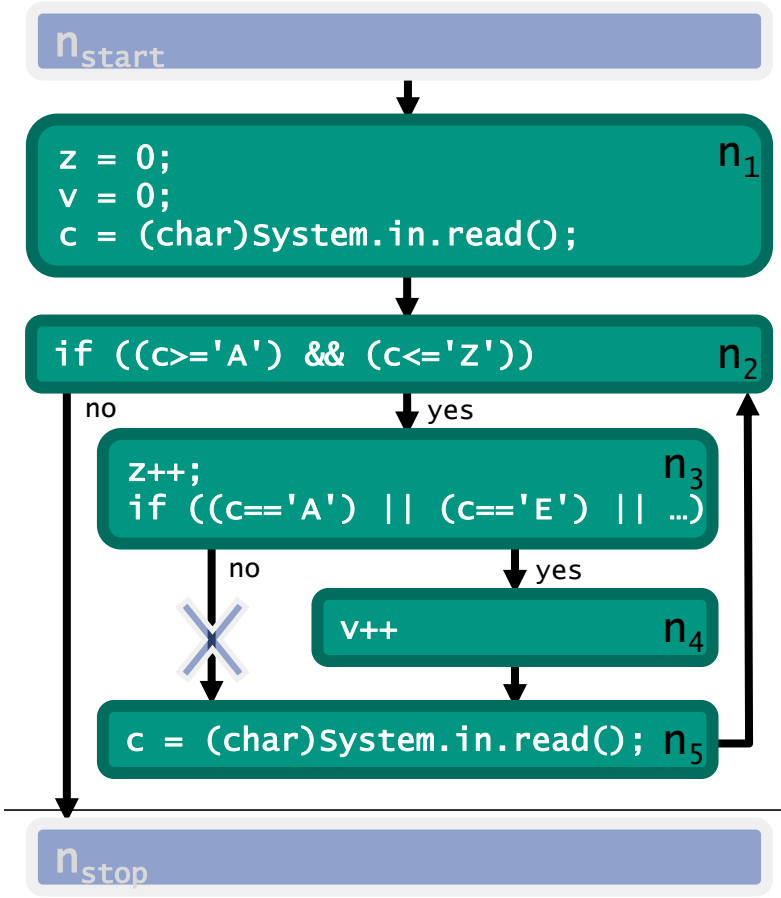
Statement coverage vs. Branch coverage

Statement coverage, all nodes

Example sequence:

(n_{start}, n_1, n_2, n_3, n_4, n_5, n_2, n_{stopp})

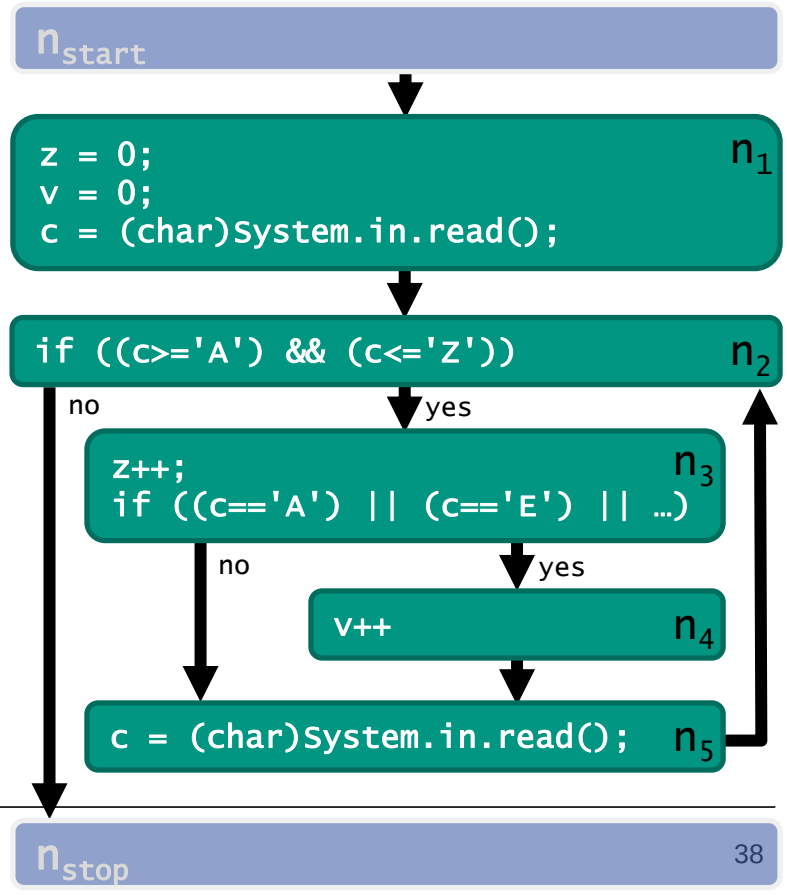
→ branch (n_3, n_5) is not executed



Branch coverage, all edges

Example sequence:

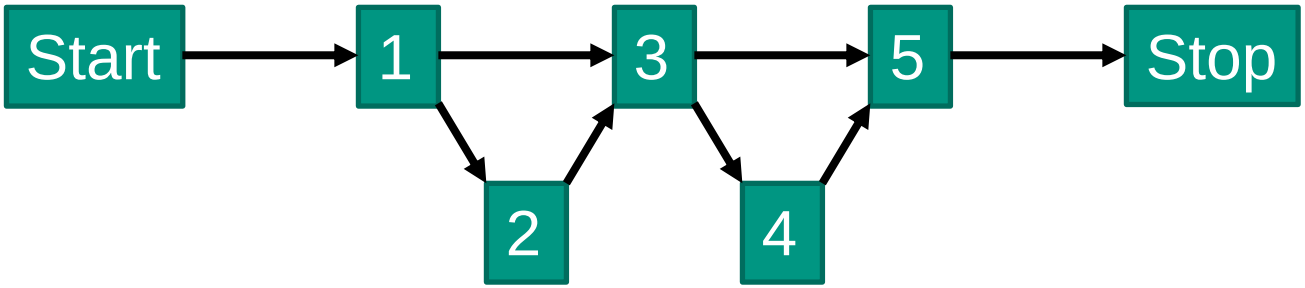
(n_{start}, n_1, n_2, n_3, n_4, n_5, n_2, n_3, n_5, n_2, n_{stopp})



Definition: Path coverage

- The **path coverage** calls for the execution of all different, full paths in the program.
  - Path number grows dramatically in loops.
  - Some paths may not be executable, due to mutually exclusive conditions
  - Most powerful CFG test strategy
  - Not practicable/feasible (Because of state explosion)

Example of statement, branch, and path coverage



- Statement Coverage

  A = {(Start, 1, 2, 3, 4, 5, Stop)}

- Branch coverage

  Z = A ∪ {(Start, 1, 3, 5, Stop)}

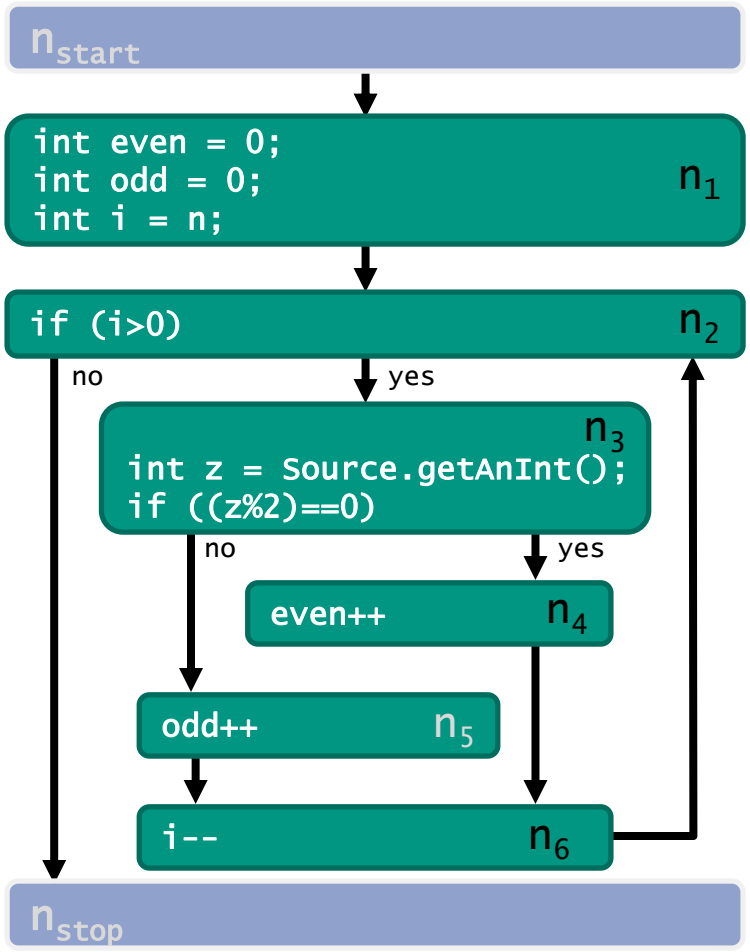
- Path coverage

  P = Z ∪ {(Start, 1, 3, 4, 5, Stop)} ∪ {(Start, 1, 2, 3, 5, Stop)}

Overhead of the path coverage – Example

```
int even = 0;
int odd = 0;
int i = n;
while (i>0) {
  int z = Source.getAnInt();
  if ((z%2)==0) even++;
  else odd++;
  i--;
}
```

n	Number of Path
0	I
1	II
2	IIII
…	…
k	2^k



Summary: CFO Test Strategies

- **Instruction coverage test** is the weakest criterion. Each statement must be executed at least once in order to have a chance to find defects in it. (if you do not execute anything at all, you will not discover any defects there!).
- **Branch coverage** subsumes statement coverage. Requires that branches be executed at least once to have a chance to detect defects in all branches.
- **Path coverage** is the most elaborate criterion, and even for small programs with loops is not feasible.
- **In general:** the different test strategies are also **test completeness criteria**. (Example: a test of the branch coverage is complete if $C_1 = 1$)

Overview Matrix

Phase						
Acceptance test						
System test						
Integration test						
Unit test	✓					
Method	Control flow oriented	Data flow oriented	Functional Tests	Performance tests	Manual testing methods	Testing programs

GUI TESTING

GUI Testing

- Refers to testing the functions of an application that are visible to a user:
  - Verifying that the application responds correctly to events such as clicking on the number and function buttons
  - Confirming that appearance elements such as fonts and images conform to design specifications
- Focuses on the critical aspects of **workflow** and **usability**
- User interface (UI) testing and GUI testing are synonyms
- GUI testing is performed from the **perspective of a user** rather than a developer
  - Analyzing an application from a user's point of view
  - Decide whether an application is ready to deploy

GUI testing techniques (1)

- **Scripted testing**: software testers design and then execute pre-planned scripts to uncover defects and verify that an application does what it is supposed to do.
  - **Example**:
    - A script might direct a tester through the process of placing a specific order on an online shopping site.
    - The script defines the entries that the tester makes on each screen and the expected outcome of each entry.
    - The tester analyzes the results and reports any defects that are found to the development team.
- Scripted testing may be performed manually or supported by test automation.

GUI testing techniques (2)

- **Exploratory testing:** Exploratory testers draw on their knowledge and experience to learn about the AUT (application under test), design tests and then immediately execute the tests (rather than following pre-written test scripts as in scripted testing!)
  - After analyzing the results, testers may identify additional tests to be performed and/or provide feedback to developers
- Same as scripted testing, exploratory testing can be completely manual, or assisted by automation

GUI testing techniques (3)

- **User experience testing:** actual end-users or user representatives evaluate an application for its ease of use, visual appeal, and ability to meet their needs.
  - The results of testing may be gathered by real-time observations of users as they explore the application on-site.
  - It Identifies defects that may be invisible to developers and testers due to their familiarity with a product
  - Could be done virtually using a cloud-based platform
- Aka **beta testing:**
  - A complete (nearly-complete) application is made available for ad hoc testing by end users at their location, with responses gathered by feedback forms

Identifying the areas to test

- Areas of the user interface to test in addition to specification documents:
  - Visual Design
  - Functionality
  - Performance
  - Security
  - Usability
  - Compliance

Identifying the areas to test – Example

- Sample areas to test the navigation for web UI applications:
  - Compatibility with all common browsers
  - Proper functioning of the page when the user clicks the back button or the refresh button
  - Page behavior after a user returns to the page using a bookmark or their browser history
  - Page behavior when the user has multiple browser windows open on the AUT at the same time.
  - …

Tools for GUI testing

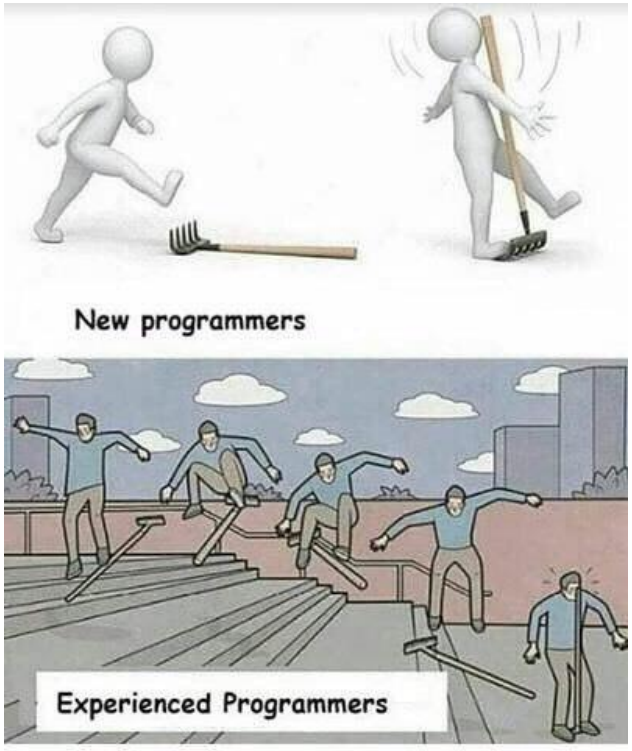
- **Selenium WebDriver**
  - https://www.seleniumhq.org/projects/webdriver/
  - Create robust, browser-based regression automation suites and tests
  - Scale and distribute scripts across many environment
  - Tutorial: https://www.guru99.com/selenium-tutorial.html
- **Jest** for UI testing
  - https://www.valentinog.com/blog/ui-testing-jest-puppetteer/
- GUI testing tools:
  - https://en.wikipedia.org/wiki/Comparison_of_GUI_testing_tools

Summary

- Testing
- CF-oriented testing (CFO)
  - Statement coverage, branch coverage, path coverage
- GUI Testing

Last but not least …

- If you want to become an experienced programmer you need to practice…



- Good Luck!

References – Testing

- [BrDu04] B. Bruegge, A.H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, Pearson Prentice Hall, 2004, S. 435ff.
- https://www.tutorialspoint.com/software_engineering/software_testing_overview.htm
- https://www.ranorex.com/resources/testing-wiki/gui-testing/