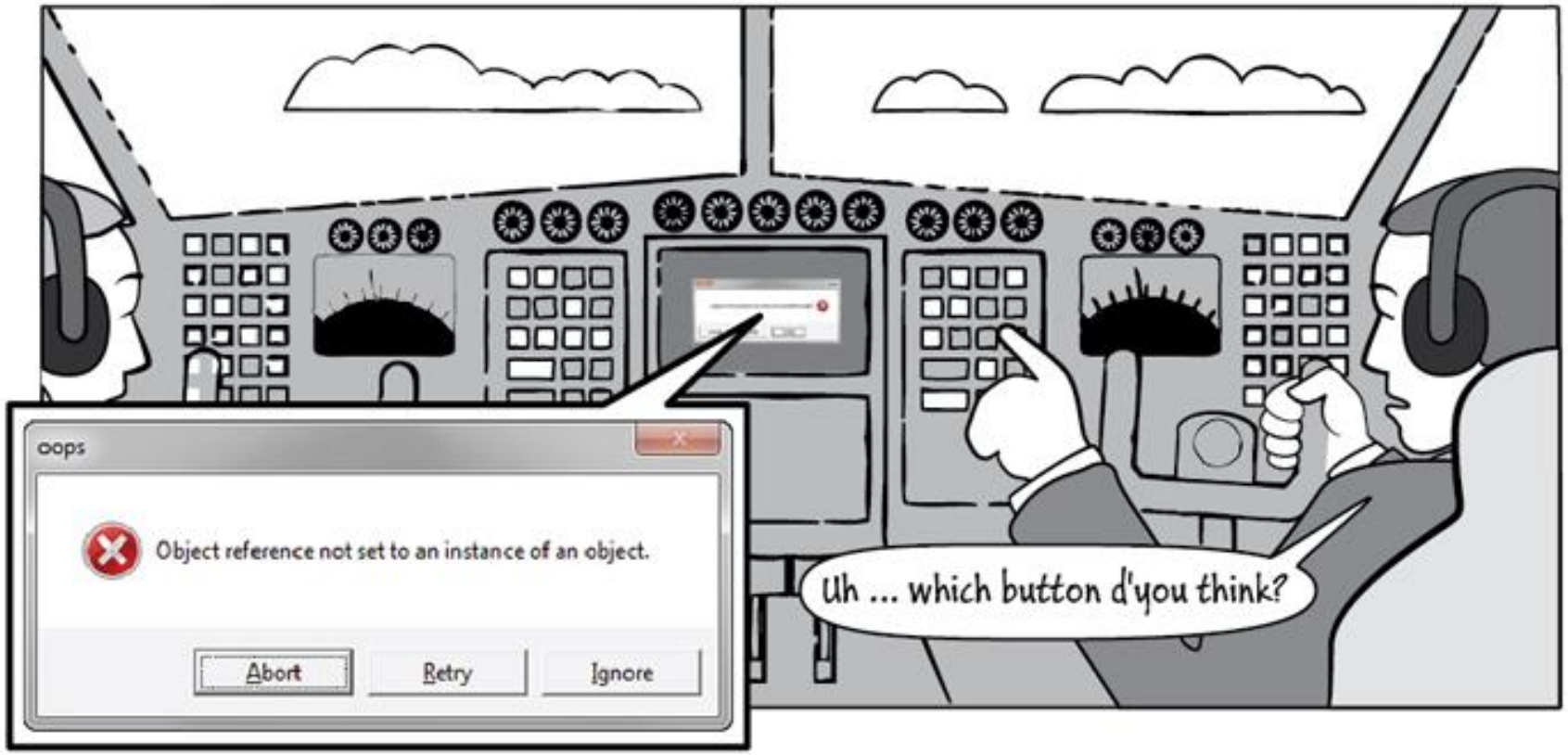
CMPT 370.3 Intermediate Software Engineering

Lecture 14 – Quality Assurance III

Dr. Zadia Codabux

Agenda

- Quality Assurance
  - Inspection/Code Reviews
  - Static Code Analysis
  - Metrics
  - White Box Testing Example
  - Test Case Design Exercise
  - Discussion - Testing at Google, Facebook, Amazon, Spotify, Microsoft



Software Inspection & Code Review

Walkthrough vs. Code Review vs. Inspection

- Walkthroughs/desk checks/code reads (informal)
  - Normally used to confirm small changes to code, say a line or two, that you have just made to fix an error.
  - Involve 2-3 people: the author of the code and the reviewer
  - In pair programming, walkthrough happens naturally

- Code Reviews (formal)
  - Performed if you've changed a substantial amount of code, or if you've added new code to an existing program.
  - Usually 3-5 attendees at a code review.

- Inspection (most formal)
  - Purpose is to find defects in a document
    - Planning documents, requirements, designs, or code, in short, any work product that a development team produces

Inspection

- Rules:
  - how many lines of code to review at once,
  - how long the review meeting must be,
  - how much preparation each member of the review team should do, etc
- Take more time and effort than walkthroughs or code reviews
- Used for mission and safety-critical software where defects can cause harm to users
- Most widely known inspection methodology – Fagan's process (1976)

Inspection Effectiveness

Faults Found	Number	Cost/fault	Total Cost
Inspections Conducted			
During design	22	1.5	33
Before test	36	6.5	234
During test	15	15.0	315
After release	3	67.0	201
			738
No Inspections Conducted			
Before test	22	6.5	143
During test	82	15.0	1230
After release	12	67.0	804
			2177

Philips, RMC 1999

Inspection Effectiveness

- **Typically catch 60% of defects**, which is higher than other techniques except prototyping and high-volume beta testing. [Schull et al., 2002]
- The combination of **design and code inspections usually removes 70-85% (or more) of defect** in a product [Jones 1996]
- Inspections **identify error-prone classes early** and Capers Jones reports that they **result in 20-30% fewer defects** than less formal review practices.
- Designers and coders learn to improve their work through participating in inspections and **increase productive by 20%** [Fagan 1976, Humphrey 1989, Gilb and Graham 1993, Wiegers 2002]
- Inspections take 10-15% of budget and typically **reduce overall costs**.

Roles During an Inspection

- **Moderator** - responsible for conducting the review. Also, distribute the design or code to be reviewed, distribute the inspection checklist, set up a meeting room, report inspection results, and follow up on the action items
- **Author** - wrote the design or code to be inspected. Present in case clarification is required.
- **Reviewer** - anyone who has a direct interest in the design or code but who is not the author. Role is to find defects.
- **Scribe** - records errors that are detected and the assignments of action items
- **Management** – not a good idea to involve management (people might feel they are under evaluation instead of the review materials). But management should have access to the inspection report.

Inspection Procedure

- **Planning**
  - Author gives design or code to moderator
  - Moderator decides who will review the material
  - Moderator schedules inspection meeting
  - Moderator distributes the design/code and a checklist
- **Overview**
  - Author provides overview to reviewer not familiar with the project
  - Can be a dangerous practice: design/code should speak for itself
- **Preparation**
  - Each Reviewer works alone to look for errors using the checklist
  - Approx 125 lines of code per hour [Humphrey 1989]
  - Reviewers may each be given a different perspective e.g., from the point of view of the maintenance programmer, the customer, or the designer

Inspection Procedure

- **Inspection Meeting**
  - Moderator chooses someone (other than the author) to paraphrase the design or read the code [Wiegers 2003]. All logic is explained.
  - The Scribe records errors and their type and severity as they are detected
  - Discussion of error stops as soon as it is recognized – no discussion of solutions
  - No more than 2 hours
- **Inspection Report**
  - Moderator produces inspection report within a day of the meeting
  - Lists each defect, including its type and severity
  - Used to develop a checklist that emphasizes problems specific to the organization
- **Rework**: Moderator assigns defects to someone (usually the author) for repair
- **Follow-up**: Moderator is responsible for seeing all rework is done
- **Third Hour Meeting**: An informal meeting to discuss solutions to problems

Effective Inspections (1/2)

- Do you have **checklists** that focus reviewer attention on areas that have been problems in the past?
- Have you focused the inspection on defect **detection** rather than correction?
- Have you considered **assigning perspectives or scenarios** to help reviewers focus on their preparation work?
- Are **reviewers given enough time** to prepare before the inspection meeting, and is each one prepared?
- Does each participant have a **distinct role** to play?
- Does the meeting move at a **productive rate**?
- Is the meeting limited to **two hours (or less)**?

Effective Inspections (2/2)

- Have all inspection participants **received specific training** in conducting inspections, and has the moderator received special training in moderation skills?
- Is **data about error types collected** at each inspection so that you can **tailor future checklists** to your organization?
- Is **data about preparation and inspection rates collected** so that you can optimize future preparation and inspections?
- Are the action items assigned at each inspection **followed up**, either personally by the moderator or with a reinspection?
- Does **management** understand that it **should not attend** inspection meetings?
- Is there a **follow-up plan** to assure that fixes are made correctly?

Inspection Metrics

- To be classified as “inspection,” metrics must be kept otherwise it is a technical review, walkthrough, or pair-programming.
- Inspections may record:
  - **Defects found in product under inspection**
  - **Where defect found in product**
  - **Time taken to perform inspection**
  - **Size of product inspected**
- Inspections may also record
  - **Category of defect: minor, major, …**
  - **What prompted detecting the defect**
  - **Suggested improvements**
  - **Failure counts collected during testing and in the field**
- Inspections use reading techniques to help focus the inspectors. Common techniques include checklists, paraphrasing and structured walkthroughs.

Sample 1

Code Review Checklist – Part I

- General
  - Does the code work? Does it perform its intended function, the logic is correct etc.
  - Is all the code easily understood?
  - Does it conform to your agreed coding conventions? These will usually cover location of braces, variable and function names, line length, indentations, formatting, and comments.
  - Is there any redundant or duplicate code?
  - Is the code as modular as possible?
  - Can any global variables be replaced?
  - Is there any commented-out code?
  - Do loops have a set length and correct termination conditions?
  - Do the names used in the program convey intent?

Code Review Checklist – Part II

- Performance
  - Are there any obvious optimizations that will improve performance?
  - Can any of the code be replaced with library or built-in functions?
  - Can any logging or debugging code be removed?
- Security
  - Are all data inputs checked (for the correct type, length, format, and range) and encoded?
  - Where third-party utilities are used, are returning errors being caught?
  - Are output values checked and encoded?
  - Are invalid parameter values handled?

Code Review Checklist – Part III

- Documentation
  - Do comments exist and describe the intent of the code?
  - Are all functions commented?
  - Is any unusual behavior or edge-case handling described?
  - Is the use and function of third-party libraries documented?
  - Are data structures and units of measurement explained?
  - Is there any incomplete code? If so, should it be removed or flagged with a suitable marker like ‘TODO’?

Code Review Checklist – Part IV

- Testing
  - Is the code testable? The code should be structured so that it doesn’t add too many or hide dependencies, is unable to initialize objects, test frameworks can use methods etc.
  - Do tests exist, and are they comprehensive?
  - Do unit tests actually test that the code is performing the intended functionality?
  - Could any test code be replaced with the use of an existing API?

Sample 2

Generic Java Code Review Checklist

1. Variable and Constant Declaration Defects
    1.1. Are descriptive identifier names used in accord with naming conventions
    1.2. Are there variables with confusingly similar names?
    1.3. Is every variable properly initialized?
    1.4. Can any non-local variables be made local?
    1.5. Are there literal constants that should be named constants?
    1.6. Are there variables that should be constants?
2. Method Definition Defects
    2.1. Are descriptive method names used in accord with naming conventions?
    2.2. Is every parameter value checked before being used?
    2.3. Does every method return a correct value at every return point?
3. Computation Defects
    3.1. Is underflow or overflow possible in any computation?
    3.2. Does any expression depend on order of evaluation of operators? Are parentheses used to avoid ambiguity?
4. Control Flow Defects
    4.1. Will all loops terminate in all cases?

Google Style Guides – Other Languages

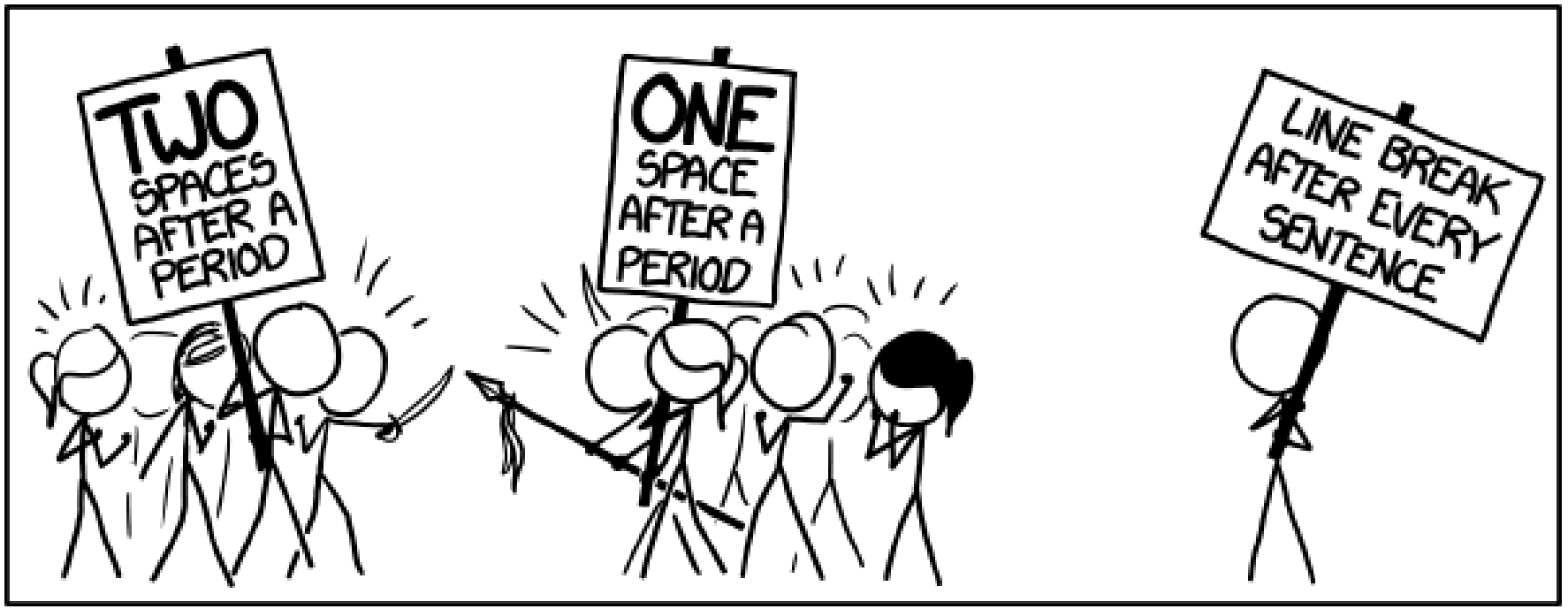
This project holds the C++ Style Guide, C# Style Guide, Swift Style Guide, Objective-C Style Guide, Java Style Guide, Python Style Guide, R Style Guide, Shell Style Guide, HTML/CSS Style Guide, JavaScript Style Guide, TypeScript Style Guide, AngularJS Style Guide, Common Lisp Style Guide, and Vimscript Style Guide. This project also contains cpplint, a tool to assist with style guide compliance, and google-c-style.el, an Emacs settings file for Google style.

https://google.github.io/styleguide/

Code Review Tips

- "**Respect others, yet don't take anything personally! We all make mistakes at some point. What's important is to learn and improve, and to treat others how *we'd* like to be treated**"
- Feedback should be actionable and concrete, not overly opinionated and theoretical.
- Explain what could go wrong and ***why*** the code needs to change; Communicate your context:
  - "We used a similar pattern in the past and it failed because of X."
  - "I recently read a blog post warning against this. Here is the link…"
  - "Here's something that helped me in the past…"

https://devblogs.microsoft.com/appcenter/how-the-visual-studio-mobile-center-team-does-code-review/



Static Code Analysis

Static Program Analysis

- Analyzes the program without running it
  - Doesn't need any test cases
  - Doesn't know what the program is supposed to do
  - Looks for violations of good programming practice
  - Looks for particular types of programming error

Static Program Analysis Tools

Linters are often static code analysis tools

- FindBugs (now Spotbugs) - https://spotbugs.github.io/
  - Originally a research project at U Maryland
  - Has large number of bug patterns

- JLint - http://jlint.sourceforge.net/
  - Developed by Konstantin Knizhnik, updated by Cyrille Artho

- PMD ("Programming Mistake Detector") - http://pmd.sourceforge.net/
  - written by Tom Copeland
  - focuses on inefficient code, e.g. over-complex expressions

- ESC/Java (Extended Static Checker for Java) -http://kind.ucd.ie/products/opensource/ESCJava2/
  - Originally developed at Compaq Research ESC/
  - Java2 is open source, managed at U College Dublin

Metrics

Metrics

- Includes methods based on using tools to count the use of features or structures in the code or other software artifacts, and compare them to standards

- Includes methods based on
  - code size (number of source lines),
  - code complexity (number of parameters, decisions, function points, modules or methods),
  - structural complexity (number or depth of calls or transactions),
  - design complexity, and so on

- Helps expose anomalous or undesirable properties that may reduce reliability and maintainability

McCabe's "Cyclomatic Complexity" Metric

If the control flow graph G of program P has e edges and n nodes, then the cyclomatic complexity v of P is

$$v(P) = e - n + 2$$

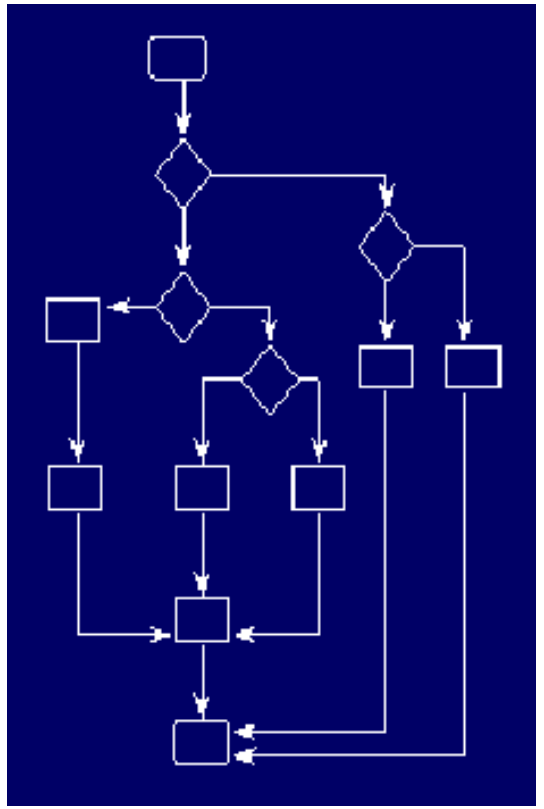
v (P) is the number of linearly independent paths in G

Example

edges = 16
nodes = 13
v (P) = 16 – 13 + 2 = 5

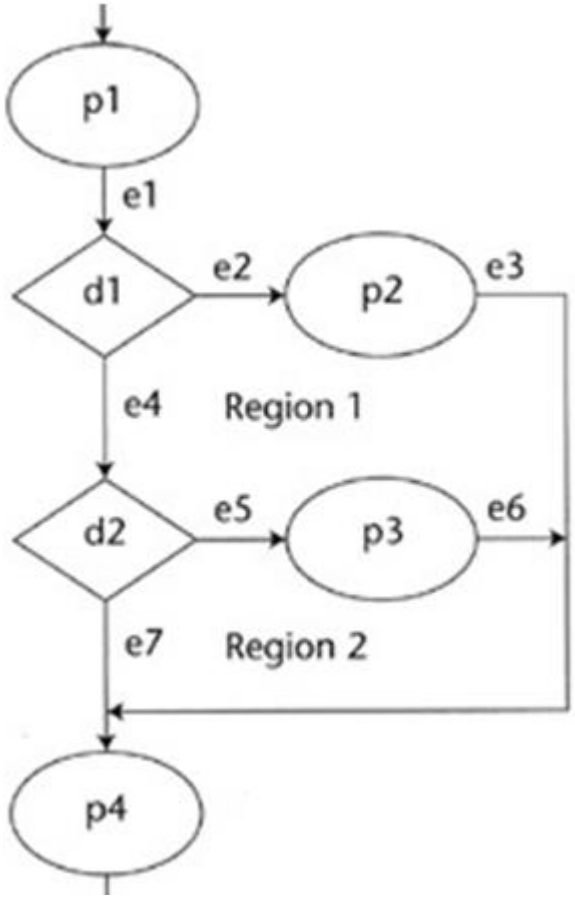
More simply, if d is the number of decision nodes in G then

$$v(P) = d + 1$$



McCabe proposed that for each module P, v (P) should be less than 10

Example



#nodes: 6
#edges: 7
#decisions : 2
#regions: 2

#Independent Paths / Cyclomatic Complexity:

#edges - #nodes + 2 = 3

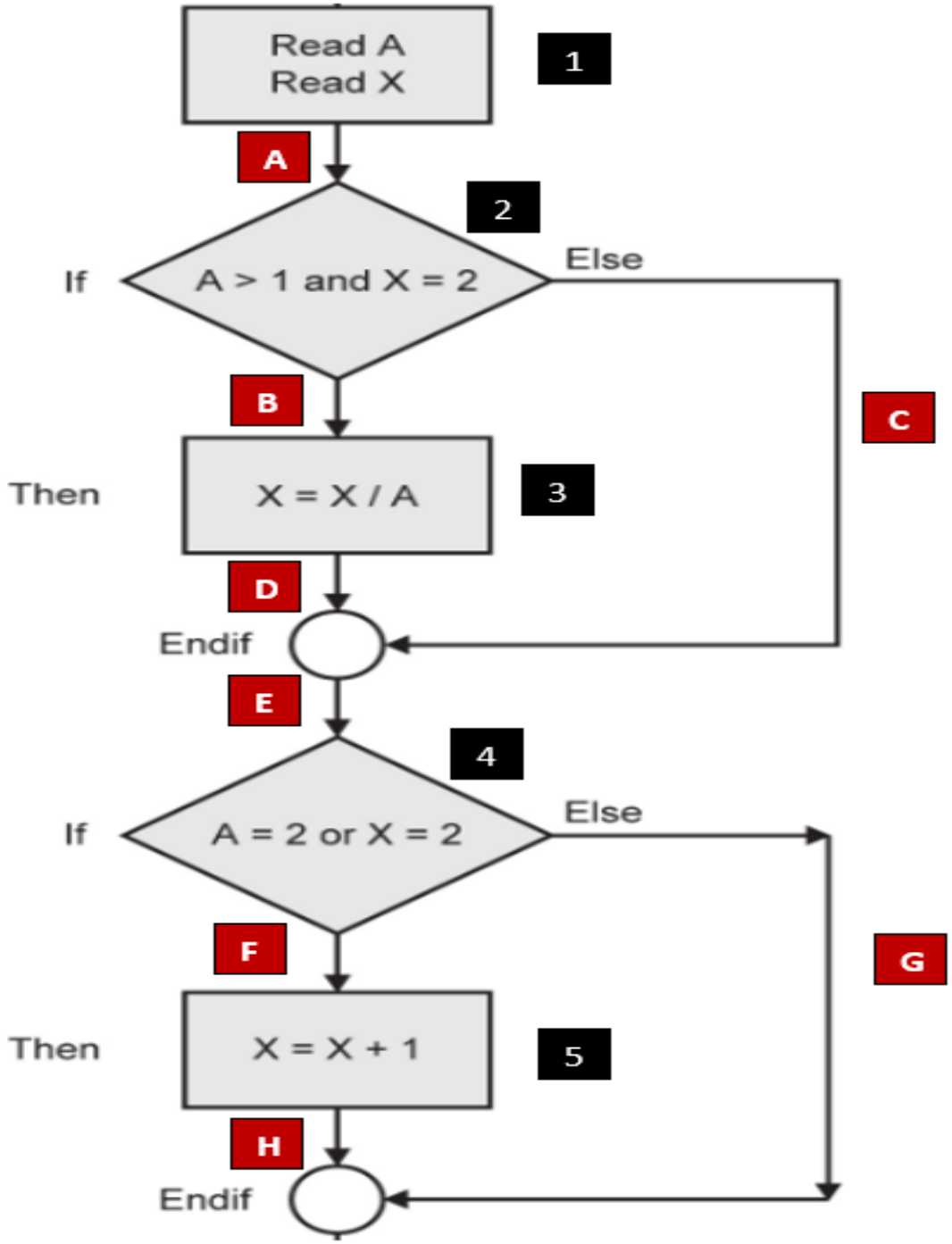
Or #decisions + 1 = 3

Or #regions + 1 = 3

So, we have 3 independent paths to tests

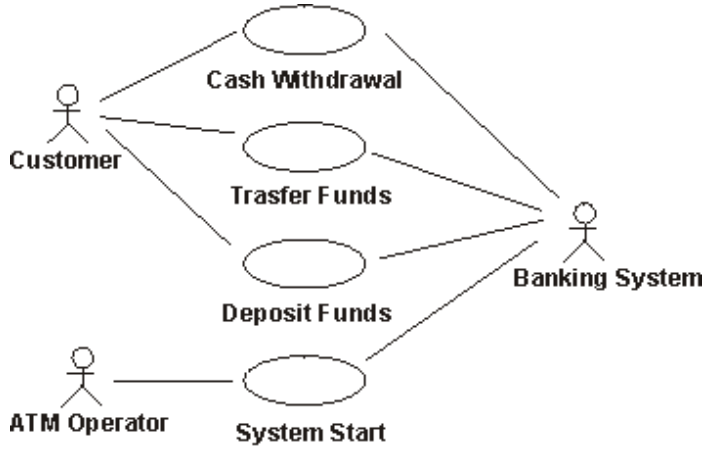
White Box Testing – Example

Statement vs. Branch Coverage?



Test Case Design - Exercise

Exercise - Test Case



1. The client inserts an ATM card. The system reads and validates the card.
2. System prompts for PIN. The client enters the PIN. The system validates the PIN.
3. System proposes a choice of operations. Client selects "Cash withdrawal"
4. System requests amount. Client enters amount.
5. System requests account type. Client selects account type. System validates client's account.
6. System asks whether the client wants a receipt. Client enters choice.
7. System ejects card.
8. System dispenses money.
9. System prints receipt.

a. Derive test cases from the use case scenario “cash withdrawal” (focus on functional requirements)
b. Derive test cases for
• performance tests
• security tests
the use case scenario “cash withdrawal” (focus on non-functional requirements)

Activity - Discussion

Discussion - Testing at Google, Facebook, Amazon, Spotify, Microsoft

What are the testing practices at these 5 companies?

Reference: 5 effective and powerful ways to test like tech giants

Discussion - Testing at Google, Facebook, Amazon, Spotify, Microsoft

- There's a "testing responsibility spectrum," ranging from "**We have dedicated testers that are primarily responsible for executing tests" to "Everybody is responsible for performing testing activities."** You should choose the one that best fits the skillset of your team.

- There is also a "testing importance spectrum," ranging from "**Nothing goes to production untested" to "We put everything in production, and then we test there, if at all."** Where your product and organization belong on this spectrum depends on the risks that will come with failure and how easy it is for you to roll back and fix problems when they emerge.

- **Test automation** has a significant presence in all five companies. The extent to which it is implemented differs, but all five employ **tools** to optimize their testing efforts. You probably should too.

Reference: 5 effective and powerful ways to test like tech giants

Resources – Test Automation

- TDD from Clean Code video series (from the library)
  - TDD Part—Part 1
  - TDD Part—Part 2
  - Advanced TDD Part—Part 1
  - Advanced TDD Part—Part 2

- Test Automation
  - Refactoring: Chapter 4 (test automation)

What's Next?

- **Project Work:**
  - Testing + code review + product version 2 (75%) – due Nov 17
  - Then, another 2 weeks to complete the entire project

- **Reading Assignments**
  - Articles:
    - 5 effective and powerful ways to test like tech giants
    - Code Reviews at Microsoft (on Canvas)
    - Code Reviews at Google (on Canvas)
  - Book Chapters: - Software Engineering at Google: Chapter 9, 11

➔Thursday: Clean code