# CITS5501 Software Testing and Quality Assurance
# Semester 1, 2020
# Week 6 Workshop – Graph-based testing

## Reading

It is strongly suggested you complete the recommended readings for weeks 1-5 *before* attempting this lab/workshop.

## Use cases

*Use cases* are a technique for specifying how users interact with a system. See chapter 7 of the textbook for more on use cases. They are especially useful for devising *system tests*.

The following example of a use case is taken from that chapter, which we will discuss in the workshop:

**Use Case Name:** Withdraw Funds
**Summary:** Customer uses a valid card to withdraw funds from a valid bank account.
**Actor:** ATM Customer
**Precondition:** ATM is displaying the idle welcome message
**Description:**

1. Customer inserts an ATM Card into the ATM Card Reader.
2. If the system can recognize the card, it reads the card number.
3. System prompts the customer for a PIN.
4. Customer enters PIN.
5. System checks the expiration date and whether the card has been stolen or lost.
6. If card is valid, the system checks whether the PIN entered matches the card PIN.
7. If the PINs match, the system finds out what accounts the card can access.
8. System displays customer accounts and prompts the customer to choose a type of transaction. Three types of transactions are Withdraw Funds, Get Balance, and Transfer Funds.

The previous steps are part of all three use cases; the following steps are unique to the Withdraw Funds use case.

9. Customer selects Withdraw Funds, selects account number, and enters the amount.

10. System checks that the account is valid, makes sure that the customer has enough funds in the account, makes sure that the daily limit has not been exceeded, and checks that the ATM has enough funds.
11. If all four checks are successful, the system dispenses the cash.
12. System prints a receipt with a transaction number, the transaction type, the amount withdrawn, and the new account balance.
13. System ejects card.
14. System displays the idle welcome message.

**Alternatives:**

- If the system cannot recognize the card, it is ejected and a welcome message is displayed.
- If the current date is past the card's expiration date, the card is confiscated and a welcome message is displayed.
- If the card has been reported lost or stolen, it is confiscated and a welcome message is displayed.
- If the customer entered PIN does not match the PIN for the card, the system prompts for a new PIN.
- If the customer enters an incorrect PIN three times, the card is confiscated and a welcome message is displayed.
- If the account number entered by the user is invalid, the system displays an error message, ejects the card, and a welcome message is
- displayed.
- If the request for withdrawal exceeds the maximum allowable daily withdrawal amount, the system displays an apology message, ejects the card, and a welcome message is displayed.
- If the request for withdrawal exceeds the amount of funds in the ATM, the system displays an apology message, ejects the card, and a welcome message is displayed.
- If the customer enters Cancel, the system cancels the transaction, ejects the card, and a welcome message is displayed.
- If the request for withdrawal exceeds the amount of funds in the account, the system displays an apology message, cancels the transaction, ejects the card, and a welcome message is displayed.

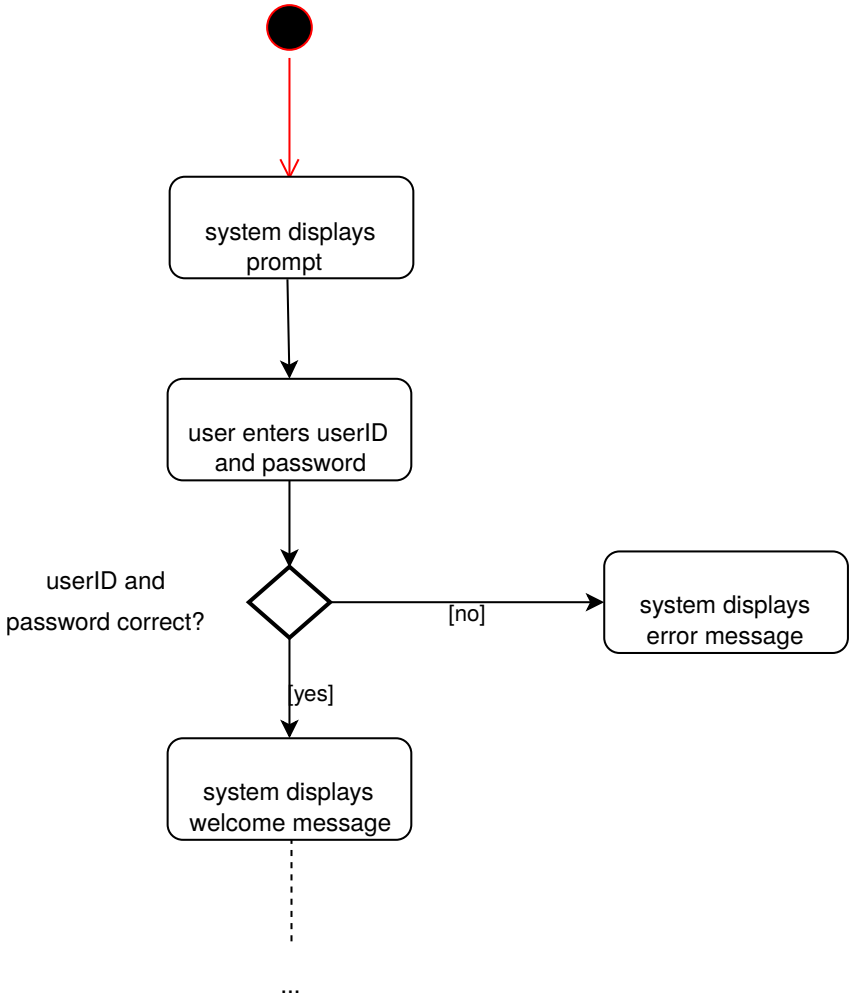
**Postcondition:** Funds have been withdrawn from the customer's account.

## Activity diagrams

*Activity diagrams* are a sort of flowchart use for showing the workflow in a use case. (The textbook contains further information on activity diagrams.)

Typically, ellipses represent *actions* performed by a user or the system; *decision points* are represented as diamonds; a filled-in circle represents the *start* (initial node) of the workflow; a filled-in circle with a circle around it represents the *end* (final node of the workflow); and arrows represent progression from one activity to another. Arrows can be annotated with text in square brackets to represent the conditions which must be true for the workflow to transition along that arrow.

A sample portion of an activity diagram:

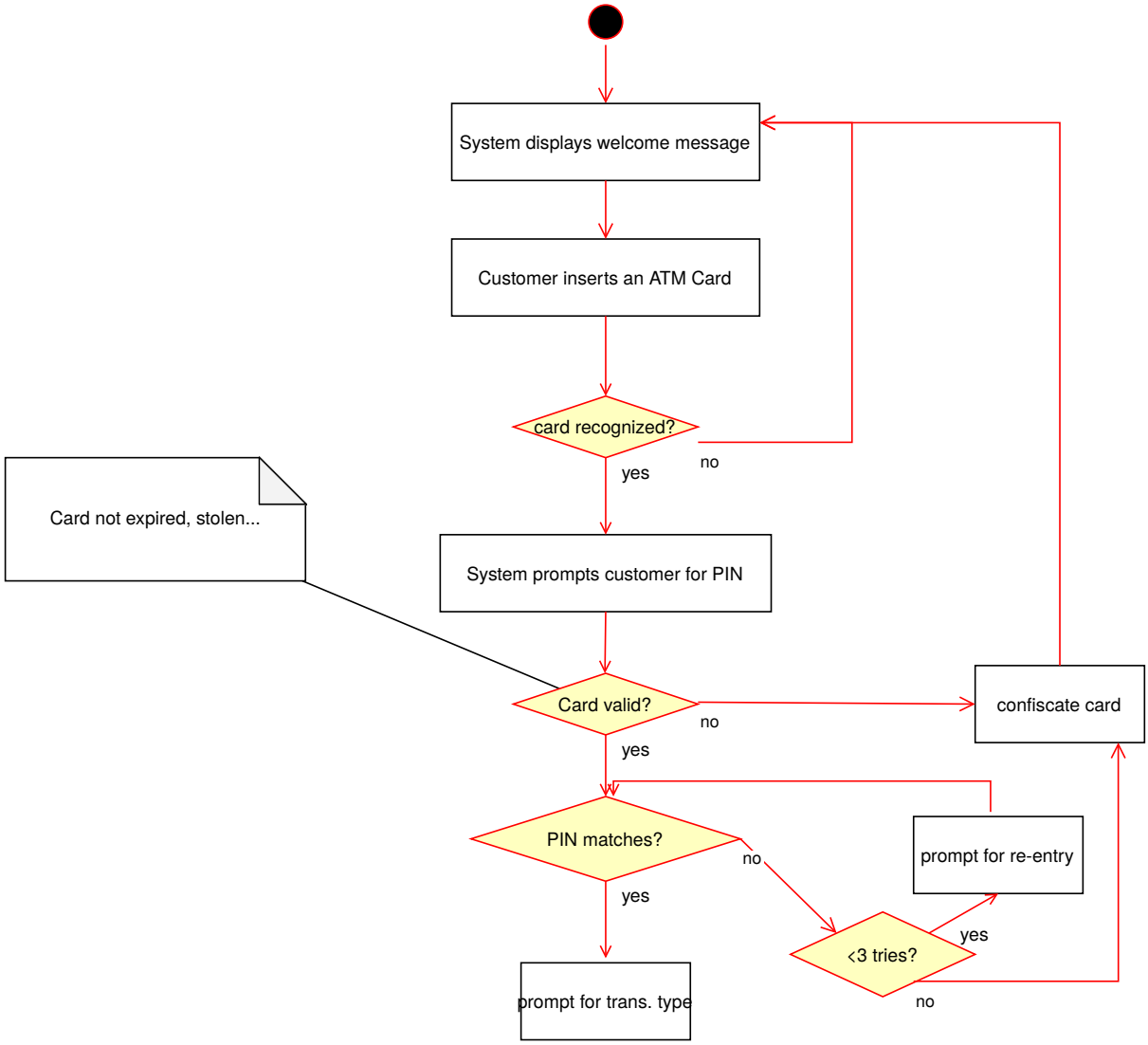



## Use case exercises

1. See if you can construct an activity diagram showing the workflow between activities in the use case for "Withdraw Funds" (steps 1–8 is enough for this exercise).

   Compare your answer with other students – do your solutions differ? Do they both represent the use case accurately?

Sample solution




A use case *scenario* describes a particular trace through use case.

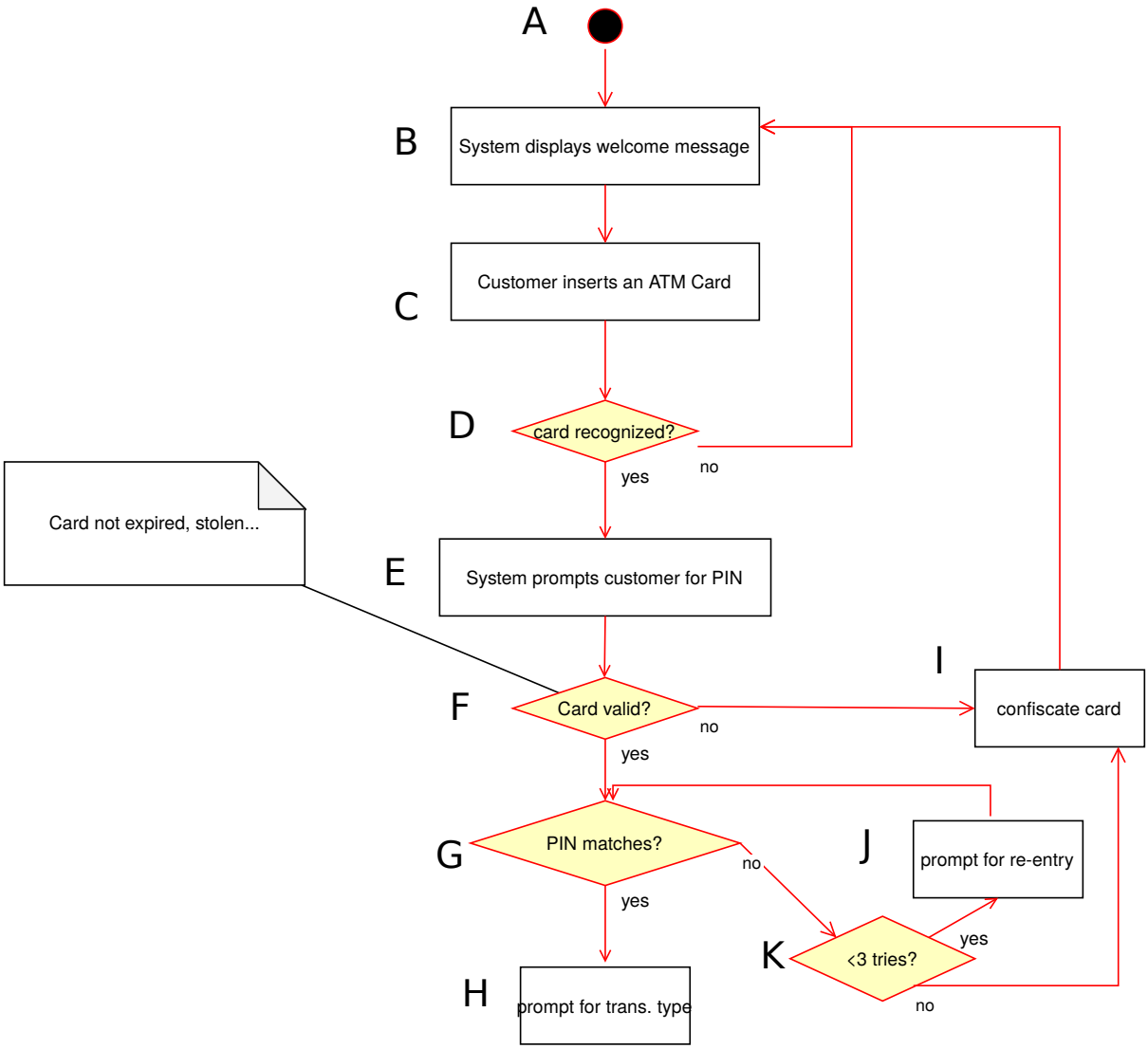
Once we have an activity diagram, a quick way of describing a scenario is just to label all the nodes with an ID (e.g. letters of the alphabet), and list the nodes of the scenario in order (e.g. "A B E" might be a scenario for some activity diagram).

2. For the activity diagram you have constructed, try identifying:

   - a *simple* path (recall that this is a path that doesn't intersect itself, except that the first and last node may be the same)
   - a *prime path* (recall that this can be thought of as a maximal simple path – a simple path that can't be extended without ceasing to be simple; see the lectures for the formal definition)

Sample solution

Suppose we label nodes in columns, top-down – A through for the start node through to "prompt for trans. type", then I, J and K for the nodes to the right.




Then AB, for instance, is an example of a *simple* path. So is FJIF. But not FJIFG - that crosses itself somewhere that isn't the start and the end.

ABCDEFGH is an example of a *prime* path. It can't be made any "longer" from the start or end at all; therefore it can't be a proper sub-path of any other simple path; therefore it's prime.

FJIF is another example of a simple path. We could "make it longer" by extending it from the start or the end – e.g. EFJIF is the same path, but with an extra node at the start – but once we do that, it ceases to be simple. Therefore, FJIF is not a proper sub-path of any simple path; therefore it's prime.

Another prime path is BCDB (start at "Systems displays welcome message", and go in a loop); so is CDBC. Both these paths start at different points round the loop.

3. Try to identify *all* prime paths through the system.

How many do you get? Compare your answer with other students.

**Solution**

A somewhat systematic way is to just start from node A, and work through the graph, shifting the start node as you go, and trying to create every prime path you can. Try "straight" paths first, then loops.

You should come up with:

- ABCDEFGH, ABCDEFGKI, ABCDEFGKJ, ABCDEFI
- BCDB, CDBC, DBCD (first loop)
- BCDEFIB, CDEFIBC, DEFIBCD, EFIBCDE, FIBCDEF, IBCDEFI (second loop)
- GKIBCDEFG (plus 7 more prime path starting at different points through that loop)
- GKJG, KJGK, JGKJ
- JGKIBCDEF
- KIBCDEFGH
- KJGH

(Let me know if you think you've found one that's not here.)

We can use a scenario to devise a system test: given particular user inputs, the system should exhibit particular behaviour, and end up in a particular state. (E.g. We can list the messages it should display, and give an English language description of the state it should end up in.)

4. Give an example system test based on a scenario for the use case that includes a prime path, including a brief note on any setup required. A natural language description is enough, but recall that any test case at a minimum should give the *test values* and the *expected results*.

   Compare your results with another student's – how did their description compare with yours? Do either of you have details the other does not?

Example:

**Test ID:** 0001
**Test description:** Case where card is unrecognized.
(Traces nodes ABCDB.)

**Test setup**:

- Card should be prepared.
- System should be put into start/ready state.
- Mock remote "lost/stolen" system should report card as lost.

**Inputs**: Insertion of card.

**Expected result:** System should display welcome message, confiscate the card, then display the welcome message.

## Self-study – control flow

Here is the body of the search method shown in last week's workshop:

```
1  public static int binarySearch(char[] array, int startIndex, int endIndex,
       ↪ char value) {
2      if (startIndex > endIndex) {
3          throw new IllegalArgumentException();
4      }
5      if (startIndex < 0 || endIndex > array.length) {
6          throw new ArrayIndexOutOfBoundsException();
7      }
8
9      int lo = startIndex;
10     int hi = endIndex - 1;
11     while (lo <= hi) {
12         int mid = (lo + hi) / 2;
13         char midVal = array[mid];
14         if (midVal < value) {
15             lo = mid + 1;
16         } else if (midVal > value) {
17             hi = mid - 1;
18         } else {
19             return mid;  // value found
20         }
21     }
22     return lo * -1;  // value not present
23 }
```

Try drawing a control-flow graph for the method, stating any simplifying assumptions you need to make.

See if you can identify prime paths in which the loop is executed:

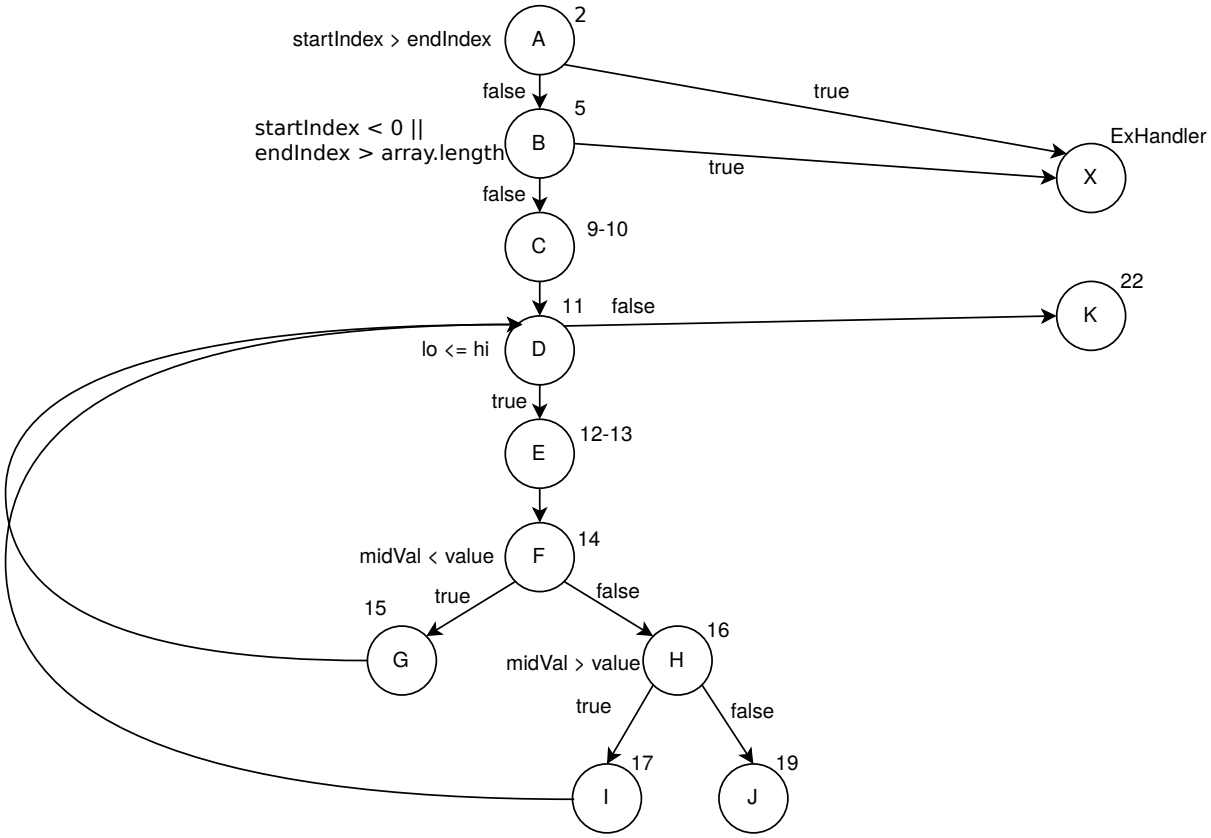
- zero times
- once
- more than once

See if you can construct test cases for these prime paths.

**Sample solution – assumptions and diagram**:

Simplifying assumptions:

- When doing control flow diagrams by hand, we'll usually ignore exceptions (or create just one "exit" node to represent all thrown exceptions)
- We'll often assume any methods called don't throw exceptions
- We'll often assume parameters are not null



**Sample solution – prime paths and test cases**:

***Zero loops:***

Prime paths in which the loop is executed zero times (i.e., no statement in the body of the loop is ever executed): Examples are AX, ABX, and ABCDK.

For a test case that exercises the path AX, we could have:

- Test values: array = `{'a'}`, startIndex = `1`, endIndex = `0`, value = `'a'`.
- Expected result: Throws an `IllegalArgumentException`.

***One loop:***

Prime paths in which the loop is executed one time: we interpret this as meaning "Some statement in the body of the loop (i.e. lines 12-20) is exected at least once, and no statement in the body of the loops is executed twice".

Examples are: ABCDEFG, ABCDEFHI, ABCDEFHJ.

Note that when we come to write a *test* based on these prime paths, it may well be that the loop runs more times than one; but the *prime path* only goes through the loop *once*.

For a test case that exercises the path ABCDEFG, we could have:

- Test values: array = `{'a','b','c'}`, startIndex = `0`, endIndex = `2`, value = `'c'`.
- Expected result: Returns the int `2`.

***More than one loop:***

Prime paths in which the loop is executed more than one time: we interpret this as meaning "At least one statement in the body of the loop (i.e. lines 12-20) is exected twice".

An example is DEFGD.

A possible test case:

Test values: array = `{'a','b','c','d'}`, startIndex = `0`, endIndex = `3`, value = `'d'`.
Expected result: Returns the int `3`.