ECS 98F - Debugging Methodologies

Noah Rose Ledesma

Debugging

Context

- Programmers spend **a lot of time debugging**
  - Almost 50% of time is spent debugging!
- Procedures for debugging are rarely taught
- This lecture presents strategies for problem solving & debugging.

Goals

- Standardize the process of debugging
- Generate problem solving framework

Obstacles

- No established method for teaching debugging & problem solving
- Experience cannot be taught
- Many pitfalls and difficulties to be aware of

Source: University of Cambridge, Judge Business School

Difficulties & Pitfalls

Struggling to understand intended versus actual behavior

- Explaining the difference between them without getting too technical
- Try explaining these two out loud to a friend, family member, or pet rock

Reluctance to split up a task into conceptual chunks

- Makes code lengthy, hard to understand, and hard to debug!
- Instead: recursively break up problems into a set of smaller problems

Failure to contextualize problems before trying to solve them

- Don't jump straight into writing code
- Lack of planning leads to great frustration

Rigidly committing to false hypotheses

- Be critical with the assumptions you make about your program
- Allow yourself to be proven wrong

MINS

Scenario

- ATM software for deposits and withdrawals.
- Automated tests have exposed some errors, how do we fix them?

Four step debugging process

- M: Make a mental model
- I: Identify inconsistencies
- N: Narrow the problem scope
- S: Solve!

Metacognition

- I.e. becoming "aware of one's own awareness"
- Practice shown to improve cognitive abilities
- Goal: be aware of your debugging thought process
- General strategies to invoke **metacognition**
  - Such as self-questioning, thinking aloud, and graphic representations
  - More later

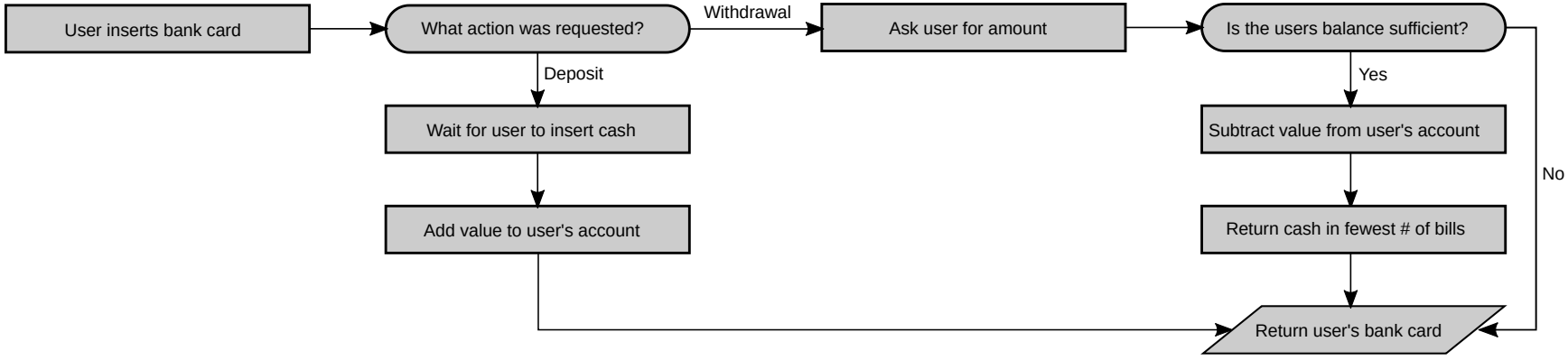
MINS

(M)ental Models

- Understand the system that has the bug
- What is the *intended* behavior of the system?

Strategies

- Explain in "plain English" what the program *should* do
- Plan by breaking down into well-defined chunk
- Create a flow chart or diagram of the program logic



Flowcharts

Purpose

- Great mechanism for planning a program or function
- Forces thinking within logical constraints
- Avoid technical explanations
  - Don't mention variables, control flow statements (if, for, while, etc...)

Building blocks

- Process: Represents an action
- Decision: Represents a question with a finite number of possible answers.
  - Decisions are usually phrased as yes or no questions.
- Terminal: Represents the beginning or end the chart

Flowcharts

Creation

- Start with a clear understanding of **what** the program should accomplish
  - This can usually be described in terms of inputs and output

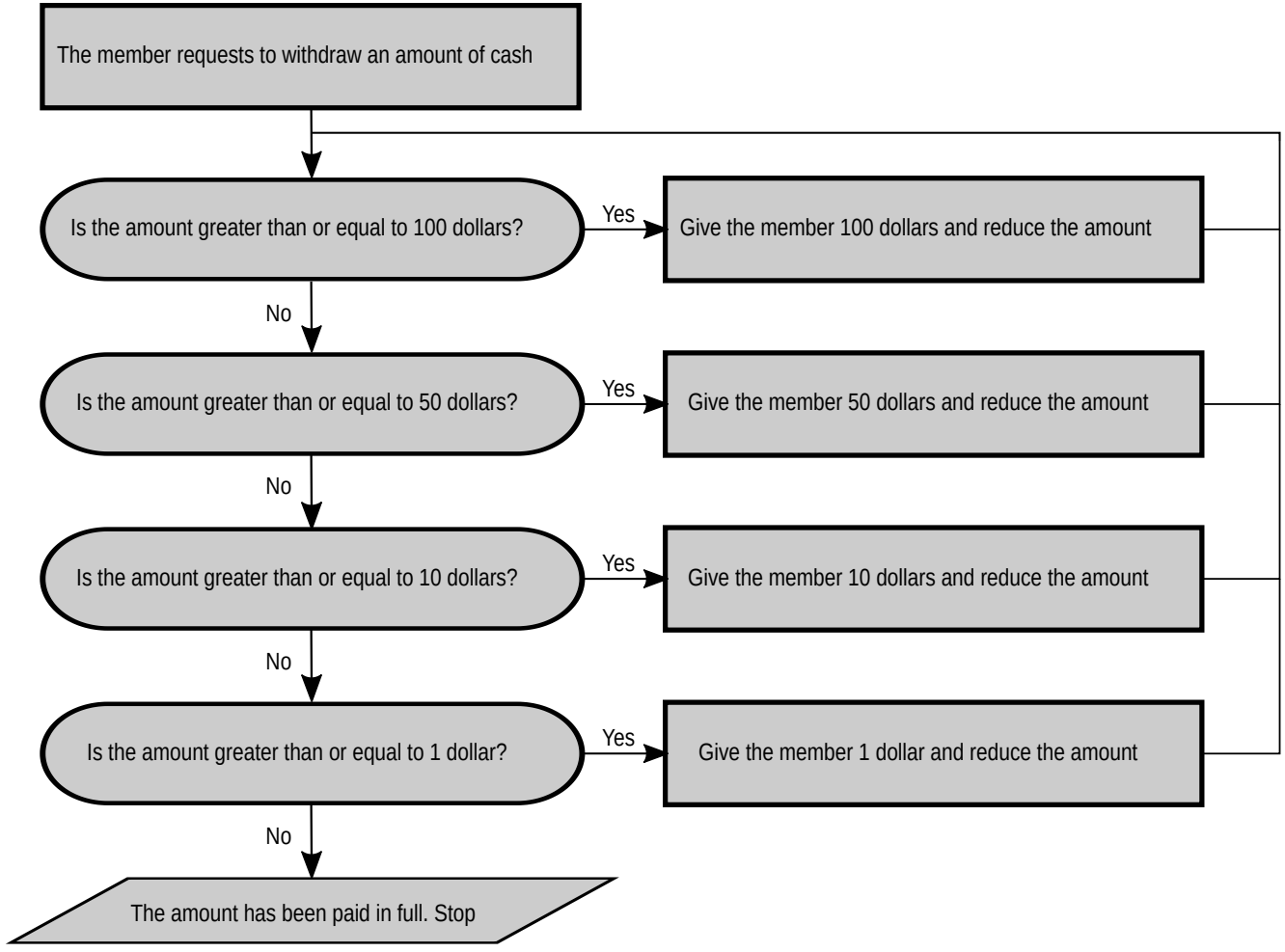
Example

A bank member requests to withdraw money in the fewest number of bills possible. How many of which bills they are to receive? You have bills in the following denominations: 1, 10, 50, and 100.

- Input: Total amount of withdrawal.
- Output: The fewest number of bills that sum to the total amount.

Flowcharts

Solution



MINS

(I)dentify Inconsistencies

- What is the *actual* behavior of the program?
- How does this differ from the *intended* behavior?

Strategies

- Describe the differences between the actual and intended behaviors.
- What other inputs trigger the bug?

Example

```
$ ./broken_change 120
The change is:
1 x $100
0 x $50
1 x $10
10 x $1
```

"The program gave the user ten one dollar bills instead of one ten dollar bill."

MINS

Try more inputs

```
$ ./broken_change 110
The change is:
1 x $100
0 x $50
0 x $10
10 x $1
```

```
$ ./broken_change 30
The change is:
0 x $100
0 x $50
2 x $10
10 x $1
```

```
$ ./broken_change 10
The change is:
0 x $100
0 x $50
0 x $10
10 x $1
```

Ten one dollar bills are always given in place of one of the ten dollar bills.

MINS

(N)arrow down the problem scope

- Interpret inconsistencies to hypothesize the bug's location
- Pose answerable questions that allow you narrow the scope of possible questions.
- Repeat this process until the location of the bug has been pinpointed.

Example hypotheses

- The first time a ten dollar bill was to be given, 10 one dollar bills were given.
- The last time a ten dollar bill was to be given, 10 one dollar bills were given.
- 10 one dollar bills were given at neither the first or last time a ten dollar bill was to be given.

Change Program

```
void MakeChange(int value) {
    unsigned int ones = 0, tens = 0, fifties = 0, hundreds = 0;
    while(value > 0) {
        if(value >= 100) {
            hundreds++;
            value -= 100;
        } else if(value >= 50) {
            fifties++;
            value -= 50;
        } else if(value > 10) {
            tens++;
            value -= 10;
        } else if(value >= 1) {
            ones++;
            value -= 1;
        }
    }
    printf("The change is:\n");
    printf("%i x $100\n", hundreds);
    printf("%i x $50\n", fifties);
    printf("%i x $10\n", tens);
    printf("%i x $1\n", ones);
}
```

broken_change.c

With Logging

```
void MakeChange(int value) {
    unsigned int ones = 0, tens = 0, fifties = 0, hundreds = 0;
    while(value > 0) {
        if(value >= 100) {
            hundreds++;
            value -= 100;
            printf("Gave $100, new value: %i\n", value);
        } else if(value >= 50) {
            fifties++;
            value -= 50;
            printf("Gave $50, new value: %i\n", value);
        } else if(value > 10) {
            tens++;
            value -= 10;
            printf("Gave $10, new value: %i\n", value);
        } else if(value >= 1) {
            ones++;
            value -= 1;
            printf("Gave $1, new value: %i\n", value);
        }
    }
}
```

broken_change_printing.c

Output

```
$ ./broken_change_printing 30
Gave $10, new value: 20
Gave $10, new value: 10
Gave $1, new value: 9
Gave $1, new value: 8
Gave $1, new value: 7
Gave $1, new value: 6
Gave $1, new value: 5
Gave $1, new value: 4
Gave $1, new value: 3
Gave $1, new value: 2
Gave $1, new value: 1
Gave $1, new value: 0
```

The last time a ten dollar bill was to be given, 10 one dollar bills were given.

MINS

(S)olve

- After locating the bug, generate a solution and test it.
- Typically, locating a bug is much harder than solving it.

```
        } else if(value > 10) {
            tens++;
            value -= 10;
```

broken_change.c

vs

```
        } else if(value >= 10) {
            tens++;
            value -= 10;
```

fixed_change.c

Demo

Scenario

- Implementation of Caesar cipher
- Oldest & simplest form of encryption
- Shift every letter in the alphabet by a set amount