Workshop on the Teaching of Software Testing

The Challenge of Teaching Software Testing Earlier into Design Courses

Ellen Francine Barbosa [1]
Richard LeBlanc [2]
Mark Guzdial [2]
José Carlos Maldonado [1]

[1] University of São Paulo (Brazil)
[2] College of Computing, Georgia Institute of Technology (USA)

Agenda

- Introduction
- CS2340 Course
  - Goals and audience
  - Prerequisites
  - Development process
- Testing Approach for CS2340 Course
  - Equivalence Partitioning Criterion
- Students' Evaluation
  - Testing early
  - Using testing criteria
  - Writing test plans
- Conclusions and Further Work

Introduction

- Teaching Software Testing…

Software Testing has traditionally been taught according to the classical approaches to software development.

  - Waterfall Model
    - » Analysis, Design, Coding, **Testing**, Code Release

  - Students' perception about Sofware Testing
    - » Not a popular discipline
      - ✓ They have to do to show that their programming assignments work as required

*Something that happens at the **end** of the development process.*

Introduction

- Teaching Software Testing…
  - Different approach

Introduce testing practices during all phases of the development process, starting from analysis and going through design and implementation phases.

  - eXtreme Programming (XP)
    - » Write tests even before writing code
    - » Development setting
      - ✓ Enhance the system's quality
      - ✓ Reduce development costs

"Test first, code later".

Introduction

- Objective

Explore the impact of introducing testing practices as early as possible, throughout all phases of the development process.

  - Investigate different learning opportunities
  - Provide extra motivation
    - Ideas for modern methodologies (like XP)
  - CS2340 Course
    - Encourage the students on thinking about Software Testing earlier in their projects
      - ✓ Provide a more pragmatic view of testing
    - Evaluate the students' attitude

CS2340 – *Objects and Design*

- Main Goal

> *Explore higher-level issues of analysis and design, with some emphasis on user interface development.*

  - Object-oriented paradigm
- Audience
  - Second-year undergraduate students
    - Fall 2002: 160 students
- Structure
  - One semester
    - 2 lecture classes per week
    - 2 exams, 1 development project

CS2340 – *Objects and Design*

- Pre-requisites
  - *Introduction to Computing*
    - » Design, construction and analysis of algorithms
  - *Object-Oriented Programming*
    - » Java
  - *Languages and Translation*
    - » Issues of language implementation, tokenizing and parsing
  - *Software Practicum* (sometimes concurrently)
    - » Basics of Software Engineering
      - ✓ General view on Software Testing

CS2340 – *Objects and Design*

- Projects
  - Team-oriented
  - Development of a small-to-medium sized OO system
  - Fall 2002: Genealogy Information
    - Create a system to...
      - ✓ Collect genealogical information
      - ✓ Note inconsistencies and flaws in the database
      - ✓ Provide some graphical representations
      - ✓ Support a standard genealogical information format

CS2340 – *Objects and Design*

- Development Process
  - Traditional phases of the object-oriented development
    - OO Analysis
      - Development of scenarios and CRC (Class-Responsibility-Collaborator) cards
    - OO Design
      - Development of UML class diagrams
    - OO Programming
      - Use of Squeak language (variant of Smalltalk)
        - Open source and highly portable language
        - Multimedia support
        - Good infrastructure for complex projects
      - Squeak: supports the XP's testing-driven practices
        - S-Unit testing framework

CS2340 – *Objects and Design*

- Development Process
  - Regarding Software Testing...

No systematic method to test the systems had been adopted in the previous terms of the CS2340 course.

  - » Fall 2002: first time that testing-related activities were explored in the context of the CS2340 classes
    - ✓ Decide what testing criteria to use
      - – Develop lecture materials
    - ✓ Figure out how to integrate testing into student assignments
      - – How they should be evaluated and graded
    - ✓ Put together an assessment to see how well it worked

Testing Approach for CS2340 Course

- Equivalence Partitioning Criterion
  - Starting point to fit testing practices into the scope of the course
    - » Key criterion for functional testing
    - » Seems to be easily understood even by students with little experience on testing
  - Educational setting
    - » Small programs
      - ✓ Numeric input values
    - » Genealogy system
      - ✓ Input domain involves more complex types of elements
        - – How to apply the criterion?

Testing Approach for CS2340 Course

- Equivalence Partitioning Criterion
  - Set of “directions”
    - » Help the students on using the criterion in their projects
  - OO Analysis

    CRC cards and scenarios + **Definition of input conditions**
    Identification of equivalence classes
  - OO Design

    UML class diagrams + **Definition of test cases**
    Development of test plans
  - OO Implementation

    Squeak + **Execution of test cases (S-Unit framework)**

ATM Simulation System

- Specification (simplified)
  - The customer is required to enter the account number and the PIN.
    - There is no need of an ATM card.
  - The ATM must provide the following transactions to the customer:
    - Cash Withdrawals, Deposits, Tranfers and Balance Inquiries.
    - Only one transaction is allowed in each session.
  - The ATM communicates the transaction to the bank and obtains verification that it was allowed by the bank.
    - If the bank determines that account number or PIN is invalid, the transaction is canceled.
  - The ATM has an operator panel that allows an operator to start and stop the servicing of customers.
    - When the machine is shut down, the operator may remove deposit envelopes and reload the machine with cash.
    - The operator is required to enter the total cash on hand before starting the system from this panel.

ATM Simulation System

- Classes...
  - ATM
  - CashDispenser
  - EnvelopeAcceptor
  - OperatorPanel
  - Session
  - Transaction
  - WithdrawalTransaction
  - DepositTransaction
  - TransferTransaction
  - InquiryTransaction
  - Bank
- Some Scenarios...
  - System Startup
  - Session
  - Cash Withdrawal Transaction
  - Deposit Transaction
  - Transfer Transaction
  - Balance Inquiry
- CRC cards...

Defining Input Conditions

Although the CRC cards do not explicitly deal with the input data items, analyzing their responsibilities under a macroscopic perspective can provide some insight into that.

Input conditions are closely related with ...put data provided by the user.

1. Analyze each CRC card, looking f... ...ggestions" on the input data items provided by the user.

(a) Consider the responsibilities in a macroscopic way. Focus on *what* each class really has to do.

(b) Analyze the responsibilities of the collaborators too.

(c) Write down all the input data items ...ed to a specific class.

Since each class can interact with others, the collaborators should also be investigated in order to find the right class where a specific input item is being treated.

Defining Input Conditions

Class ATM

Responsibility	Collaborator
Start up system operation (initial cash)	OperatorPanel
Start a session for each customer	Session
Shut down on operator request	OperatorPanel
Get PIN from the customer	
Get transaction choice from the customer	
Get account from the customer	
Get amount entry from the customer	(Class xxxTransaction)
Verify the availability of cash for withdrawal	CashDispenser
Dispense cash	CashDispenser
Accept deposit envelope	EnvelopeAcceptor

Defining Input Conditions

Class CashDispenser

Responsibility	Collaborator
Set initial cash on hand at startup	Operator Panel
Report cash available	
Dispense cash	

Class EnvelopeAcceptor

Responsibility	Collaborator
Accept deposit envelope	

Class OperatorPanel

Responsibility	Collaborator
Get initial cash on hand from operator	

Defining Input Conditions

Class Session

Responsibility	Collaborator
Perform session use case	ATM, Transaction
Perform invalid data	ATM, Bank
Furnish account to Transaction	
Furnish PIN to Transaction	

Class Transaction

Responsibility	Collaborator
Perform a particular transaction use case	WithdrawalTransaction, DepositTransaction, TransferTransaction, InquiryTransaction, Session, Bank

Defining Input Conditions

Class WithdrawalTransaction

Responsibility	Collaborator
Get specifics from customer	ATM
Send to bank	Session, Bank
Dispense cash and notify bank when complete	ATM, Bank

Class DepositTransaction

Responsibility	Collaborator
Get specifics from customer	ATM
Send to bank	Session, Bank
Accept envelope and notify bank when complete	ATM, Bank

Class TransferTransaction

Responsibility	Collaborator
Get specifics from customer	ATM
Send to bank	Session, Bank

Defining Input Conditions

Class InquiryTransaction

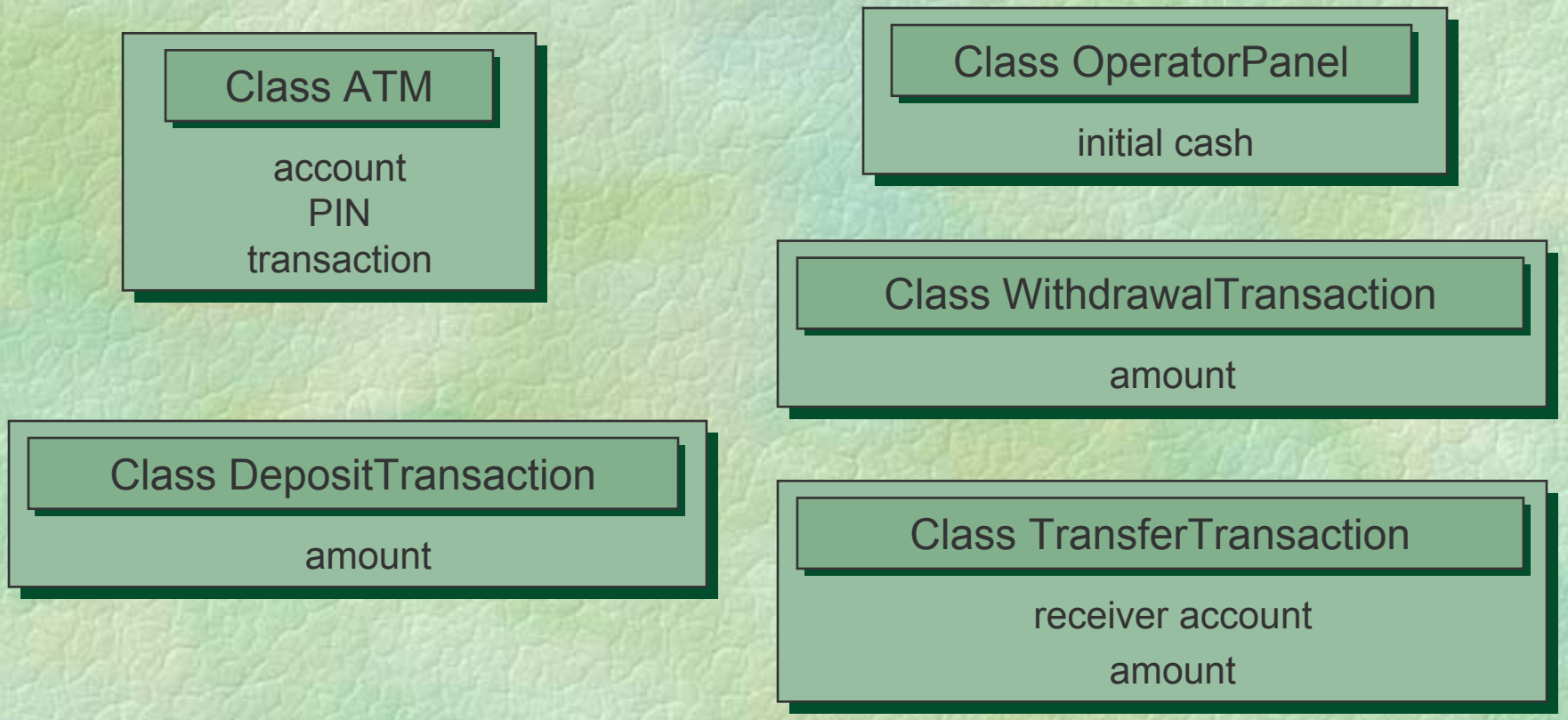
Responsibility	Collaborator
Get specifics from customer	ATM
Send to bank	Session, Bank

Class Bank

Responsibility	Collaborator
Initiate withdrawal	
Finish withdrawal	
Initiate deposit	
Finish deposit	
Do transfer	
Do inquiry	

Defining Input Conditions

- Input Data Items



Defining Input Conditions

Each in
characteristic
affect

Input data items can interact with each other, also resulting on different values in the output domain.

2. Identify the main charac[illegible]tics of each input data item.

3. Identify the interactions among the input data items.
Think about how a specific input entry can be related to the other input data of the system.
Focus on the system's operation (scenarios can help on this).

4. Write down the characteristics and interactions of all input data items related to all classes of the system.

The set of characteristics and interactions related to all input data items can be seen as the ***input conditions*** *for the system.*

Defining Input Conditions

Class ATM

Input Data Item	Characteristic / Interaction
account	matching with bank
PIN	matching with account
transaction	type

Class OperatorPanel

Input Data Item	Characteristic / Interaction
initial cash	valid characters
	availability of money in cash dispenser

Defining Input Conditions

Class WithdrawalTransaction

Input Data Item	Characteristic / Interaction
amount	valid characters
	availability of money in the account for withdrawal

Class DepositTransaction

Input Data Item	Characteristic / Interaction
amount	valid characters

Class TransferTransaction

Input Data Item	Characteristic / Interaction
receiver account	matching with bank
amount	valid values
	availability of money in the sender account for transfer

Identifying Equivalence Classes

If two or more conditions … each other, it can make m… into a single …

The valid and invalid equivalence classes are established by analyzing the input conditions, in terms of the correct and incorrect values needed to cover them.

1. Analyze the set of input cond…

(a) Look for related conditions. Try … them.

(b) Look for broad conditions. Tr… refine them.

2. Analyze each input condition separately.

(a) Think about the valid values to satisfy the input condition. They will correspond to the elements of a valid equivalence class.

(b) Think about other possible values associated to the input condition, i.e., the invalid values. They will correspond to the elements of a invalid equivalence class.

3. Enumerate the equivalence classes, assigning a unique number to each class.

Identifying Equivalence Classes

Class ATM

Input Condition	Valid	Invalid
account matches with bank		
size (a) of account	a = 8 (1)	a > 8 (2) 0 ≤ a < 8 (3)
valid characters for account	numerical (4)	alpha, special (5)
PIN matches with account		
size (p) of PIN	p = 4 (6)	p > 4 (7) 0 ≤ p < 4 (8)
valid characters for PIN	numerical (9)	alpha, special (10)
type of transaction	withdrawal (11) deposit (12) transfer (13) inquiry (14)	none (“blank choice”) (15)

Identifying Equivalence Classes

Class WithdrawalTransaction

Input Condition	Valid	Invalid
valid characters for amount	numerical (1)	alpha, special (2)
availability of money in the account for withdrawal	balance - amount ≥ 0 (3)	balance - amount < 0 (4)

Defining Test Cases

The definition of test cases to cover the equivalence classes requires a more detailed knowledge on the classes (and objects) in terms of structure, attributes, and services that need to be provided.

1. Derive test cases associa
them. A test case can cov
classes, as large a set as

2. Derive test cases associa
of them. A test case shou
equivalence class.

3. Write a test plan providing information on the test cases.

Students have to provide information on:
- purpose of the test case
- system's class it tests
- equivalence class it covers
- expected result, obtained result.

Defining Test Cases

- Test plan

Class	Equivalence Class #	Test Case	Expected Result	Actual Result
Withdrawal Transaction	(3)	Balance: US$ 100,00 Amount: US$ 90,00	Transaction OK Balance: US$ 10,00	Transaction Canceled

Error: Debugging

Actual Result: fill in after the test case execution

Executing Test Cases

- S-Unit
  - Unit testing framework
    - Squeak's environment
  - Structure, describe the context of test cases and run them automatically
  - Development of a well-documented test suite
  - Mechanism to motivate regression tests

Testing Approach for CS2340 Course

- None specific structural testing criterion was adopted
  - » Instead… some “general” guidelines…
    - ✓ Test the base class
    - ✓ Test each subclass in conjunction with the base class

Students were encouraged to keep defining test cases while coding the system, based on its implementation details.

- OO Implementation
  - » Design structural test cases
    - ✓ Test plans
  - » Execute the structural test cases against the system
    - ✓ S-Unit framework

Testing Approach for CS2340 Course

- Relevant Points
  - CRC cards can be helpful to define input conditions and…
  - Input conditions can be useful in checking the consistency of CRC cards
    - » Responsibilities and interactions among classes
  - Defining input conditions and equivalence classes during the OOA phase can provide a better understanding of the system's requirements
  - Writing test cases even before writing code (OOD phase)
    - » Encourage the students to think about the functionality they are designing

Students' Evaluation

- Survey
  - Applied at the end of the course
    - After students had finished their projects
  - Voluntary
    - 105 students
  - Questions consisting in some statements
    - Students were required to choose all the statements they agreed with
  - Evaluate the student's attitude toward…
    - XP idea of starting testing early
    - Using testing criteria
    - Writing test plans

Students' Evaluation

- XP testing ideas

Statement	
I prefer to deal with testing a at the end of the develo	
I prefer to apply testi phases of the develo	53
Start thinking about testing in the analysis phase is helpful to better understand the system's functionality.	41
I feel it helpful to design a set of test cases at least before implementing the system.	23

*Further attempts should be made to integrate the idea of "**testing early**" into CS courses as a way improve students' attitude toward testing and their ability to do it effectively.*

Attitude!!!

Students' Evaluation

- Using testing criteria

70 students agreed on the relevance of applying testing criteria for some aspect of the development process.

Statement	
I prefer to test the system by my "own way", without applying a specific testing criterion.	46
I feel more confident about the **test cases** when working with some testing criterion.	21
I feel more confident about the **system's quality** when working with some testing criterion.	36
I feel motivated to think about **common errors** that could be committed during the development when working with some testing criterion.	29
I feel that working with some testing criterion I can reduce the total **development effort**.	10

Students' Evaluation

- Using testing criteria
  - Moreover…
    - » Negative attitude on using Equivalence Partitioning

Teaching the criterion for testing systems whose input values are not numerical requires more investigation.

- ✓ Keep refining/improving our directions for applying the criterion in the next offerings of the course
- ✓ Investigate the application of other testing criteria

Students' Evaluation

- Writing testing plans
  - Students' opinion on having to do a test plan

Attitude	Students
Completely worthless	62
Neutral position	31
Valuable / really valua	12

Negative Attitude!!!

Students' Evaluation

37 students (from the 62 who had a r
15 students (from the 31 who presen
02 students (from the 12 who had a p

Some of these students were the same who had a negative attitude toward test plans.

Although this specific group of students had been "against" the test plans idea, they were able to recognize its usefulness for some aspects of the development process.

Statement	
I believe it is more useful to elaborate complex systems than for small/medium	
I believe it is valuable to elaborate test plans as part of any development process.	23
Test plans are valuable as a way to keep track of test cases.	13
Test plans are valuable when changes in the system are required.	16

Conclusions

- Positive attitudes towards…
  - Starting testing activities as early as possible
  - S-Unit

> *XP's testing-driven practices can be seen as an interesting and challenging mechanism to create new Software Testing learning opportunities.*

- Negative attitudes towards…
  - Equivalence Partitioning
  - Test plans

Further Work

- CS2340 Course
  - Keep investigating ways to introduce testing practices into the context of the course
    - Introduce the use of S-Unit earlier in the development process
  - Keep applying the Equivalence Partitioning
    - Investigate better ways for teaching the criterion
    - Refine/improve our directions
  - Explore the use of other testing criteria
    - Functional and structural
- Explore the idea of "*testing early*" into the context of other courses
  - Software Testing
  - Software Engineering