Introduction to Software Engineering

by
Josh Hartman

Topics in Software Engineering

Software Requirements - What exactly does the software have to do?

Software Design - How to design the structure and layout of software

Software Development - How will we implement the design

Software Testing - Do we know our code is correct?

Software Maintenance
Software Quality

Software Requirements

* Business Requirements - What must be done
* Product Requirements - Describe a produce which is one way to solve the business requirements
* Process Requirements - Things limiting the product requirement, like what processor to use or the budget

Product Requirements

- Functional requirements describe the system capabilities and functions the system performs
- Non-functional requirements constrain the system in terms of performance, reliability, etc.

Good requirements should be
- Cohesive (only address 1 thing)
- Correct (actually meet the business need)
- Observable (a requirement should be visible to a user, it should not specify software architecture)
- Feasible
- Mandatory
- Verifyable
- Unambiguous

Requirement Analysis

* Stakeholder identification
* Prototyping
* Use cases - A sequence of simple steps between an actor and the system. Actors could be end users or other systems. They are describe from the point of view of the actor.

Bank Transaction Use Case

Started within a session when the customer chooses a transaction type from a menu of options. The customer will be asked to furnish appropriate details (e.g. account(s) involved, amount). The transaction will then be sent to the bank, along with information from the customer's card and the PIN the customer entered.

If the bank approves the transaction, any steps needed to complete the transaction (e.g. dispensing cash or accepting an envelope) will be performed, and then a receipt will be printed. Then the customer will be asked whether he/she wishes to do another transaction.

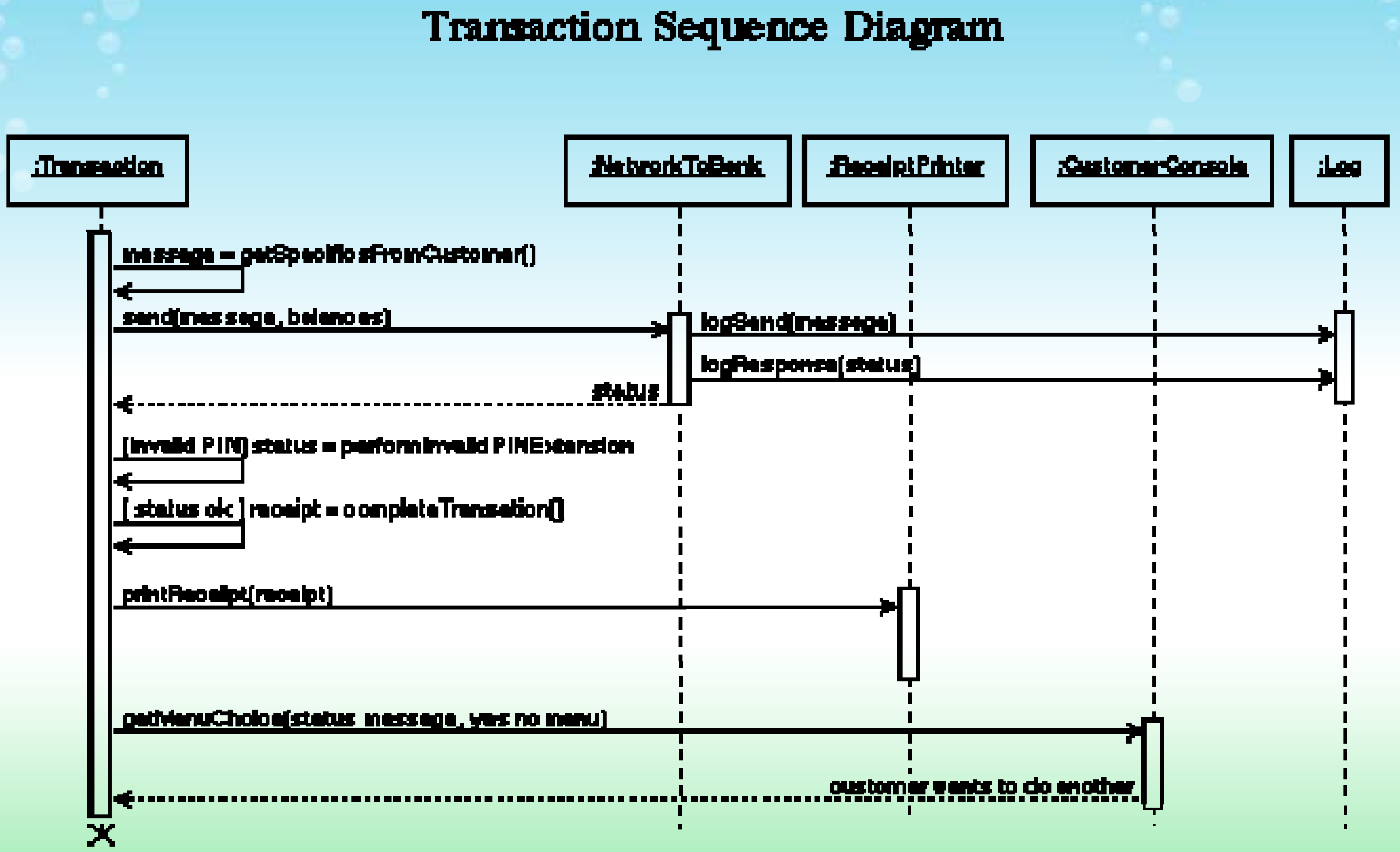
If the bank reports that the customer's PIN is invalid, the Invalid PIN extension will be performed and then an attempt will be made to continue the transaction. If the customer's card is retained due to too many invalid PINs, the transaction will be aborted, and the customer will not be offered the option of doing another.

Bank Transaction Use case cont.

If a transaction is cancelled by the customer, or fails for any reason other than repeated entries of an invalid PIN, a screen will be displayed informing the customer of the reason for the failure of the transaction, and then the customer will be offered the opportunity to do another.
The customer may cancel a transaction by pressing the Cancel key as described for each individual type of transaction below.

All messages to the bank and responses back are recorded in the ATM's log.

Sequence Diagram


Transaction Sequence Diagram
:Transaction
:NetworkToBank
:ReceiptPrinter
:CustomerConsole
:Log
message = getSpecificsFromCustomer()
send(message, balances)
logSend(message)
logResponse(status)
status
[invalid PIN] status = performInvalidPINExtension
[status ok] receipt = completeTransaction()
printReceipt(receipt)
getMenuChoice(status, message, yes no menu)
customer wants to do another

Software Design

- Reliability - The software is able to perform a required function under stated conditions for a specified period of time.
- Robustness - The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with a resilience to low memory conditions.
- Extensibility - New capabilities can be added to the software without major changes to the underlying architecture.

Software Design cont.

- Modularity - the resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.

The Alternate path

When designing use cases, you must always include alternate paths. These are things that may not happen often, but could disrupt your system.

Tips for Software Modularity in C

* Break software into "modules", like ADC routines, LCD routines, motor routines, and so on.

* Use functions AS MUCH AS POSSIBLE. This allows for much easier testing and more readable code. Functions (including main) should be <15 lines most of the time.

* Data should correspond to actors in use cases
* Functions should be the actions taken

Software Portability cont.

* Software portability is also a consideration. What if you want to change uC families or migrate from PIC -> Atmel?

* Use #defines to rename specific ports and registers

```
# define ADC_PORT PORTC
```

* Can use #ifdef, #if, etc. to do conditional compilation

```
#define DEBUG 1
// some code here...
#ifdef DEBUG
LCD_STRING("message");
#endif
```

UML / object oriented programming

* Object oriented programming is all the rave
* Use classes (available in C++, Java, etc)
* Classes group data with behavior and hence are great ways to store state in a program
* Classes can be extended for customization
* Classes can be composed of other classes for more advanced behavior
    - Come up with things like an extendable list of strings, etc.

Software Development Process

* There are several common models for software development
* Each have several benefits and drawbacks

Iterative Process

Iterative development[2] involves the building software from initially small but ever larger portions of a software project to help all those involved to uncover important issues or faulty assumptions.

Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what they want.

Several requirements are planned for completion during each iteration.

Agile Development

Similar to an iterative process. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software.

Iterative/agile processes are great for senior design. You can start writing testing code and at each iteration, integrate parts together to create new modules with more functionality.
Also, most of you are not sure what your final product will do or how it will be done when you start coding!

Extreme Programming

* Coders work on a product subsystem
* First automated tests of software functionality are written
* Coding is then done with a very small team. It is finished when all tests are passed and no more useful tests can be thought of.
* Design and architecture are created after refactoring becomes necessary.

Waterfall model

The waterfall model is the traditional model following these step:

1. Requirements specification
2. Design
3. Implementation
4. Integration
5. Testing and debugging
6. Installation
7. Maintenance

In this model, much of the work is spent in requirements and design, ensuring everyone is on the same page. However, it can be a rigid model in the case of changing requirements or new influences.

Software Testing

NIST reports software bugs cost the economy $59.5 billion annually

Two of the most common sources for bugs in software are:

Lack of compatability: Software is only fully compatible with 1 web brower, OS, or other software package.(Try running a DOS program on Windows Vista)

Uncommon inputs: Programs that do not carefully check for unusual input are vulnerable to bugs. Exception handling can help a lot with this (not available in C)

Types of testing

Static testing: Consists of code reviews, walkthroughs, or inspections. May include the use of code analysis tools.

Dynamic Testing: Actually running the code against test suites or test cases

Verification: Does the software match the specification?
Validation: Is the software actually what the user wants?

Quality Assurance: Implementing practices to reduce bugs in previous stages (like pair programming and scheduled code reviews)

Black-box / White boxTesting

Black Box: Essentially the tester runs code attempting to crash it. He has no access to the internal workings. This has the advantage of finding unexpected bugs, but the disadvantage of blind exploring

White box: Tester has access to program internals.
- Code coverage: What percentage of the code is actually executed (not all code is executed or tested due to conditional logic!)
- Function coverage

Regression testing

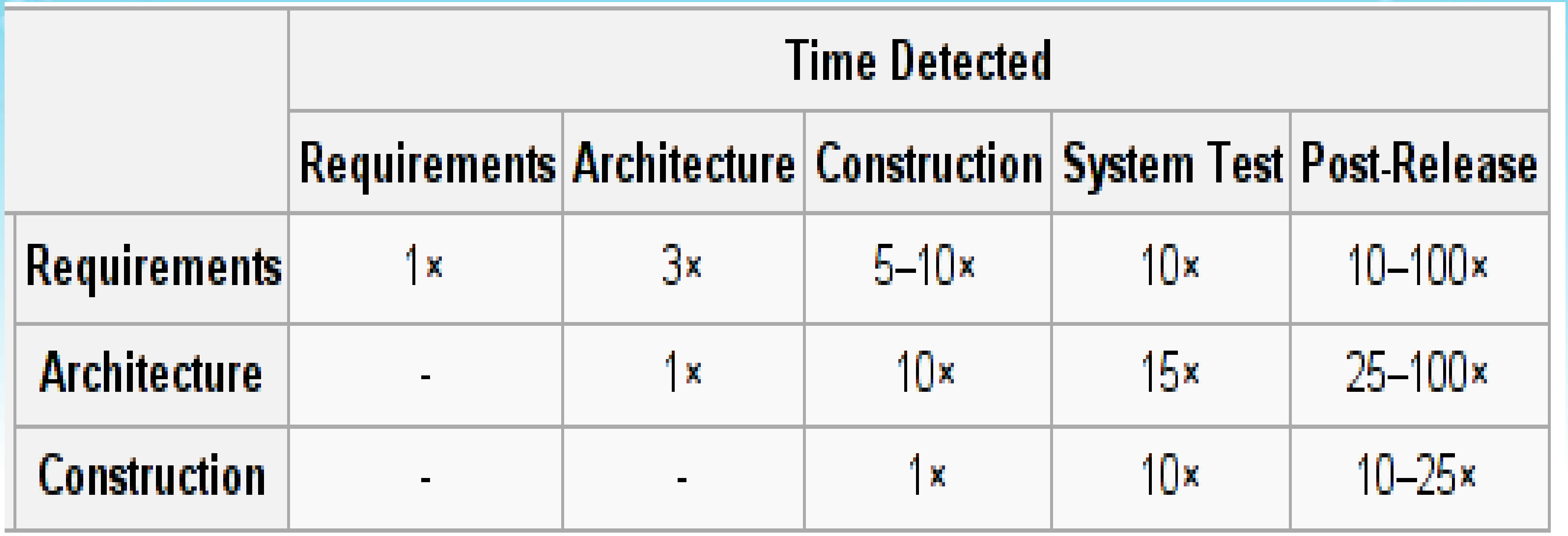
Unit testing: Test minimal software components. Would be like testing an LCD library.

Integration tests: Tests if two modules can work together
System testing: Testing the whole system
System Integration testing: Testing if the system works properly with external software

Testing tools

Software Debugger
Code coverage / analysis tools
Performance toolkits

Cost of bugs



	Time Detected				
	Requirements	Architecture	Construction	System Test	Post-Release
Requirements	1×	3×	5–10×	10×	10–100×
Architecture	-	1×	10×	15×	25–100×
Construction	-	-	1×	10×	10–25×