Software Quality Assurance and Testing

System Testing

Outline

- System Testing
- Functional System Testing
- Risk-Based System Testing
- Non-Functional System Testing
- Acceptance Testing
- Summary

System Testing

System Testing

- Key characteristics:
  - Comprehensive (the whole system, the whole spec)
  - Based on the specification of observable behavior
    - Verification against a requirements specification, not validation, and not opinions
  - Independent of design and implementation
    - Avoid repeating software design errors in system test design

What is System Testing?

	System	Acceptance	Regression
Test for ...	Correctness, completion	Usefulness, satisfaction	Accidental changes
Test by ...	Development test group	Test group with users	Development test group
	Verification	*Validation*	Verification

Independent V&V

- One strategy for maximizing independence:
  - System (and acceptance) test performed by a different organization
- Organizationally isolated from developers
  - no pressure to say "ok"
- Sometimes outsourced to another company or agency
  - Especially for critical systems
  - Outsourcing for independent judgment, not to save money
  - May be additional system test, not replacing internal V&V
- Not all outsourced testing is IV&V
  - Not independent if controlled by development organization

Achieving Independence Without Changing Staff

- If the development organization controls system testing …
  - Perfect independence may be unattainable, but we can reduce undue influence
- Develop system test cases early
  - As part of requirements specification, before major design decisions have been made
    - Agile "test first"
    - Conventional "V model"
    - Critical system testing early in project

Incremental System Testing

- System tests are often used to measure progress
  - System test suite covers all features and scenarios of use
  - As project progresses, the system passes more and more system tests
- Assumes a “threaded” incremental build plan:
  - Features exposed at top level as they are developed

System Testing

- Functional Testing
  - Validates functional requirements
- Performance Testing
  - Validates non-functional requirements
- Acceptance Testing
  - Validates client’s expectations
- Installation Testing

Impact of requirements on system testing:

- The more explicit the requirements, the easier they are to test
- Quality of use cases determines the ease of functional testing
- Quality of nonfunctional requirements and constraints determines the ease of performance tests

Functional System Testing

Functional Testing

- Functional testing finds differences between functional requirements and the implemented system
- Essentially the same as black box testing
- Goal: Test functionality of system
- Test cases are designed from the requirements analysis document (better: user manual) and centered around requirements and key functions (use cases)
- Select tests that are relevant to the user and have a high probability of uncovering a failure
    - Use techniques like equivalence tests

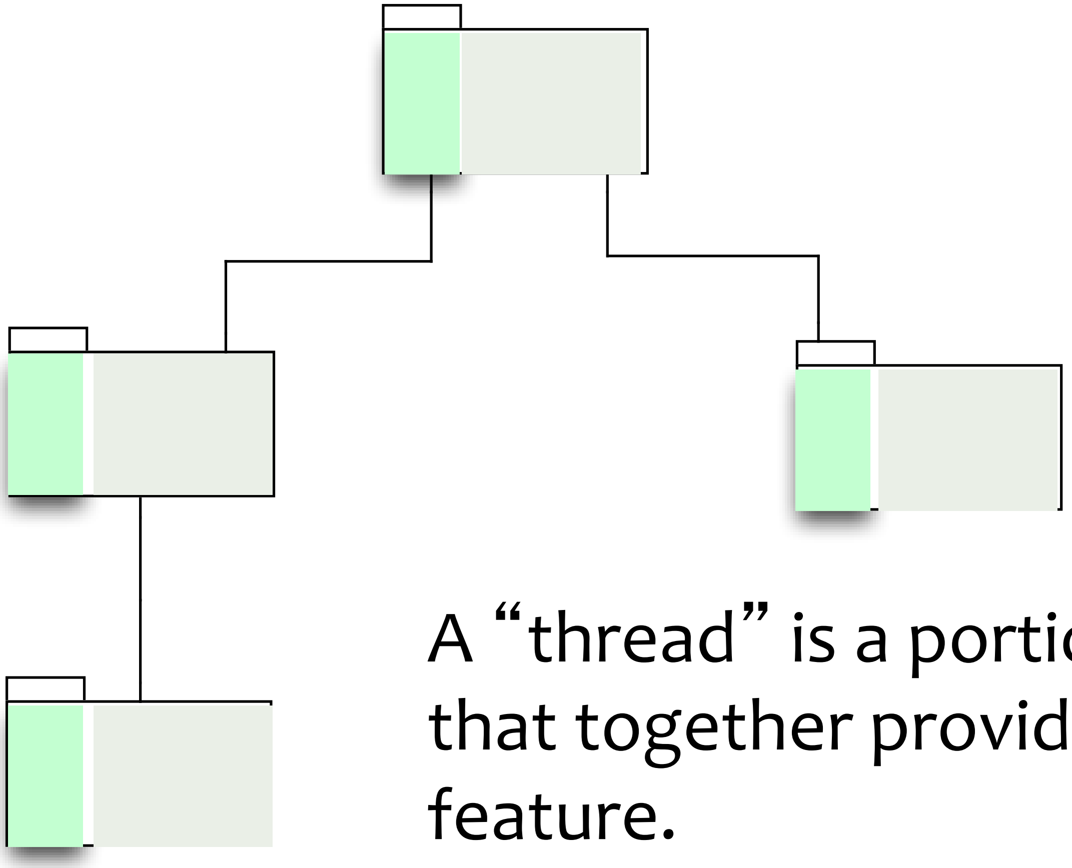
System Testing

- “Intuitively clear”
  - customer expectations
  - close to customer acceptance testing
- BUT we need a better basis for really understanding system testing
- Threads—the subject of system testing
- How are they identified?
  - ad hoc?
  - from experience?
  - from a possibly incomplete requirements specification?
  - from an executable model? (Model-Based Testing)

Threads...

- An execution time concept
- Per the definition, a thread can be understood as a sequence of atomic system functions.
- When a system test case executes
  - a thread occurs, and
  - can be observed at the port boundary of the system
- The BIG Question: where do we find (or how do we identify) threads?
- Our approach—Model-Based Testing

Threads



A “thread” is a portion of several modules that together provide a user-visible program feature.

Threads—Several Views

- A scenario of normal usage
- A use case
- A stimulus/response pair
- Behavior that results from a sequence of system-level inputs
- An interleaved sequence of port input and output events
- A sequence of transitions in a state machine description of the system
- An interleaved sequence of object messages and method executions
- A sequence of machine instructions
- A sequence of source instructions
- A sequence of atomic system functions (to be defined)

Some Choices—Threads in an ATM System

- Entry of a digit
- Entry of a personal identification number (PIN)
- A simple transaction: ATM Card Entry, PIN Entry, select transaction type (deposit, withdraw), present account details (checking or savings, amount), conduct the operation, and report the results
- An ATM session containing two or more simple transactions
- Each of these can be understood as an interleaved sequence of port level inputs and outputs.

Details of PIN Entry as a Thread

- A screen requesting PIN digits.
- An interleaved sequence of digit keystrokes and screen responses.
- The possibility of cancellation by the customer before the full PIN is entered.
- A system disposition:
  - A customer has three chances to enter the correct PIN.
  - Once a correct PIN has been entered, the user sees a screen requesting the transaction type.
  - After three failed PIN Entry attempts, a screen advises the customer that the ATM card will not be returned, and no access to ATM functions is provided.

Definition: Atomic System Function

- Definition: An Atomic System Function (ASF) is an action that is observable at the system level in terms of port input and output events.
- About ASFs
  - characterized by a sequence of port level inputs and outputs
  - could be just a simple stimulus/response pair (e.g. digit entry)
- Sample ASFs in our ATM example
  - Card entry
  - PIN entry
  - Transaction selection
  - Session termination

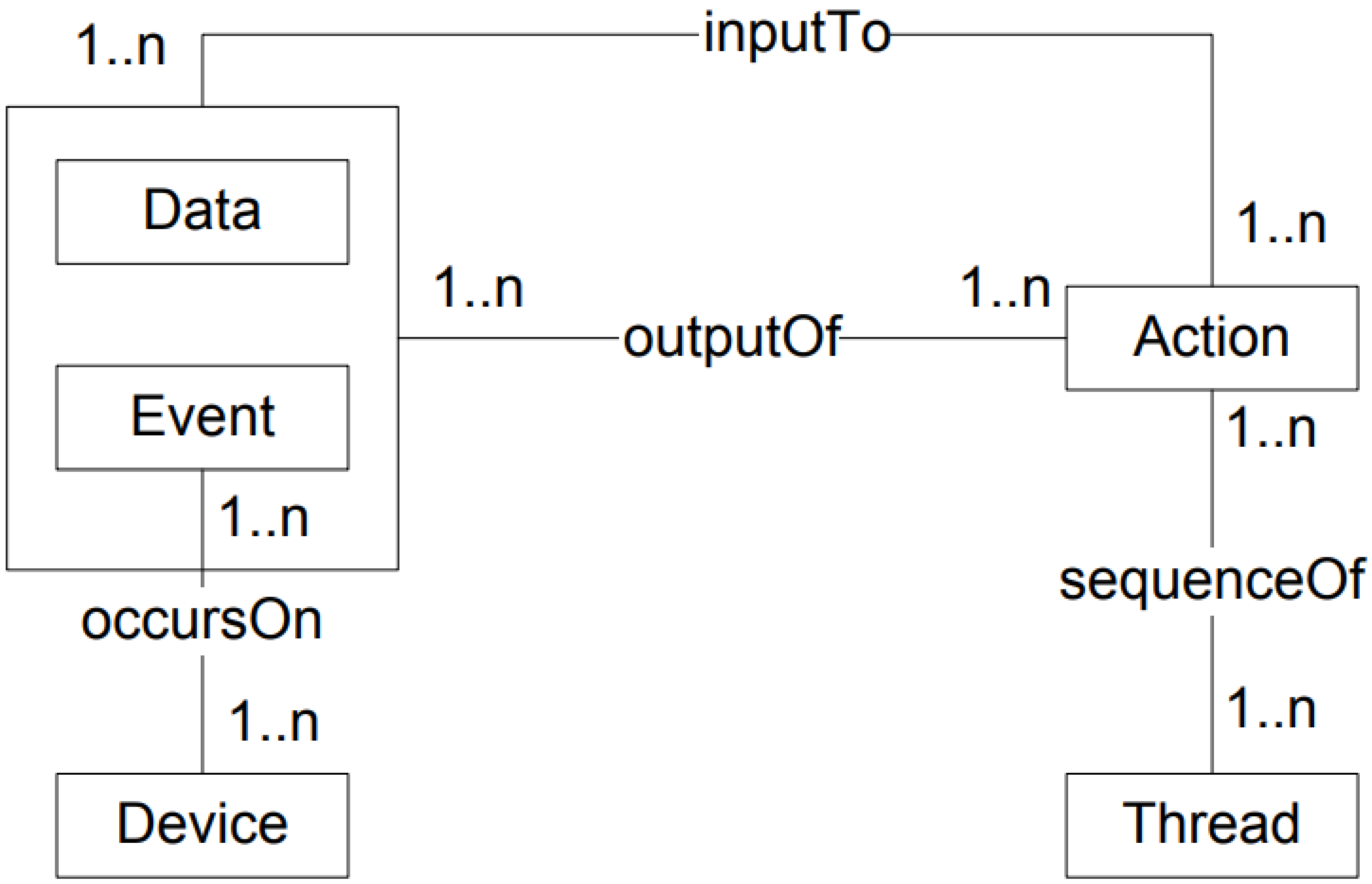
More Definitions…

- Given a system defined in terms of atomic system functions, the ASF Graph of the system is the directed graph in which nodes are ASFs and edges represent sequential flow.
- A source ASF is an Atomic System Function that appears as a source node in the ASF graph of a system.
- A sink ASF is an Atomic System Function that appears as a sink node in the ASF graph.
- A system thread is a path from a ***source*** ASF to a ***sink*** ASF in the ASF graph of a system.

Requirements Specification

- All of requirements specification models are developed on these basis concepts.
- Data
  - Inputs to actions
  - Outputs of actions
- Events
  - Inputs to actions
  - Outputs of actions
- Actions
- Threads (sequences of actions)
- Devices

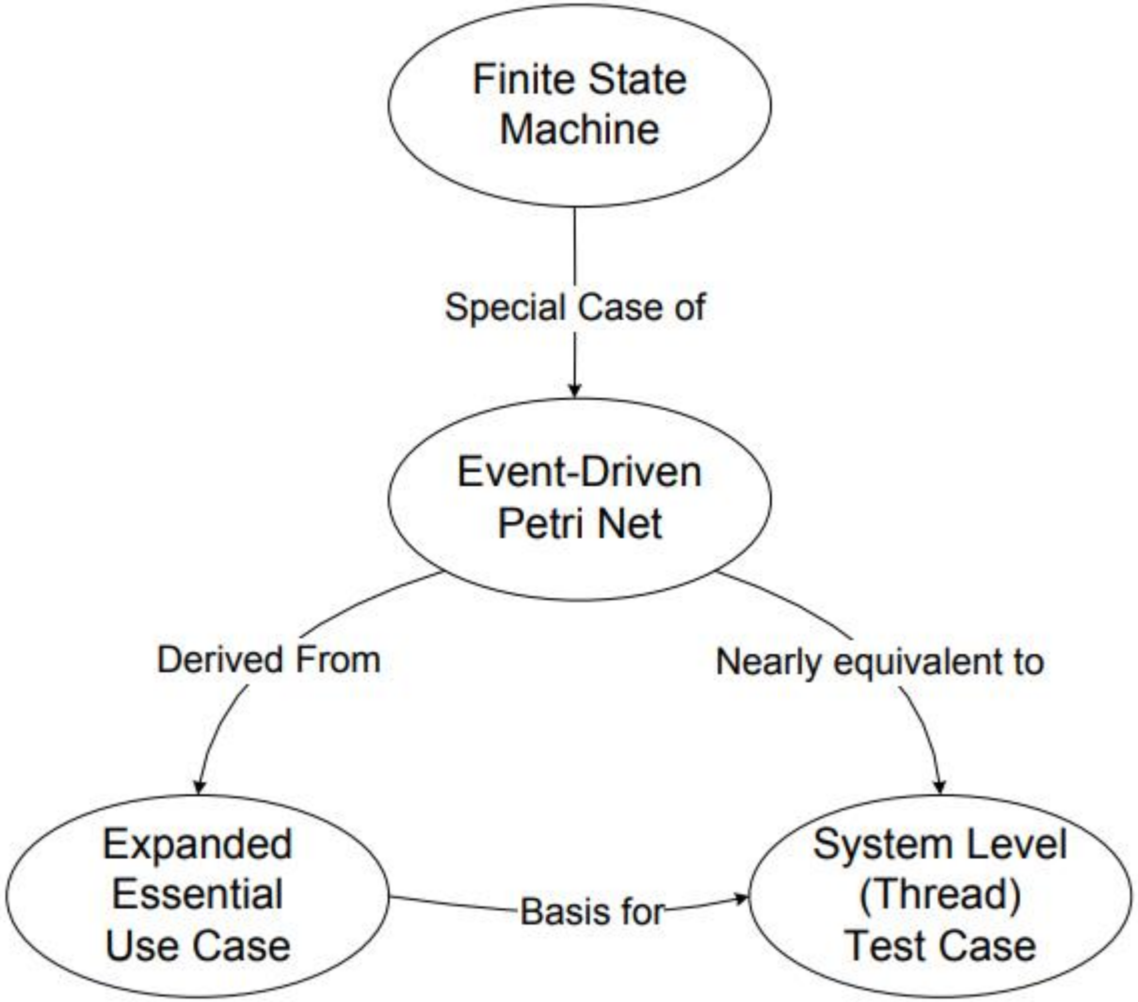
E/R Model of Basis Concepts



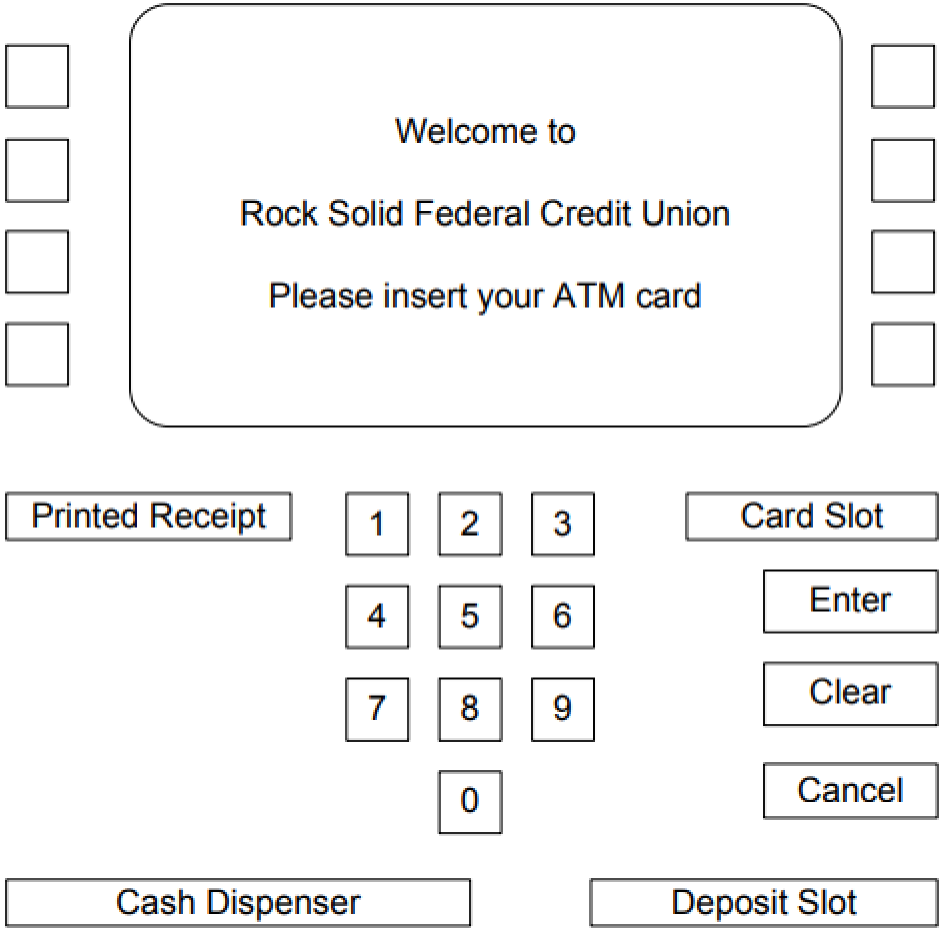
Sources of Threads

- An “Expanded Essential Use Case”
  - pre-conditions
  - interleaved sequence of input and output events
  - post-conditions
- A path in an executable model
  - finite state machine
  - Event-Driven Petri Net
- Continuing example: the Simple ATM System (SATM)

Sources of Threads—Model-Based Testing



SATM System User Interface



SATM System Screens

Screen 1

Welcome
Please insert your ATM card

Screen 2

Please enter your PIN
– – – –

Screen 3

Your PIN is incorrect.
Please try again.

Screen 4

Invalid ATM card. It will be retained.

Screen 5

Select transaction:
balance >
deposit >
withdrawal >

Screen 6

Balance is
$dddd.dd

Screen 7

Enter amount.
Withdrawals must be multiples of $10

Screen 8

Insufficient Funds!
Please enter a new amount

Screen 9

Machine can only dispense $10 notes

Screen 10

Temporarily unable to process withdrawals.
Another transaction?

Screen 11

Your balance is being updated. Please take cash from dispenser.

Screen 12

Temporarily unable to process deposits.
Another transaction?

Screen 13

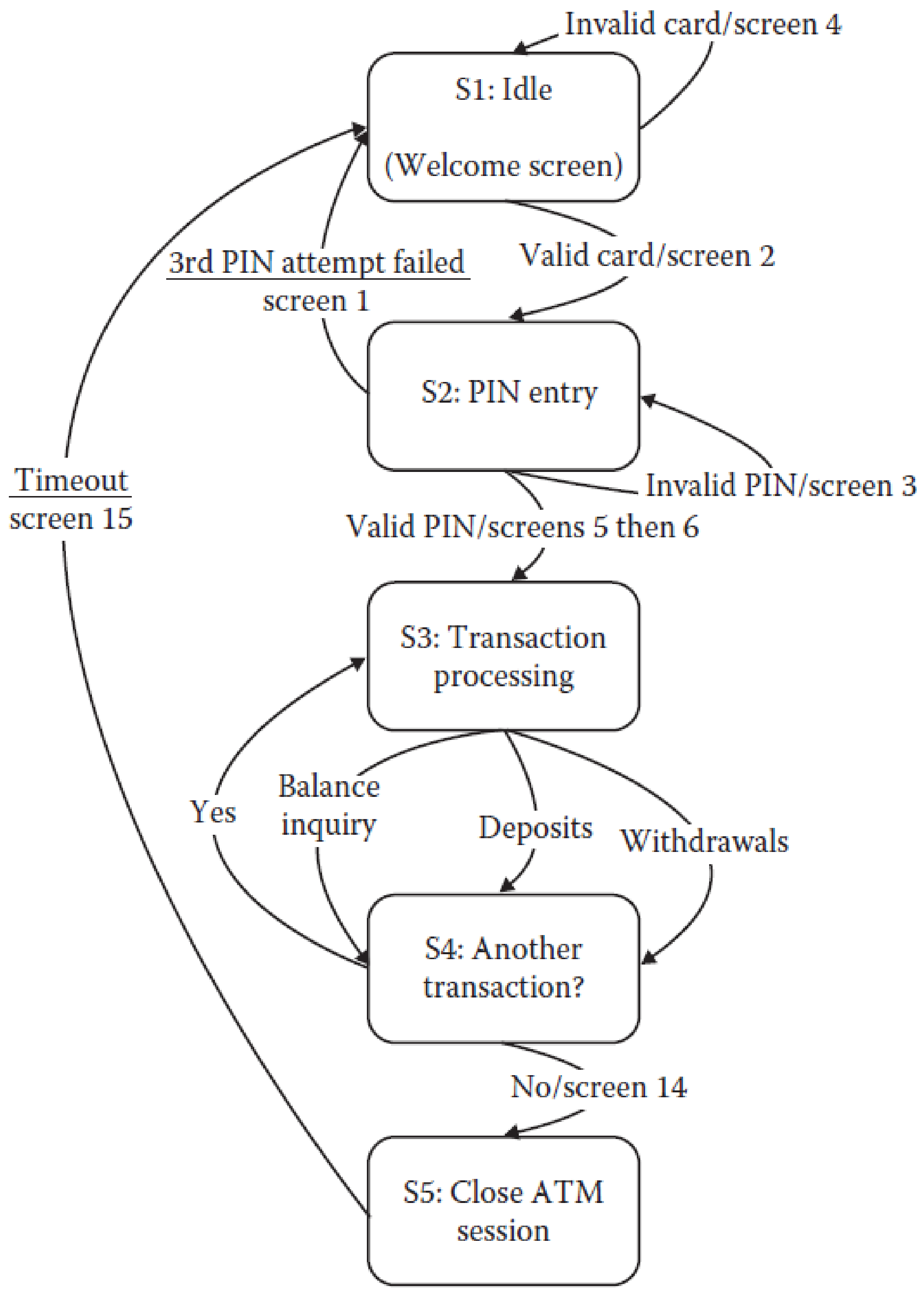
Please insert deposit into deposit slot.

Screen 14

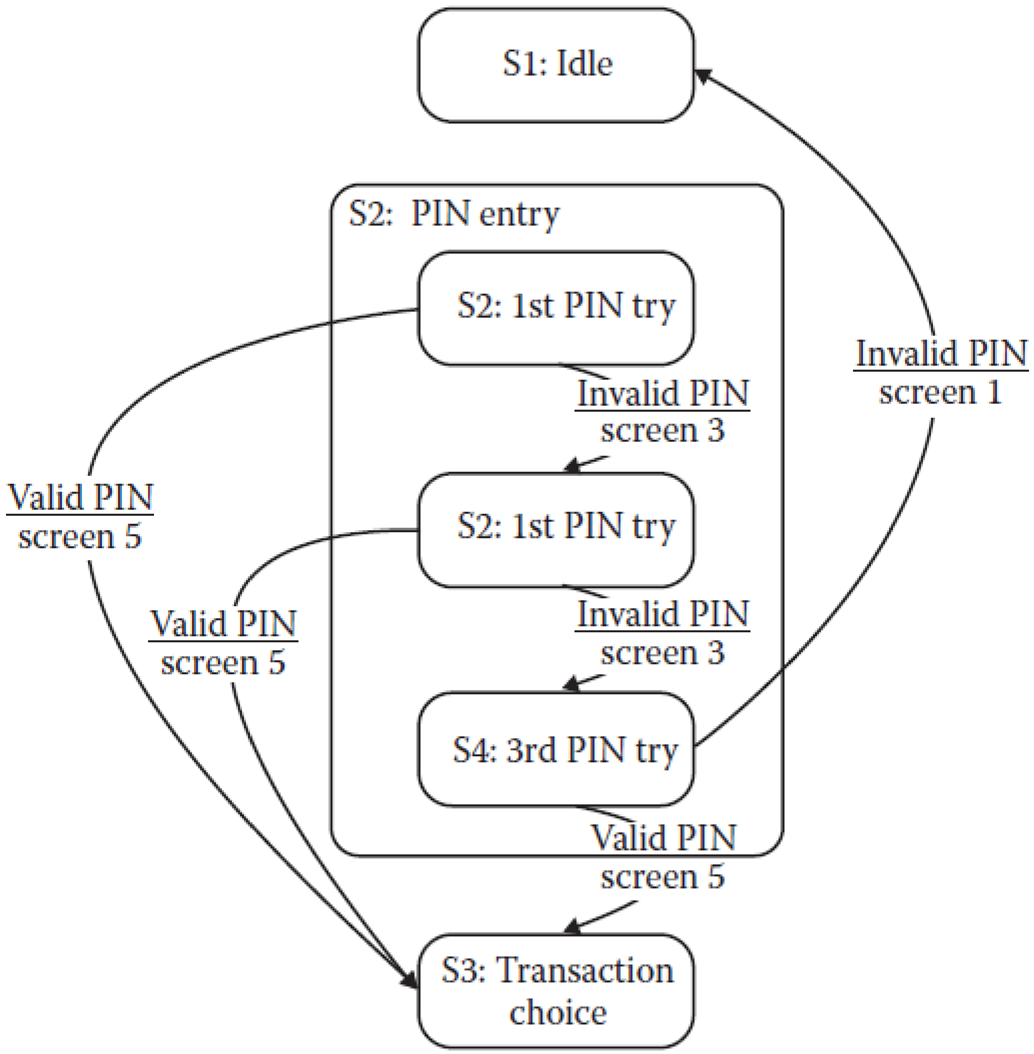
Your new balance is being printed. Another transaction?

Screen 15

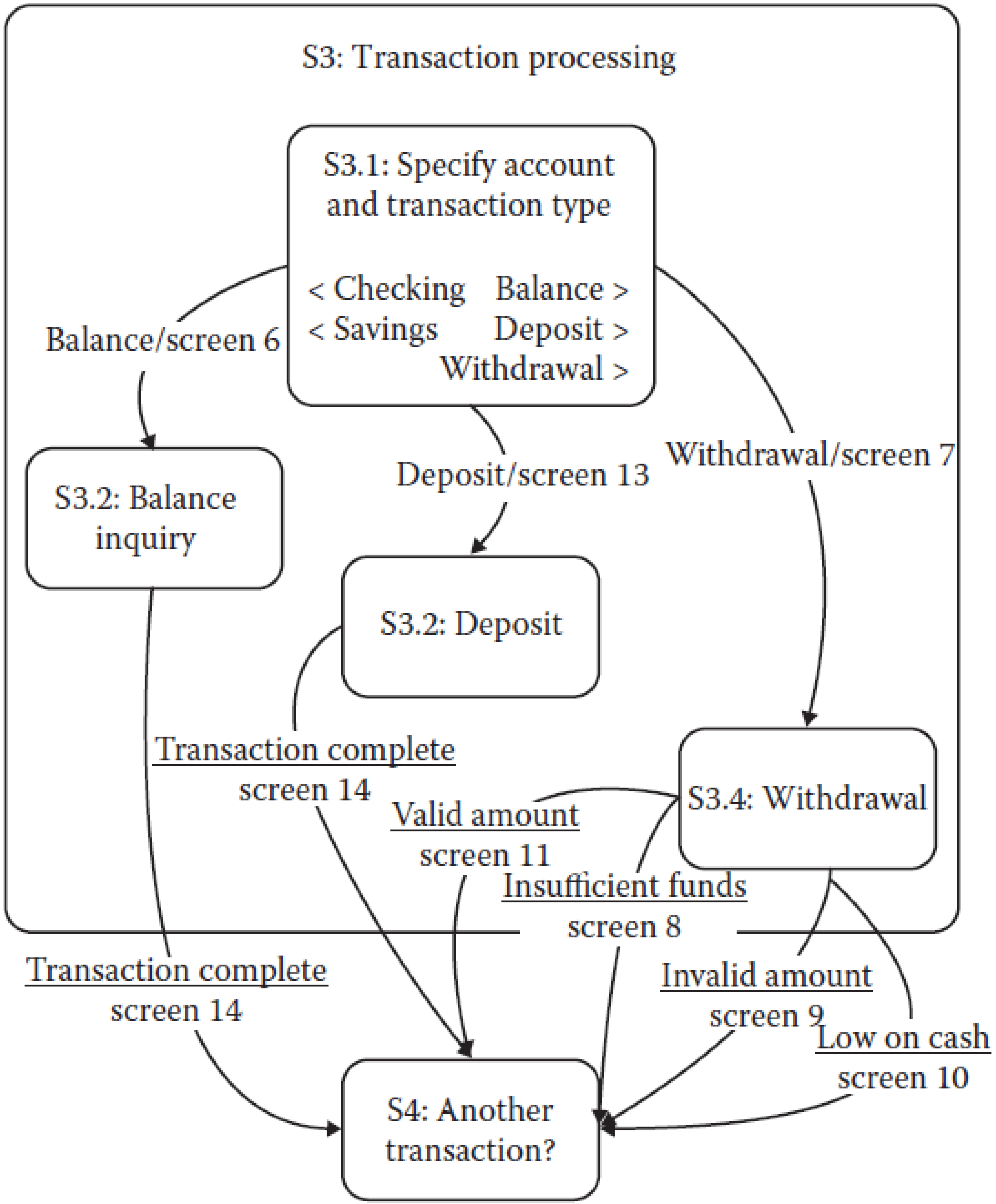
Please take your receipt and ATM card.
Thank you.



Uppermost level SATM finite state machine.



Decomposition of PIN entry state.



Decomposition of transaction processing state

Paths in the SATM PIN Try State

- Correct PIN on first try state sequence
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.3, S2.n.4, S3>
- Port Event Sequence
  - 1^{st} digit, echo "- - - *"
  - 2^{nd} digit, echo "- - * *"
  - 3^{rd} digit, echo "- * * *"
  - 4^{th} digit, echo "* * * *"
  - Enter
- Failed PIN on first try state Sequences
  - <S2.n.0, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.3, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.3, S2.n.4, S2.n.6>

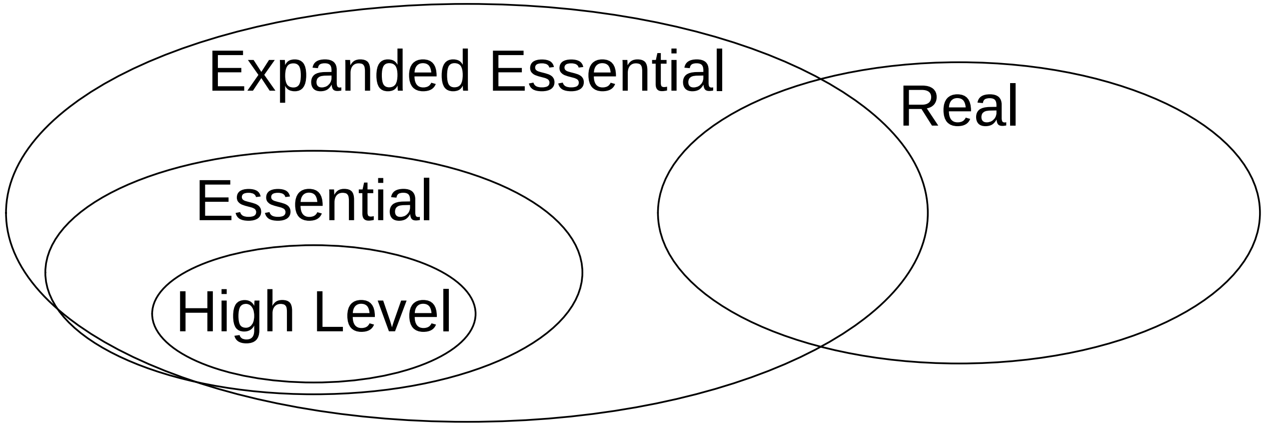
How Many Paths in the PIN Try State?

- 1st try: 1 correct + 5 failed attempts
- 2nd try: 5 failed 1st attempts * 6 second attempts
- 3rd try: 25 failed 1st and 2nd attempts * six third attempts
- Do we really want to test all of these?
- This foreshadows the question of "long" versus "short" use cases.

Port Event Sequence: Correct PIN on 1st Try

Port Input Event	*Port Output Event*
	Screen 2 displayed with '- - - -'
1st digit	
	Screen 2 displayed with '- - - *'
2nd digit	
	Screen 2 displayed with '- - * *'
3rd digit	
	Screen 2 displayed with '- * * *'
4th digit	
	Screen 2 displayed with '* * * *'
(valid PIN)	Screen 5 displayed

Information Content of Larman’s Use Cases

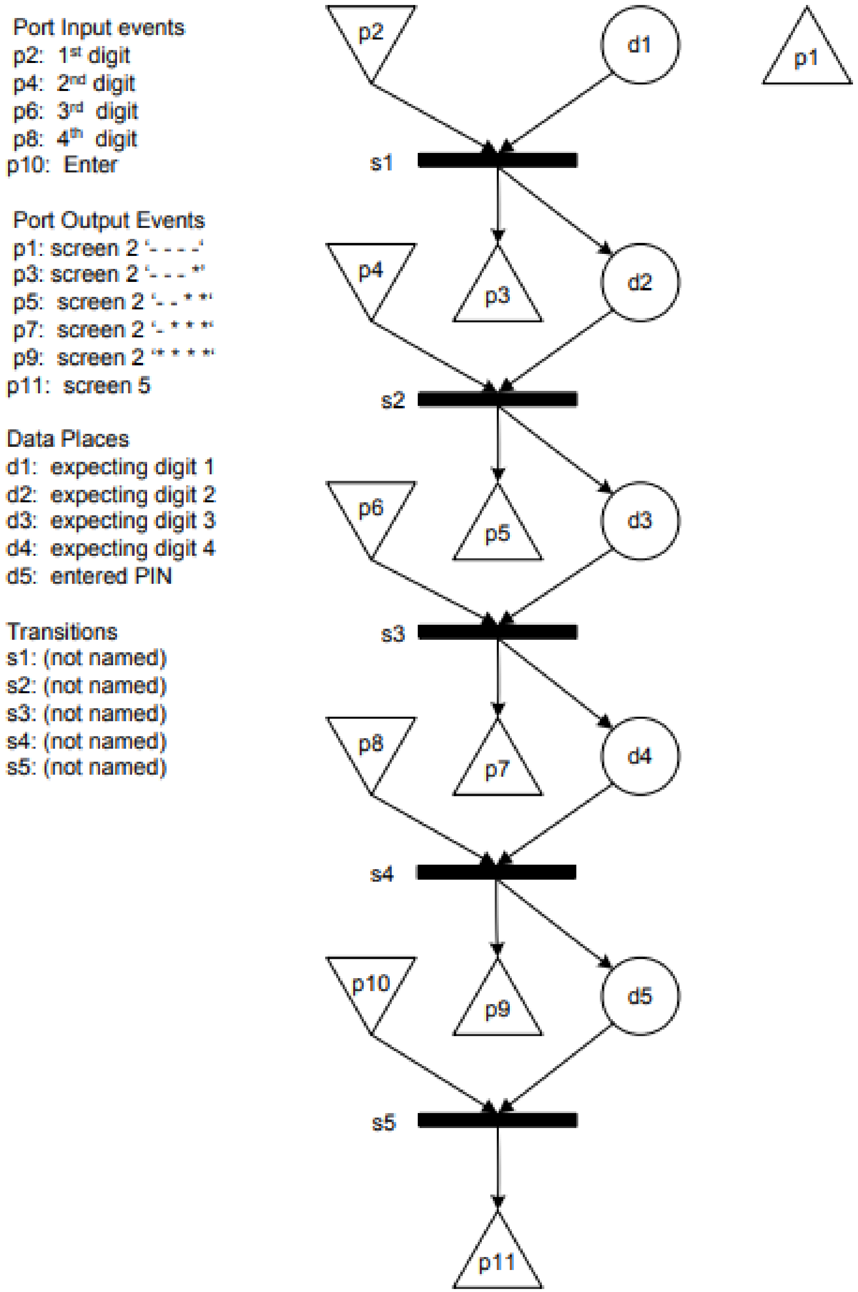


Use Case: Correct PIN on 1st Try

Use Case Name	Correct PIN entry on first try
Use Case ID	EEUC-1
Description	A customer enters the PIN number correctly on the first attempt.
Pre-Conditions	1. The expected PIN is known
	2. Screen 2 is displayed
Event Sequence	
Input events	Output events
	1. Screen 2 shows '- - - - '
2. Customer touches 1st digit	3. Screen 2 shows '- - - * '
4. Customer touches 2nd digit	5. Screen 2 shows '- - * * '
6. Customer touches 3rd digit	7. Screen 2 shows '- * * * '
8. Customer touches 4th digit	9. Screen 2 shows '* * * * '
10. Customer touches Enter	11. Screen 5 is displayed
Post conditions	Select Transaction screen is active

System Test Case: Correct PIN on 1st Try

Test Case Name, ID	Correct PIN entry on first try, TC-1
Description	A customer enters the PIN number correctly on the first attempt.
Pre-Conditions	1. The expected PIN is ‘2468’
	2. Screen 2 is displayed
Event Sequence	
Input events	Output events
	1. Screen 2 shows '- - - - '
2. Customer touches digit 2	3. Screen 2 shows '- - - * '
4. Customer touches digit 4	5. Screen 2 shows '- - * * '
6. Customer touches digit 6	7. Screen 2 shows '- * * * '
8. Customer touches digit 8	9. Screen 2 shows '* * * * '
10. Customer touches Enter	11. Screen 5 is displayed
Post conditions	Select Transaction screen is active
Test Result, Run by	Pass, Paul Jorgensen



Event-Driven Petri Net of Correct PIN on First Try

Long versus Short Use Cases

- A “Long” use case is typically an end-to-end transaction.
- SATM example: A full traversal of the high level finite state machine, from the Welcome screen to the End Session screen: <s1, s2, s3, s4, s5>
- A “Short” use case is at the level on an atomic system function.
- Examples
  - PIN Entry
  - Transaction selection
  - Session closing

Short Use Cases

- "Short" use case is at the level on an atomic system function (ASF).
- In the directed graph of ASFs,
  - nodes are ASFs
  - edges signify possible sequential execution of ASFs
- Consider an ASF as a "Short" use case, with
  - pre-conditions
  - post-conditions
- Short use case (ASF) B can follow short use case (ASF) A if the pre-conditions of B are consistent with the post-conditions of A, that is…
- Short use cases "connect" at their pre- and post condition boundaries.

Short Use Cases for the SATM System

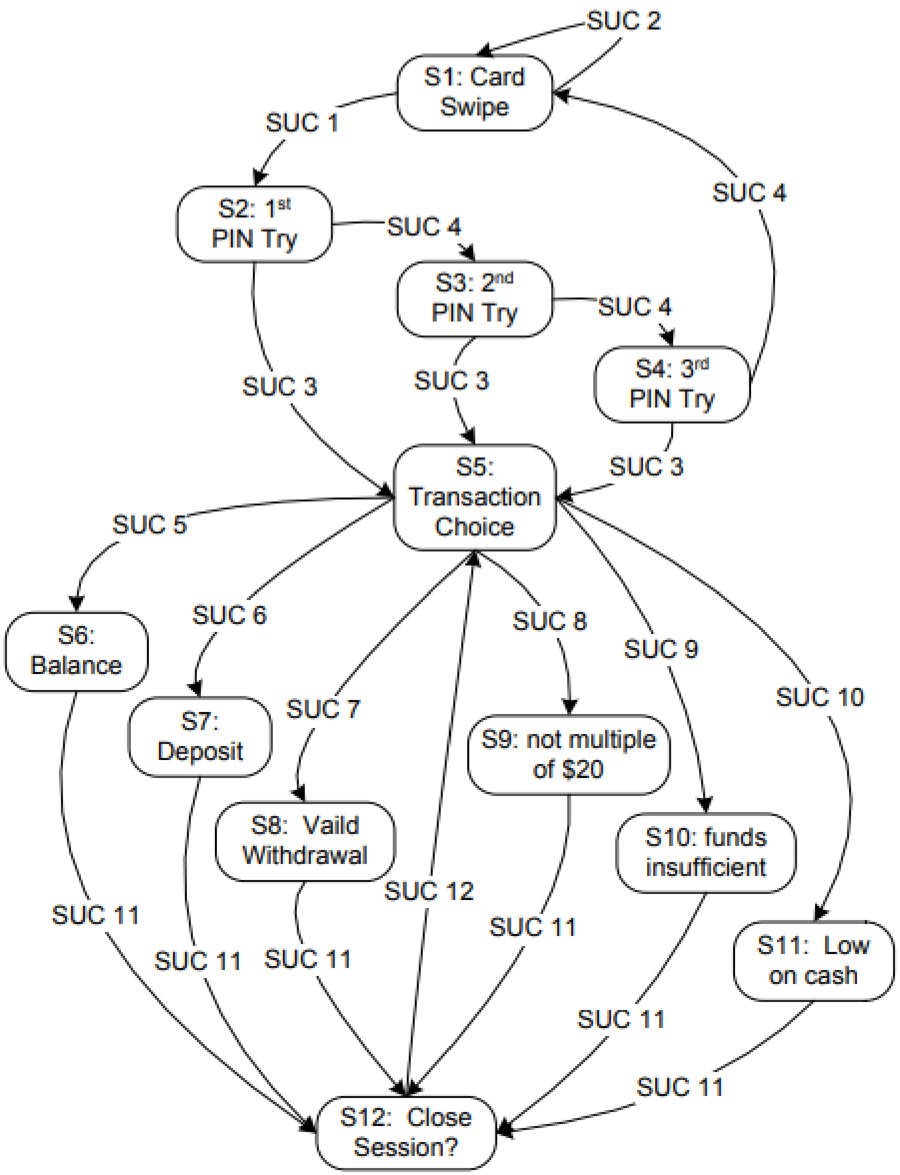
Short Use Case	Description
SUC1	Valid ATM card swipe
SUC2	Invalid ATM card swipe
SUC3	Correct PIN attempt
SUC4	Failed PIN attempt
SUC5	Choose Balance
SUC6	Choose Deposit
SUC7	Choose Withdrawal: valid withdrawal amount
SUC8	Choose Withdrawal: amount not a multiple of $20
SUC9	Choose Withdrawal: amount greater than account balance
SUC10	Choose Withdrawal: amount greater than daily limit
SUC11	Chose no other transaction
SUC12	Chose another transaction

Short Use Cases for Failed PIN Attempts

Short Use Case	Description
SUC13	Digit 1 entered
SUC14	Digit 2 entered
SUC15	Digit 3 entered
SUC16	Digit 4 entered
SUC17	Enter with valid PIN
SUC18	Cancel before digit 1
SUC19	Cancel after digit 1
SUC20	Cancel after digit 2
SUC21	Cancel after digit 3
SUC22	Cancel after digit 4
SUC23	Enter with invalid PIN
SUC24	Next PIN try
SUC25	Last PIN try

How Many Use Cases?

- 1909 “long” use cases
- 25 “short” use cases
- Ways to determine “how many?
  - Incidence with input events (cover every input event)
  - Incidence with output events (cover every output event)
  - Incidence with classes (need a use case/class incidence matrix)
- These lead directly to system testing coverage metrics.



Short Use Cases for the SATM System

System Testing with Short Use Cases

- Basic idea: a short use case is an atomic system function (ASF)
- ASFs …
  - begin with a port input event
  - end is one of possibly several port output events
- ASFs can be identified
  - in source code
  - in executable models
  - from short use cases

Model-Based Coverage Metrics

- Decision table metrics
  - every condition
  - every action
  - every rule
- Finite state machine metrics
  - every state
  - every transition
  - every path (cycles need to be addressed as in code coverage metrics)
- Petri net metrics
  - every place
  - every port event
  - every transition
  - every marking

Conclusions and Observations

- System testing is based on threads
  - thread identification is the hard part
  - automated thread execution is a good idea
- Model-Based system testing works well
- Helpful to have system level coverage metrics

Risk-Based System Testing

Risk-Based System Testing

- Risk-based testing (RBT) is a type of software testing that functions as an organizational principle used to prioritize the tests of features and functions in software, based on:
  - The risk of failure,
  - The function of their importance, and
  - Likelihood or impact of failure.

Risk-Based System Testing

- Risk = Cost * (Probability of occurrence)
- Hans Schaefer's risk categories
  - Catastrophic: deposits, invalid withdrawals
  - Damaging: normal withdrawals
  - Hindering: invalid ATM card, PIN entry failure
  - Annoying: balance inquiries
- Logarithmic weighting (low = 1, medium = 3, high = 10)

Risk-Based System Testing

- The fundamental objectives are to:
  - Design and execute testing events that involve the highest risk
  - Smoothen customer implementation process not to let risks hamper it
  - Find out possible risks or failures way ahead of time to prevent it from occurrence
  - Avoid the impact of risks on organizational deadlines, costs, and business prospects
  - Ensure a quality rich and error-free software for clients

Risk-Based System Testing

- Risk based testing can be implemented when
  - Projects have a limited time schedule, budget, resource allocation, etc.
  - There is an implementation of incremental, iterative, agile, and DevOps project methodologies
  - New projects have high-risk factors involved like new technologies, lack of skilled resources, insufficient planning etc.
  - There is the involvement of cloud-based services or the latest project approaches
  - The project is research-oriented or more complex with challenges

Selected Path Risks

Use Case Description	Use Case Probability	Cost of Failure	Risk
1st try, normal withdrawal	81.5184%	3	2.4456
1st try, deposit	4.7952%	10	0.4795
1st try, withdrawal but insufficient funds	1.9181%	10	0.1918
1st try, withdrawal not multiple of $20	4.7952%	3	0.1439
2nd try, normal withdrawal	3.2607%	3	0.0978
1st try, withdrawal, ATM low on cash	0.9590%	10	0.0959
1st try, balance inquiry	1.9181%	1	0.0192
2nd try, deposit	0.1918%	10	0.0192
Insertion of invalid ATM card	0.1000%	10	0.0100
2nd try, withdrawal insufficient funds	0.0767%	10	0.0077
2nd try, withdrawal not multiple of $20	0.1918%	3	0.0058

Non-Functional System Testing

Non-Functional System Testing

- Non-Functional Testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc.) of a software system.
- It is designed to test the readiness of a system as per nonfunctional aspects which are never addressed by functional testing.
- Non-functional testing is equally important as functional testing and affects client satisfaction.

Performance Testing

- Stress Testing
  - Checks if the system can respond to many simultaneous requests (maximum # of users, peak demands)
- Volume testing
  - Test what happens if large amounts of data are handled
- Configuration testing
  - Test the various software and hardware configurations
- Compatibility test
  - Test backward compatibility with existing systems
- Security testing
  - Try to violate security requirements

Performance Testing

- Timing testing
  - Evaluate response times and time to perform a function
- Environmental test
  - Test tolerances for heat, humidity, motion, portability
- Quality testing
  - Test reliability, maintainability & availability of the system
- Recovery testing
  - Tests system’s response to presence of errors or loss of data.
- Human factors testing
  - Tests user interface with user

Non-Functional System Testing

- Performance Testing:
  - Evaluates the overall performance of the system.
- Key elements are as follows:
  - Validates that the system meets the expected response time.
  - Evaluates that the significant elements of the application meet the desired response time.
  - It can also be conducted as a part of integration testing and system testing.

Non-Functional System Testing

- Load Testing:
  - Evaluates whether the system’s performance is as expected under normal and expected conditions.
- Key points are:
  - Validates that the system performs as expected when concurrent users access the application and get the expected response time.
  - This test is repeated with multiple users to get the response time and throughput.
  - At the time of testing, the database should be realistic.
  - The test should be conducted on a dedicated server which stimulates the actual environment.
- How many hits/requests should the system be able to handle?
- What should be its performance under these circumstances?

Non-Functional System Testing

- Stress Testing:
  - Evaluates whether the system’s performance is as expected when it is low on resources.
- Key points are:
  - Test on low memory or low disc space on clients/servers that reveal the defects which cannot be found under normal conditions.
  - Multiple users perform the same transactions on the same data.
  - Multiple clients are connected to the servers with different workloads.
  - Reduce the Think Time to “Zero” to stress the servers to their maximum stress.
- Think Time: Just like the time interval between typing your user and password.

Stress Testing

- Often requires extensive simulation of the execution environment
  - With systematic variation: What happens when we push the parameters? What if the number of users or requests is 10 times more, or 1000 times more?
- Often requires more resources (human and machine) than typical test cases
  - Separate from regular feature tests
  - Run less often, with more manual control
  - Diagnose deviations from expectation
    - Which may include difficult debugging of latent faults!

Non-Functional System Testing

- Volume Testing:
  - Evaluates the behavior of the software when a large volume of data is involved.
- Key points are:
  - When the software is subject to large amounts of data, checks the limit where the software fails.
  - Maximum database size is created and multiple clients query the database or create a larger report.
- Example– If the application is processing the database to create a report, a volume test would be to use a large result set and check if the report is printed correctly.

Non-Functional System Testing

- Usability Testing:
  - Evaluates the system for human use or checks if it is fit for use.
- Key points are:
  - Is the output correct and meaningful and is it the same as which was expected as per the business?
  - Are the errors diagnosed correctly?
  - Is the GUI correct and consistent with the standard?
  - Is the application easy for use?

Non-Functional System Testing

- Compatibility Testing:
- Evaluates that the application is compatible with other hardware /software with minimum and maximum configuration.
- Key points are:
  - Test each hardware with minimum and maximum configuration.
  - Test with different browsers.
  - Test cases are the same as those that were executed during functional testing.
  - In case the number of hardware and software are too many, then we can use Orthogonal Array Testing (OAT) techniques to arrive at the test cases to have maximum coverage.

Non-Functional System Testing

- Recovery Testing:
  - Evaluates that the application terminates gracefully in case of any failure and the data is recovered appropriately from any hardware and software failures.
- The tests are not limited to the below points:
  - Power interruption, to the client while doing CURD activities.
  - Invalid database-pointers and keys.
  - Database process is aborted or prematurely terminated.
  - Database pointers, fields and keys are corrupted manually and directly within the database.
  - Physically disconnect the communication, power turn off, turn down the routers and network servers.

Accessibility Testing

- Check usability by people with disabilities
  - Blind and low vision, deaf, color-blind, …
- Use accessibility guidelines
  - Direct usability testing with all relevant groups is usually impractical; checking compliance to guidelines is practical and often reveals problems
- Example: W3C Web Content Accessibility Guidelines
  - Parts can be checked automatically
  - but manual check is still required
    - e.g., is the “alt” tag of the image meaningful?

Installation Testing

- Before the testing
  - Configure the system
  - Attach proper number and kind of devices
  - Establish communication with other system
- The testing
  - Regression tests: to verify that the system has been installed properly and works

UI testing ("acceptance")

- Automated UI testing ("automation")
  - Scripts and such that use your app and look for failures
  - A black-box system test
- Manual tests
  - Human beings click through predetermined paths
  - Need to write down the specific tests each time
- Ad-hoc tests
  - Human beings are "turned loose" on the app to see if they can break it

Usability Test

- A usable product
  - is quickly learned
  - allows users to work efficiently
  - is pleasant to use
- Objective criteria
  - Time and number of operations to perform a task
  - Frequency of user error
- Plus overall, subjective satisfaction

Non-Functional System Testing Tools

- There are several tools available in the market for Performance (Load & Stress) testing.
- Few of them are listed below:
    - JMeter
    - Loadster
    - Loadrunner
    - Loadstorm
    - Neoload
    - Forecast
    - Load Complete
    - Webserver Stress Tool
    - WebLoad Professional
    - Loadtracer
    - vPerformer

Test Cases for Performance Testing

- Goal: Try to violate non-functional requirements
- Push the (integrated) system to its limits.
- Goal: Try to break the subsystem
- Test how the system behaves when overloaded.
  - Can bottlenecks be identified? (First candidates for redesign in the next iteration)
- Try unusual orders of execution
  - Call a receive() before send()
- Check the system's response to large volumes of data
  - If the system is supposed to handle 1000 items, try it with 1001 items.
- What is the amount of time spent in different use cases?
  - Are typical cases executed in a timely fashion?

Global Properties

- Some system properties are inherently global
  - Performance, latency, reliability, ...
  - Early and incremental testing is still necessary, but provide only estimates
- A major focus of system testing
  - The only opportunity to verify global properties against actual system specifications
  - Especially to find unanticipated effects, e.g., an unexpected performance bottleneck

Context-Dependent Properties

- Beyond system-global: Some properties depend on the system context and use
  - Example: Performance properties depend on environment and configuration
  - Example: Privacy depends both on system and how it is used
    - Medical records system must protect against unauthorized use, and authorization must be provided only as needed
  - Example: Security depends on threat profiles
    - And threats change!
- Testing is just one part of the approach

Establishing an Operational Envelope

- When a property (e.g., performance or real-time response) is parameterized by use …
  - requests per second, size of database, …
- Extensive stress testing is required
  - varying parameters within the envelope, near the bounds, and beyond
- Goal: A well-understood model of how the property varies with the parameter
  - How sensitive is the property to the parameter?
  - Where is the "edge of the envelope"?
  - What can we expect when the envelope is exceeded?

Acceptance Testing

Types of Acceptance Testing

- Acceptance testing is a formal testing conducted to determine whether a system satisfies its acceptance criteria
- There are two categories of acceptance testing:
  - User Acceptance Testing (UAT)
    - It is conducted by the customer to ensure that system satisfies the contractual acceptance criteria before being signed-off as meeting user needs.
  - Business Acceptance Testing (BAT)
    - It is undertaken within the development organization of the supplier to ensure that the system will eventually pass the user acceptance testing.

Types of Acceptance Testing

Three major objectives of acceptance testing:

- Confirm that the system meets the agreed upon criteria
- Identify and resolve discrepancies, if there is any
- Determine the readiness of the system for cut-over to live operations

Acceptance Criteria

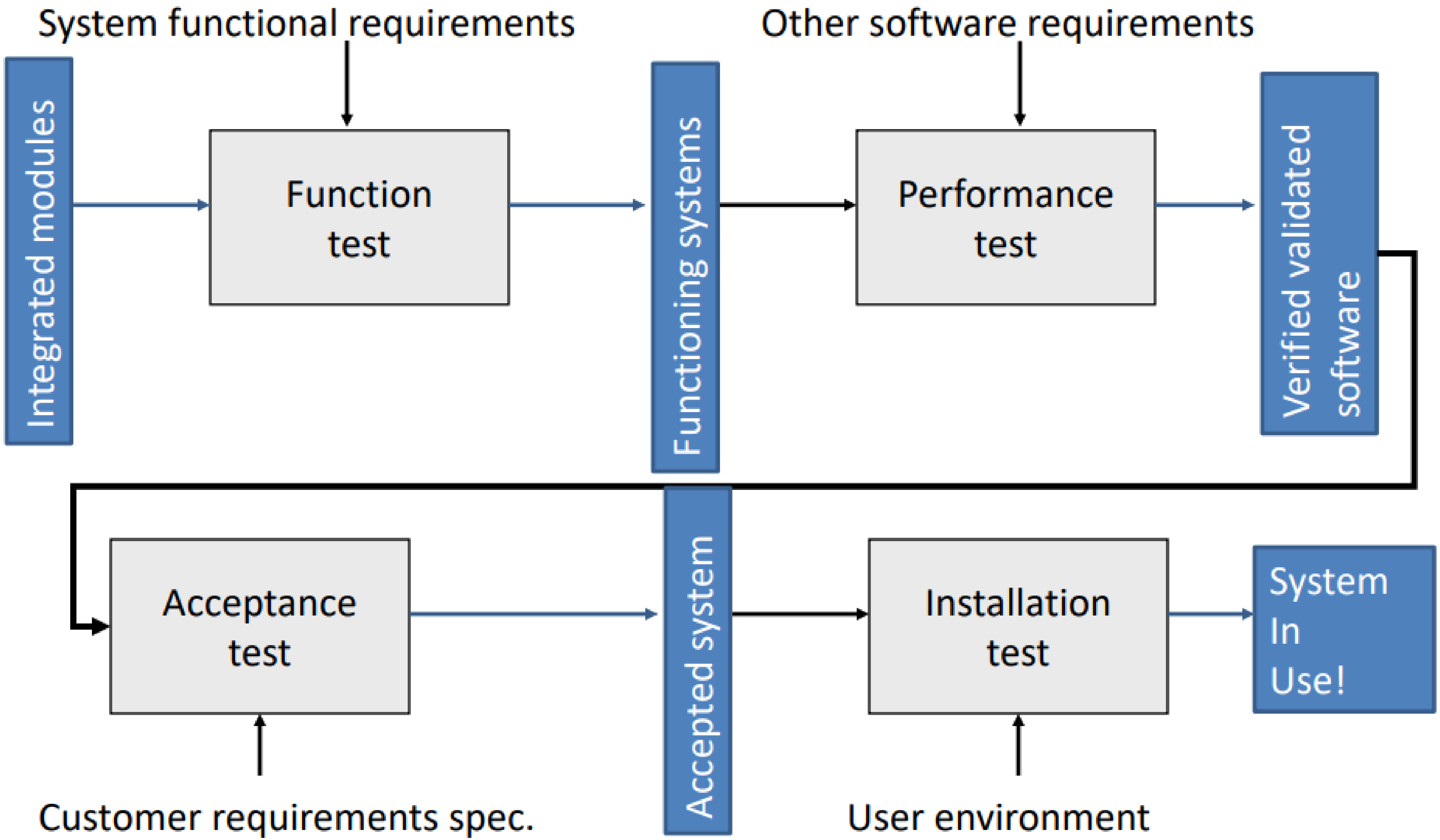
- The acceptance criteria are defined on the basis of the following attributes:
  - Functional Correctness and Completeness
  - Accuracy
  - Data Integrity
  - Data Conversion
  - Backup and Recovery
  - Competitive Edge
  - Usability
  - Performance
  - Start-up Time
  - Stress
  - Reliability and Availability
  - Maintainability and Serviceability
  - Robustness
  - Timeliness
  - Confidentiality and Availability
  - Compliance
  - Installability and Upgradability
  - Scalability
  - Documentation

Acceptance Test Execution

- The acceptance test cases are divided into two subgroups
  - The first subgroup consists of basic test cases, and
  - The second consists of test cases that are more complex to execute
- The acceptance tests are executed in two phases
  - In the first phase, the test cases from the basic test group are executed
  - If the test results are satisfactory then the second phase, in which the complex test cases are executed, is taken up.
  - In addition to the basic test cases, a subset of the system-level test cases are executed by the acceptance test engineers to independently confirm the test results
- Acceptance test execution activity includes the following detailed actions:
  - The developers train the customer on the usage of the system
  - The developers and the customer co-ordinate the fixing of any problem discovered during acceptance testing
  - The developers and the customer resolve the issues arising out of any acceptance criteria discrepancy

Acceptance Testing

- Goal: Demonstrate system is ready for operational use
  - Choice of tests is made by client
  - Many tests can be taken from integration testing
- Majority of all bugs in software is typically found by the client after the system is in use, not by the developers or testers. Therefore two kinds of additional tests:
- Alpha test:
  - Sponsor uses the software at the developer's site.
  - Software used in a controlled setting, with the developer always ready to fix bugs.
- Beta test:
  - Conducted at sponsor's site (developer is not present)
  - Software gets a realistic workout in target environment
  - Potential customer might get discouraged


System functional requirements
Other software requirements
Integrated modules
Function test
Functioning systems
Performance test
Verified validated software
Acceptance test
Accepted system
Installation test
System In Use!
Customer requirements spec.
User environment

Summary

- System testing is verification
  - System consistent with specification?
  - Especially for global properties (performance, reliability)
- Acceptance testing is validation
  - Includes user testing and checks for usability
- Usability and accessibility require both
  - Usability testing establishes objective criteria to verify throughout development