417 Exam 2 Review

1 vocabulary

1.1 requirements

Mathur 1.3 Requirements specify the functions that a product is expected to perform.

1.1.1 types

functional
Wikipedia "Functional Rquirement"

In Software engineering and systems engineering, a **functional requirement** defines a function of a system or its component. A function is described as a set of inputs, the behavior, and outputs.
non-functional (NFR)
Mathur 1.2.1
Wikipedia https://en.wikipedia.org/wiki/Non-functional_requirement. The list is HUGE!
performance
security
capacity
reliability
usability
safety
constraints

1.2 test categories

1.2.1 by imp technique

dynamic tests
Mathur 1.12
static tests
Mathur 1.12
manual
automatic
exhaustive
Mathur 1.3.1 Testing a program on all possible inputs.

1.2.2 by design info

Mathur table 1.14 (in section 1.18.1)
black box
white box

1.2.3 by relation to phase

Mathur Table 1.5 (Section 1.18.2)
Related: (V-model drawing in 1.18.5)

acceptance - requirements
system - architecture
integration - subsystem (module) design
module test - detailed design
unit test - implementation

1.2.4 by test goal

Mathur 1.18.3 (See also, Wikipedia non-functional testing)
stress tests
recovery tests
load tests
volume tests
configuration tests
compatibility tests
security tests
reliability tests
Mathur 1.2.2
correctness vs. reliability
Mathur 1.4.1, 1.4.2
usability tests
regression tests
Mathur 5.1. Regression testing refers to that portion of the test cycle in which a program P′ is tested to ensure that not only does the newly added or modified code behaves correctly, but also that code carried over unchanged from the previous version P continues to behave correctly.

Wikipedia (with changes)
Regression testing tests whether software which was previously developed and tested still performs the same way after it was changed or interfaced with other software.

1.2.5 Artifact under test

Mathur 1.18.4

1.3 equivalence classes

Mathur 2.3
The equivalence classes are created assuming that the program under test exhibits the same behavior on all elements, that is tests, within a class.

1.3.1 blocks

slides

1.3.2 characteristic

slides

1.3.3 domain

Mathur 1.3.1

1.3.4 input types

Mathur 1.3.2
invalid
valid

1.3.5 uni dimensional

Mathur 2.3.4
[When identifying equivalence classes]... considers one input variable at a time.

1.3.6 mutli-dimensional

Mathur see Figure 2.4 and surrounding discussion.
See also Lecture 7 (may be mislabelled as lecture 6 in the deck) discussion of homework problem 2

1.4 fault nomenclature

Lecture 2 slides. Note that we will NOT follow Mathur's definition of error.

Ammann and Offutt, p 12
Mathur 1.1.1

1.4.1 fault

1.4.2 error (infection)

1.4.3 failure

1.4.4 defect

1.5 fault activation/failure model

RIPR in lecture 2
RIPon page 13 of Ammann and Offutt (1st ed.)

1.5.1 reachability

1.5.2 infection

1.5.3 propagation

1.5.4 reveal

1.6 practices

1.6.1 TDD

Lecture 11

1.6.2 CodeKata

Lecture 11

1.7 java assert

See Lecture 2, slides 11,12.

1.8 tests

1.8.1 test case

Ammann and Offutt
"A test case is composed of the test case values, expected results, prefix values, and postfix values necessary for a complete execution and *evaluation* of the software under test."

We often use "Test Case" to mean the specification of one test.
Lecture 3, slide 5.
preconditions
parameters
the combination of state and input values needed to specify a test.
state
Mathur 1.5.4

"The state of a program is the set of current values of all its variables and an indication of which statement in the program is to be executed next. One way to encode the state is by collecting the current values of program variables into a vector known as the state vector."

domain
Mathur 1.3.1 Input Domain: The Set of all possible iputs to a program P is known as the input domain, or input space of P.

1.8.2 test set

A set of test cases.

1.8.3 Coverage Criteria

Subsumption
Lecture 10, Ammann and Offutt, section 3.2.3

1.8.4 Test Suite

One of more test sets that share some commonality, usually related to purpose or the scope of the tests. For example they are part of the system smoke test, or they exercise the modules in a particular subsystem.

1.8.5 infeasible test

See infeasible in control flow graph.

1.9 metrics

Lecture

1.9.1 coverage

Mathur 6.2.1

Lecture slides, lecture 4, slides 11-16

1.9.2 defect density

1.10 tooling

1.10.1 test harness

Mathur, 1.5.3
Junit is a library for creating test harnesses.

1.10.2 test fixture

Similar to a test harness, but generally with more attention to controlling the execution environment
https://github.com/junit-team/junit4/wiki/Test-fixtures

1.10.3 instrumentation

Lecture 2, slide 10.
Wikipedia, Instrumentation (computer programming)
https://msdn.microsoft.com/en-us/library/aa983649(v=vs.71).aspx

1.10.4 stubs, mocks, proxies

Lecture 9 and links in reading assignment at end of Lecture 9.

1.11 Traceability

Lecture 11 (Slides 20)
Mathur 6.2.14

2 inconvenient truths

Section 1.3 "Paradoxes and main principles"

"The test team activities are software risk limitation (mitigation) activities."

2.1 can not prove correctness

2.1.1 drives focus on risk

2.1.2 executing a fault does not guarantee failure

2.2 cannot afford exhaustive coverage

2.2.1 drives interest in equivalence classes

2.3 number of faults is unknowable

2.3.1 drives indirect measures like LOC

2.4 faults tend to cluster (not uniformly distributed)

3 Why Test

3.1 damage mitigation

3.1.1 Types of damage

L1, slide 9

3.2 risk evaluation

3.3 Testing vs. QA and debugging

Quality Assurance is an encompassing concern, it includes techniques for preventing the creation of faults. Testing contributes to QA by producing measurements that can be used to measure the effect of quality assurance initiatives. Debugging relies upon testing to expose failures, but, unlike testing, actually involves locating and correcting the fault.

3.4 beziers testing maturity model

Ammann and Offutt
1.1.2

4 How do we test

4.1 test design process

When can tests be designed

4.1.1 requirements

Mathur 1.3
Lecture 3, slides 6-9
define desired behavior
The standard for correct results.
Mathur 1.3.1: A program is considered correct if it behaves as desired on all possible test inputs.
vary by organization

sources
customers and stakeholders
requirements engineers
product owner
customers
management
testers
prior products

4.1.2 requirements analysis

depending upon completeness -- may require substantial interpretation
Define input domain
Define expected behaiovr
identify parameters
include state

4.1.3 input partitioning

Lecture 3, slides 15 et seq.
wikipedia, "Equivalence Partitiioning"
equivalence classes
Mathur 2.3
The equivalence classes are created assuming that the program under test exhibits the same behavior on all elements, that is tests, within a class.

characteristic/membership rule
heuristics

requirements
Lecture 3, slide 19
complete
disjoint
choice criteria
valid /invlid inputs
Mathur 1.3.2
vs. behaviors
partitioning is about identifying sets of inputs that produce the same result under test, thus equivalence classes are always driven by differences in behavior (or at least in the behavior the test evaluates).
boundary conditions
wikipedia "boundary testing"
Lecture 7 slides 11 and 12

uni dim vs multi-dim
Mathur 2.3.4
Lecture 7 slides 7 to 12

4.1.4 operational (use) profile

Mathur 1.4.3
affects risk: highest risk is in most common usages.

4.1.5 test set generation

combinational from unidimensional partitions
Mathur
one per
Mathur (2.2) seems to consider only this approach, at least in the first parts of the book.
" [input partitioning] ... allows a tester to select exactly one test from each equivalence class, resulting [for a partition of N equivalence classes] in a test suite of exactly N tests.

each choice
Assuming a multiple parameter input space and that each parameter subdomain has been partitioned (divided into blocks), then "each choice" requires that each block appear in at least
one test case. See lecture 3 slides.
all combinations
See Lecture 3 slides.
pairwise
Base Choice
Lecture 7, slide 13-15
Multiple Base Choice
Lecture 7
Slide 16
point representative from multidimensional partitions
Mathur seems to default to this strategy ... one test per block.
one per
model driven
Mathur 1.13
Modelling techniques
control flow graph
Mathur 1.14
See also 6.1, 6.1.2, 6.1.3, 6.1.4, and y.2
basic block
Mathur 1.14.1
conditional
Mathur 1.14.1 as basic block
complete paths
Mthur 1.14.3
infeasible paths
1.14.3 see infeasible path in control flow graphs.

nodes and edges
independent paths
Wikipedia
structured modules
data flow graph
Mathur 6.3
Lecture 8
Data life cycle
non-extant
define

use
kill
faults detectable in pair wise transitioins.
limitations
predicate satisfaction
Not part of exam 1
from flow graphs
Slides, Lecture 6.
Ammann & Offutt (1st ed) Section 2.4
node coverage
edge coverage
edge pair
See Lecture 13 for another example.
basis paths
Wikipedia, Basis path Testing
and cyclomatic complexity
Wikipedia, cyclomatic complexity, later part
from data dependency graphs
Slides, Lecture 8
From Logic Structure (predicates)
condition or predicate coverage
Lecture 10,

combinatorial Testing
Lecture 17 -- especially the definition.
Interaction complexity
relation to pair-wise, t-wise
growth patterns
strength -- applicability
branch coverage
Lecture 10
branch AND condition coveraage
Lecture 10
MCDC
Lecture 10 -- Mathur 6.2.8
MC/DC criteria requires:

- Condition Coverage
- Branch Coverage
- and every condition must be able to independently *determine* the predicate.

Predicate Determination
Lecture 10 and
Ammann and Offutt, section 3.2.4.
Via truth table
via exclusive or reduction

Cause and Effect Graphs
Lecture 10,

Mathur 2.6.1
Sizing Test Sets
See Ammann and Offutt ch 4. sec. 2 (On digital reserve).
All Combinations
Each Choice
Pair-Wise
Base Choice Coverage

4.1.6 Test Design Documents

Lecture 10
Purpose
Users
Test Case (def.)
See Lecture 9, also Mathur, 1.5.2. Notice in this course, we treat initial conditions separately and mathur treats those as just another type of input.
A complete design for one Test Case requires:
<initial conditions, input values, expected behavior>

4.2 behavior vs. requirements

4.3 manual vs. automated

4.4 TDD

Lecture 11

5 when do we test

5.1 earliest test design opp'ty

5.2 v-model design/test correlations

Mathur Figure 1.24

5.3 In iterative and agile dev models

Mathur 1.18.5

5.4 Regression testing

See notes at regression tests in vocabulary part

6 when do we stop testing

It's a business decision that varies with risk, resources, and application context. One approach to the decision: Testing effort should be commensurate with the perceived risk and cost of failure.

7 relative advantages of different tests

7.1 white box vs black box

Lecture 3, slides 10-12

8 How do we manage test costs

8.1 automation

8.1.1 of test generation and selection

8.1.2 of test execution and evaluation

8.2 eliminate redundant tests

8.3 a mixture of test techniques

8.4 focus test effort on high risk areas

9 Test Management

Lecture 11

9.1 Product mgmt

Lecture 11
ISTQB and other links at end of Lecture 11

9.2 Engineering mgmt

Lecture 11
ISTQB and other links at end of Lecture 11

9.3 QA Mgmt

Lecture 11
ISTQB and other links at end of Lecture 11

9.4 Roles and Responsiblities

Lecture 11
ISTQB and other links at end of Lecture 11

9.5 Test Stakeholders

Lecture 11
ISTQB and other links at end of Lecture 11

9.6 Test Process

Lecture 11
ISTQB and other links at end of Lecture 11

9.6.1 Planning

Lecture 11
ISTQB and other links at end of Lecture 11

9.6.2 Specification

Lecture 11

ISTQB and other links at end of Lecture 11

9.6.3 Execution

Lecture 11
ISTQB and other links at end of Lecture 11

9.6.4 Recording

Lecture 11
ISTQB and other links at end of Lecture 11

9.6.5 Check for Completion

Lecture 11
ISTQB and other links at end of Lecture 11

9.6.6 Test Closure Activities

Lecture 11
ISTQB and other links at end of Lecture 11

9.7 DevOps

Lecture 11
Note relationship to org structure.

9.8 Documents/artifacts

9.8.1 Test Plan

Lecture 11

9.8.2 Traceability and Defect Management

Lecture 11
Bugzilla
Lecture 11
Jira
Lecture 11

9.8.3 Defect Reports

Lecture 11

9.8.4 Test Case/Test Script

Lecture 11
also, see https://stackoverflow.com/questions/1550157/manual-test-script-templates

9.8.5 Defect Classification

Lecture 11

10 test tools

10.1 static analysis

Lecture 4, slide 23

10.1.1 complexity

equation

10.1.2 some data flow analysis

10.1.3 style checker

10.1.4 find bugs

10.1.5 manual

code reviews
design reviews

10.2 instrumentation

Lecture 2, slide 10

10.2.1 code coverage

see test quality/metrics
EMMA
Lab 3

10.2.2 SUT state capture

10.3 test harnesses

10.3.1 junit topics

Lecture 2
@Before
@After
@BeforeClass/@AfterClass
@Test
@Test(expect ...)
how to detect missing exceptions
Lecture 13
@RunWith (parameterized tests)
Lecture 5, slides 11 - 17.
assertSame()/assertEquals()
other assertions in library
fail()
@Test(timeout...)
best practices
Lecture 2
parallel hierarchy in test source folder

production and test code are separate
easier to organize suites
access to SUT

name test classes by intent
each class has many test methods
each test method is one test case
No side effects
Do not assume tests execute in any particular order
Do not use constructor as test setup (you will need it for runwith)
Externalize test data whenever possible
timeouts
advantages to tests in special main()

10.4 mocks and stubs

10.4.1 POJO

"Not the point" slide in Lecture 12
(POJO stands for Plain Old Java Object)

10.4.2 Mockito

Lecture 12
mocked interface
Lecture 12
control interface
defines stub behavior
supports creation
defines verifications
key methods
Lecture and mockito documention. See links in
when
verify
atLeast
atMost
inOrder
any
as protocol verifier
Lab 4, Lecture 12 slide 15

10.5 test evaluation

10.5.1 assertions

Lecture 2

10.5.2 oracle

A test oracle is a piece code that can examine an input and a proposed output and tell you if the output is correct for that input.
It is valuable, especially in parameterized tests because :

1) sometimes we can recognize a correct output with much less computation

than it
takes to recompute the output.
2) If we have an appropriate oracle, a single JUnit test function can be used to evaluate all members of a test set.

10.5.3 evidence based

assumption free if possible.

11 Test Environments/contexts

11.1 Web Apps

11.1.1 TCP/IP

Lecture 14
See link on "Network Basics" slide.
Ports
Connections
Client/Server pattern
Server as listener, client as initiator.
As distributed application

network tools
Wireshark

11.1.2 HTTP

Lecture 14
And links in reading slide at end.

See also https://docs.postman-echo.com/ With this service, and curl, you can see the HTTP request and the returned response.
Methods
Post/Get
Lecture 16
and https://docs.postman-echo.com/
Request Structure
Lecture 16
Response Structure
url structure
See:
https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.2.0/com.ibm.cics.ts.internet.doc/topics/dfhtl_uricomp.html

url encoding
https://www.tutorialspoint.com/html/html_url_encoding.htm
also
https://docs.oracle.com/javase/7/docs/api/java/net/URLEncoder.html
Status Codes
See http://www.restapitutorial.com/httpstatuscodes.html
AND
https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.ht

ml

11.1.3 Servlets

Lecture 14, 15
life cycle
Context root
ServletRequest responsibilities
Servlet Response Responsibilities
JSP
Lecture 15, reading assignment at end.
Lecture 16, and readings at end of Lecture 15.
scriptlet
EL expression
Lecture 16 & 17
Page Directive

11.1.4 Servlet Engines

Lecture 14
Tomcat
Glassfish

11.1.5 Server-side frameworks

Lecture 14
Struts, JSP, Spring MVC, Tapestry, JSF (JavaServerFaces
Front Door Pattern

11.1.6 Testing Challenges

Lecture 14

11.2 Test Tools

11.2.1 HTTP Client

11.2.2 network tools

Wireshark

11.2.3 curl

See https://docs.postman-echo.com/
If you don't have curl on your windows machine, I suggest you install git-bash. You get both a friendly shell and the most important of the unix utilties -- including curl -- with one easy install.
See https://git-for-windows.github.io/

11.2.4 Echo services

12 test quality/metrics

12.1 coverage

Lecture 4. Slides 11 to end.
See Metrics/coverage in vocabulary branch.

12.1.1 types of

line
branch
condition
predicate
see logic, graph, and data flow topics

12.1.2 limiting assumptions for line-based

correlation between reach and propagation
faults uniformly distributed

13 Exercises

Amman and Offutt, Section 1.2, # 3

example in slide 14 of http://crystal.uta.edu/~ylei/cse4321/data/isp.pdf

Ammann and Offutt, Section 4.2, # 2 and #3