

Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan

Babeş-Bolyai University

Cluj-Napoca

2018-2019



Outline

- **Surprises!**
- Class schedule
- Grading
 - Seminar activity
 - Laboratory activity
- Surprises!



UNIVERSITATEA BABEŞ-BOLYAI
DEPARTAMENTUL PENTRU PREGĂTIREA
PERSONALULUI DIDACTIC
CLUJ-NAPOCĂ

1937
UNIVERSITATEA BABEŞ-BOLYAI

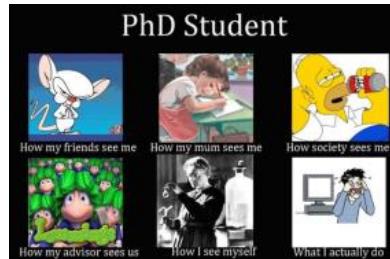
LICEUL DE INFORMATICĂ "TIBERIU POPOVICIU"

HOME DESPRE NOI ELEVI PROFESORI ACTIVITATE UTIL ASOCIAȚIA IT CONTACT

KEEP
CALM
I'M A
TEACHING
ASSISTANT



-Leonardo Da Vinci



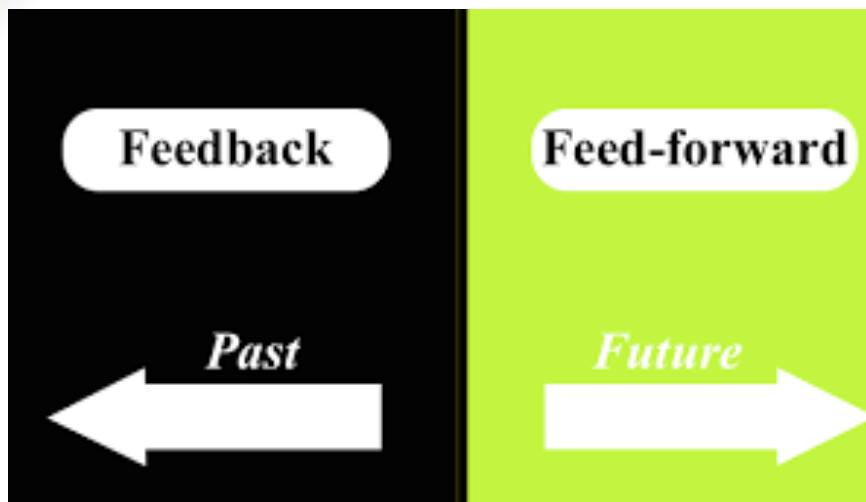


Q A



Surprise!

Feedforward!



1. Plan to Attend – how many?
 - 1.1. Lectures (12)
 - 1.2. Laboratories (6)
 - 1.3. Seminar (6)
2. Expect/want to learn/do
 - 2.1. Lectures
 - 2.2. Laboratories
 - 2.2. Seminars
3. Lectures/Laboratories/Seminars – how
 - 3.1. given/individual/team study
 - 3.2. assignment – in class/ home
 - 3.3. Lecture – By Students For Students
4. Grading
 - 3.1. Normal/retake session
 - 3.2. Activities to be evaluated
5. Other remarks

Outline

- Surprises!
- **Class schedule**
- Grading
 - Seminar activity
 - Laboratory activity
- Surprises!

Class schedule (tentative)

Software Systems Verification and Validation (TENTATIVE)				
Week	Date	Lecture	Seminar	Laboratory
1	1March	Intro+Inspection	1. Inspection	1. Inspection
2	8March	Testing. BBT		
3	15March	WBT	2. BBT	2. BBT
4	22March	IT firm – Altom (different hours)		
5	29March	Levels of testing	3. WBT	3. WBT
6	5April	Symbolic execution		
7	12Apr	IT firm – EVOZON (different hours)	4. Levels	4. Levels
8	19April	Correctness		
9	26April 29Apr-5May	Model checking Holiday	5. Web	5. Web
10	10May	It firm – Endava (different hours)		
11	17May	Presentations + Security	6. Agile	6. Agile
12	24May	Presentations + Agile		

Class schedule

Sales paradigm

- Sales revolves around two fundamental objectives.

Motivate the buyer

(Make him WANT to buy.)

Overcome objections

Get past his excuses and
reasons for not buying.

Sales paradigm - SSVV

Motivate the STUDENT

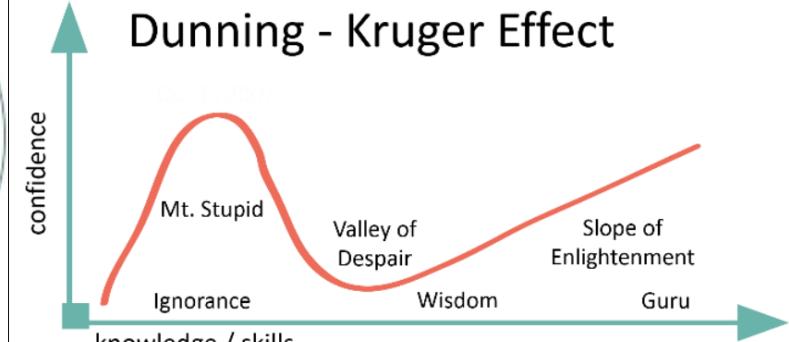
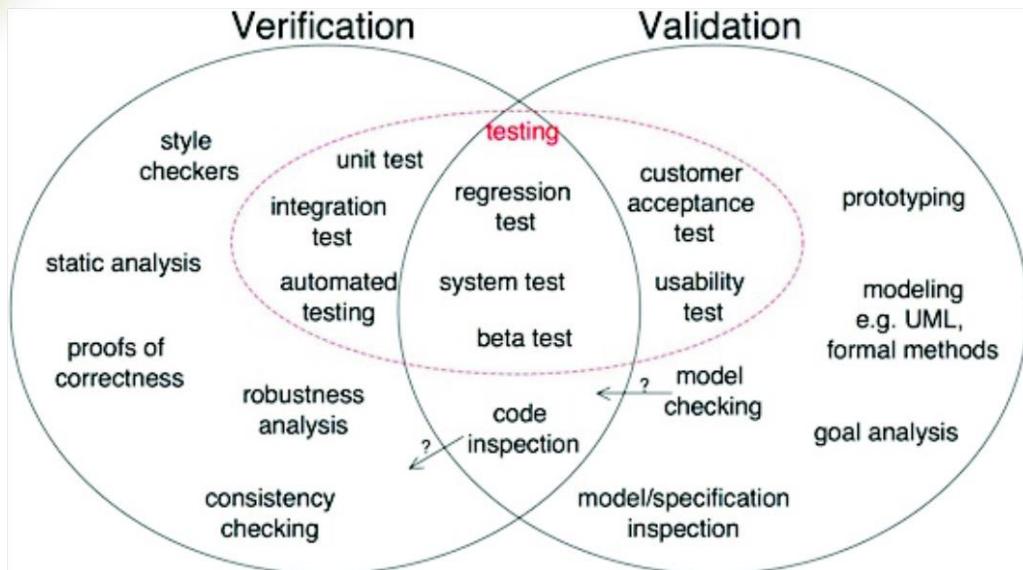
Make him WANT to
learn/participate.

Overcome objections

Get past his excuses and reasons
for not participating in class.

Sales paradigm - SSVV

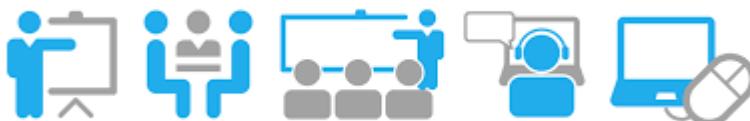
- Motivate the STUDENT - what you will learn!



- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>

Sales paradigm - SSVV

- Overcome STUDENTS's objections



skills



Outline

- Surprises!
- Class schedule
- **Grading**
 - Seminar activity
 - Laboratory activity
- Surprises!

Grading

Grading (Tentative) – will be modified – Lecture 01

- $F = 20\% L + 20\% S + 10\% Q + 50\% E$
 - L=lab; S=Seminar; Q=Quizzes; E=Written;
 - Bonus points! See the homepage of the course!
- Conditions to participate at the final exam
 - There is no restriction regarding the participation at the written examination regarding obtained marks at L, S, Q.
 - Laboratory activity: 3 out of 6 laboratories must be delivered.
 - Attendance lab (5 out of 6) -90%
 - Attendance sem (4 out of 6) – 75%
 - Motivations
 - <http://www.cs.ubbcluj.ro/hotarare-privind-motivarea-absentelor-studentilor-nivel-licenta/>
 - Students will present the documents to motivate absences from the seminar/laboratory within one week from the date of absence.
- L/S/Q work may not be redone in the retake session.
- Students from Previous Years to 2018-2019 - All the above rules apply to students from previous years.
- Conditions to pass/complete the SSVV discipline:
 - $F \geq 5$ final grade.

Seminar

- Attendance: 4 out of 6 required
 - 20% of the final grade
- **Required readings before seminars!**
- Seminar structure
 - **Assignment 1** - 10-minutes – discussion on a given topic (the teacher is an observer!)
 - **Assignment 2** - 60-80 minutes – assignments on a given topic
 - **Assignment 3** – 10 minutes – quiz about required reading and seminar discussions.

Laboratory

- Attendance: 5 out of 6 required
- 20% of the final grade
- Lab structure
 - **First hour of each lab** - current lab discussion, problem assignment and **in-class** problem solving.
 - **The second our of each lab** - delivery of previous lab (exception first laboratory).
- Work in teams – exactly 2 member per team.
- Lab grading for each laboratory (except the last one)
 - **In class** assignments! (3 points)
 - **Take-home** (Homework) assignments! (7 points)
- No more than two lab problems will be delivered in one lab meeting. An extra lab problem is delivered **only if time allowed**.
- Delay in lab submissions –2p from that lab grade.
- Maximum 2 weeks delay in submission of the homework assignment.
- **Remark:**
 - At the end of the semester all laboratory homework assignments must be saved on CD and given to the teacher.
 - Each time you deliver a laboratory - the Deliverables of the in-class and take-home assignments must be uploaded in canvas.

Grading Gamifying Education

	Heroic Quests (quizzes)	Side Quests (Lab projects)	Social Quests (Seminars +Video Presentation)	Epic Quests (Final exam)	XP intervals	Grade
Normal session	300 XP	600XP Each Lab 100 XP (in-class 25XP+ take-home 75XP)	600 XP Each Sem 100XP (in-class 25XP + Quiz 75 XP)	Up to 1500 XP	[1400,1500]	5
					[1501,1800]	6
					[1801,2100]	7
					[2101,2400]	8
					[2401,2700]	9
Retake session	0 XP	0 XP	0 XP	Up to 1500XP	Over 2700	10

- **Final exam – you must come (be present) to final exam in order to compute the grade!**
- Bonus points = 300 XP (1p)
 - 200 XP – activities during lectures
 - 100 XP – Lecture 11 and Lecture 12 – Presentations
- Bonus points = 300 XP (1p)
 - Research paper only if you (will) have 300 XP for Labs
 - Topic by teacher + 2 members/team + deliverables
 - Paper submitted to journal for review

Software System Verification and Validation

Lectures/Lab assignments/Sem topics

- <https://canvas2.cs.ubbcluj.ro/login/canvas>
- You will receive an invitation - scs email
 - At the first login you have to go to the branch "I forgot password "and then the system sends an email to your scs address and then you can add your password and login.

Outline

- Surprises!
- Class schedule
- Grading
 - Seminar activity
 - Laboratory activity
- Surprises!

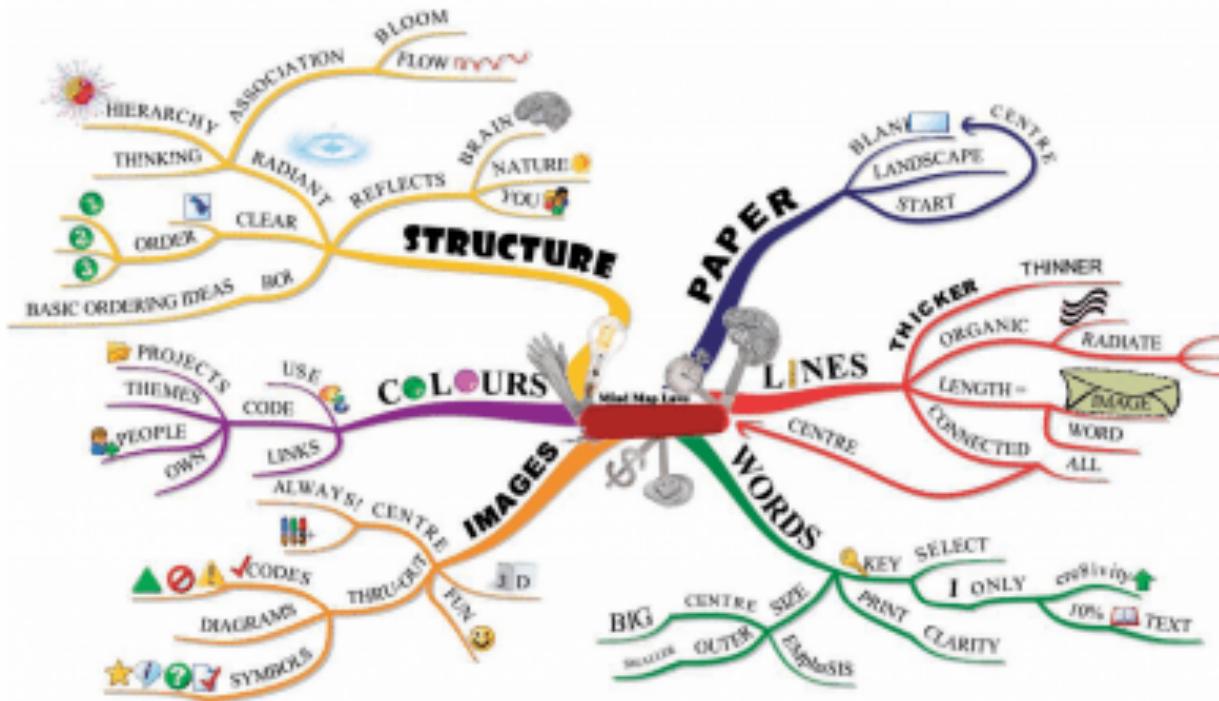
Surprise!



Surprises!

- Experiment: Mindmapping

How to Mind Map®

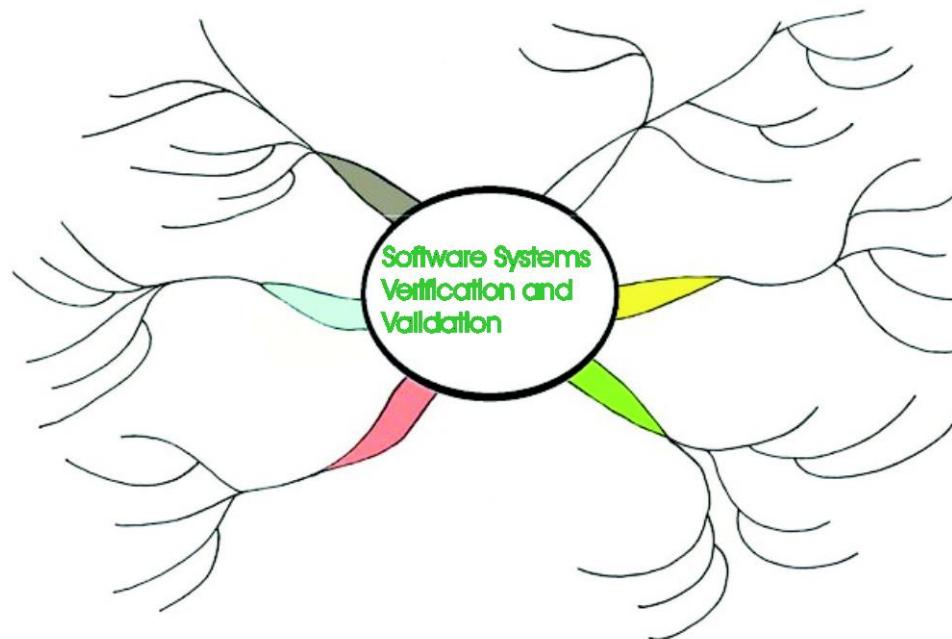


Quick Start Guide

- Set your purpose/goal.
- Start in the CENTRE of blank paper turned sideways.
- Quickly sketch an IMAGE of your focus in the centre.
- Use at least 3 COLOURS, for emphasis, structure, texture, creativity.
- Draw curved lines, radiating from centre (thick to thin) CONNECTING main branches to central image & at each level.
- Use 1 key word or image per line for more power and flexibility in thinking.
- Use images throughout as a picture paints a 1,000 words.

Surprises!

- Experiment: Mindmaping
“Software System Verification and Validation” in the next 5 minutes!



© Paul Foreman <http://www.mindmapinspiration.com>

- Earn: 25 XP in Lecture 12
(if the second mindmap is created)

Questions

- Thank You For Your Attention!

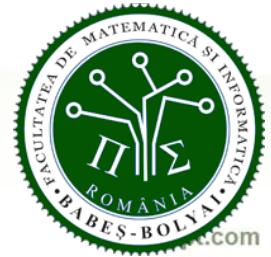
Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan

Babeş-Bolyai University

Cluj-Napoca

2018-2019



Outline

- Verification and Validation
- Software development life cycle Model
 - V-Model
 - Extended V-Model [CB03]
- Quality
 - Quality control vs. Quality Assurance
 - Quality definitions
- What is a bug?
 - First bug
 - Terms for software failures
 - Software error (or bug)
 - When? Why? Cost?
 - Famous Software bugs

Verification and Validation (SEI and NASA)

[NT05],[PY08]

Software Engineering Institute

- **Verification**
 - assures the product is developed according to requirements, specifications and standards.
 - building the product correctly.
 - Are we building the product right?
- **Validation**
 - assures that the product will be usable on the market.
 - building the correct product.
 - Are we building the right product?

NASA - Software Assurance Guidebook and Standard [NAS]

- **Verification and Validation**
 - the process that assures that the software product:
 - will satisfy the requirement (functional and others) = **validation**.
 - every step in the product development is resulting in a correct (sub)product = **verification**.

Verification and Validation - comparison

Verification

- evaluates if the product of a given development phase satisfies the requirements of that phase;
- reviews products to ensure their quality (consistency, completeness, correctness);
- static and dynamic analysis techniques.

Validation

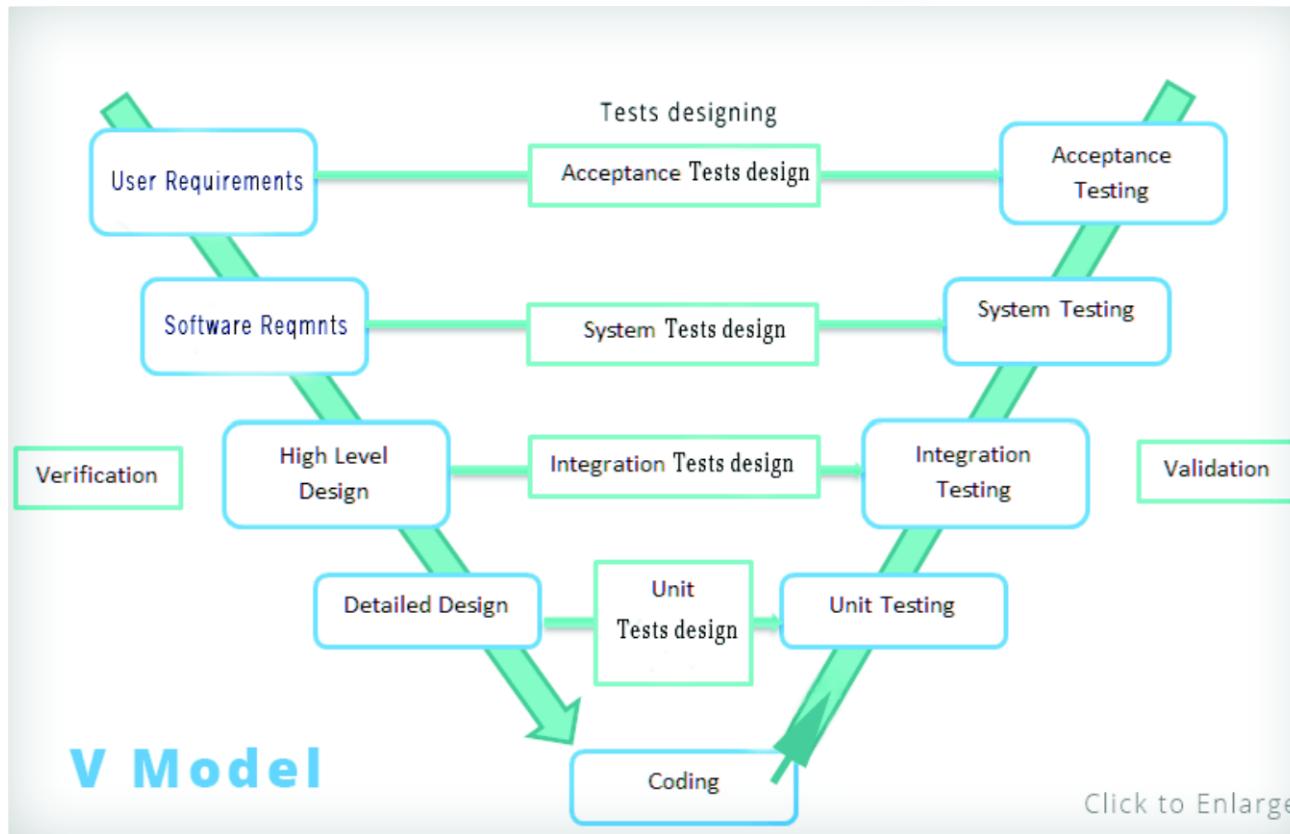
- helps us at confirming that a product meets its intended use.
- is performed toward the end of the system development to determine if the entire system meets the customer's needs and expectations;
- is performed on the entire system by actually running the system in its real environment and using a variety of tests.

Outline

- Verification and Validation
- Software development life cycle Model
 - V-Model
 - Extended V-Model [CB03]
- Quality
 - Quality control vs. Quality Assurance
 - Quality definitions
- What is a bug?
 - First bug
 - Terms for software failures
 - Software error (or bug)
 - When? Why? Cost?
 - Famous Software bugs

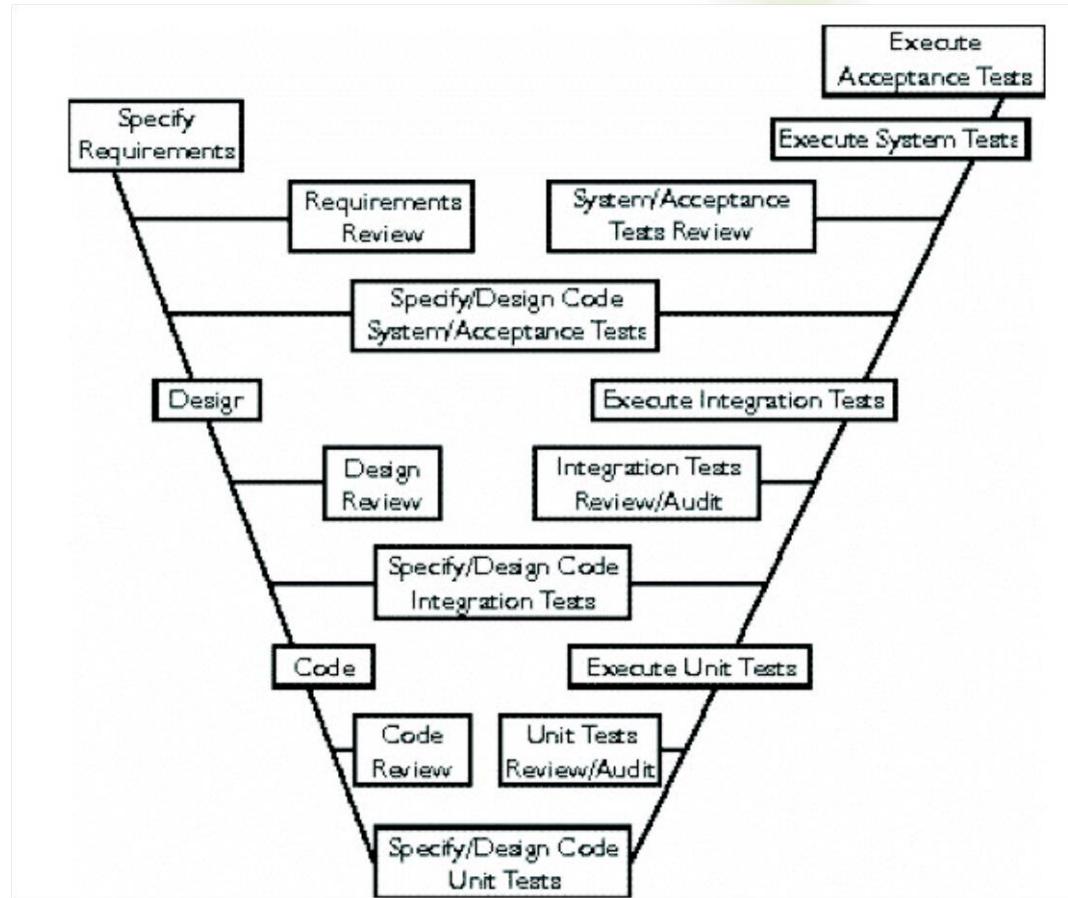
Software development life cycle Model

- V-Model



Software development life cycle Model

- Extended V-Model



Outline

- Verification and Validation
- Software development life cycle Model
 - V-Model
 - Extended V-Model [CB03]
- Quality
 - Quality control vs. Quality Assurance
 - Quality definitions
- What is a bug?
 - First bug
 - Terms for software failures
 - Software error (or bug)
 - When? Why? Cost?
 - Famous Software bugs

Quality control vs. Quality Assurance

- Quality control
 - QC = Quality of products
 - How do you control the quality of the work you have done?
 - Goal: detect problems in the work products
- Quality Assurance
 - QA = Quality of processes
 - How do you assure the quality of the work you are going to do?
 - Goal: Ensure adherence to processes, standards and plans

Quality definitions [BBST]

- “Quality is conformance”
 - “Software quality: Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.” [Pre00]
- “Quality is free.” by Phil Crosby [Cro80]
 - conformance with requirements, i.e. conformance to the user’s actual requirements which may or may not be written down in a specification.
 - Quality - conformance to the needs and not to documents.
- “Quality - fitness for use.” by Joseph Juran [JJ98]
 - satisfiers - anything that makes you like the application
 - dissatisfiers - anything that makes you like the application less.
 - Quality – according to who? (Program manager, Programmer, Tech writer, Tester)
- “Quality - is value to some person.” by Jerry Weinberg [Wei92]
 - quality is subjective - what’s valuable for you may not be so valuable for me.

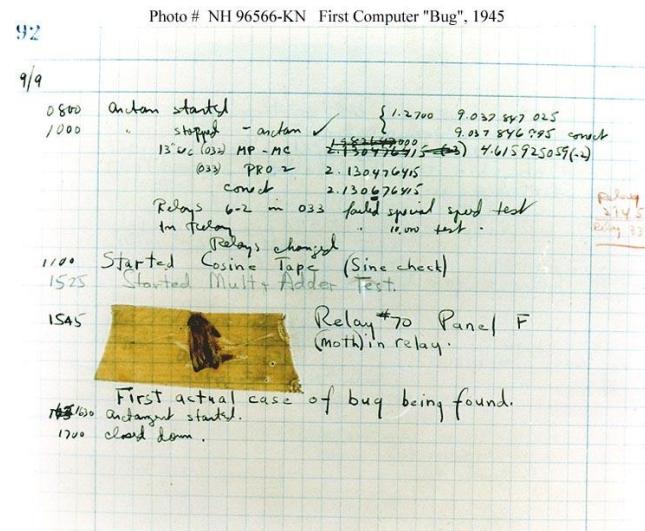


Outline

- Verification and Validation
- Software development life cycle Model
 - V-Model
 - Extended V-Model [CB03]
- Quality
 - Quality control vs. Quality Assurance
 - Quality definitions
- What is a bug?
 - First bug
 - Terms for software failures
 - Software error (or bug)
 - When? Why? Cost?
 - Famous Software bugs

What is a bug?

- First bug
 - Grace Hopper - About first software bug:
 - <https://www.youtube.com/watch?v=lQS0hDqpVLE>
 - 1947 - Harvard University - Mark II



- The first computer bug was born!
 - Well, okay, it died!

Terms for software failures [Pat05]

- **Failure**
 - A failure is said to occur whenever the external behavior of a system does not conform to that prescribed in the system specification.
- **Error**
 - An error is a state of the system. In the absence of any corrective action by the system, an error state could lead to a failure which would not be attributed to any event subsequent to the error.
- **Fault**
 - A fault is the adjudged cause of an error.
- Process of failure manifestation - represented as a behavior chain:
fault → error → failure.

Software error (or bug)

- A bug is an aspect of the product that causes an unnecessary or unreasonable reduction in the quality of the product.
 - design weaknesses, documentation error, usability annoyances

Remark. Some aspects of a product do limit its quality but are not bugs!

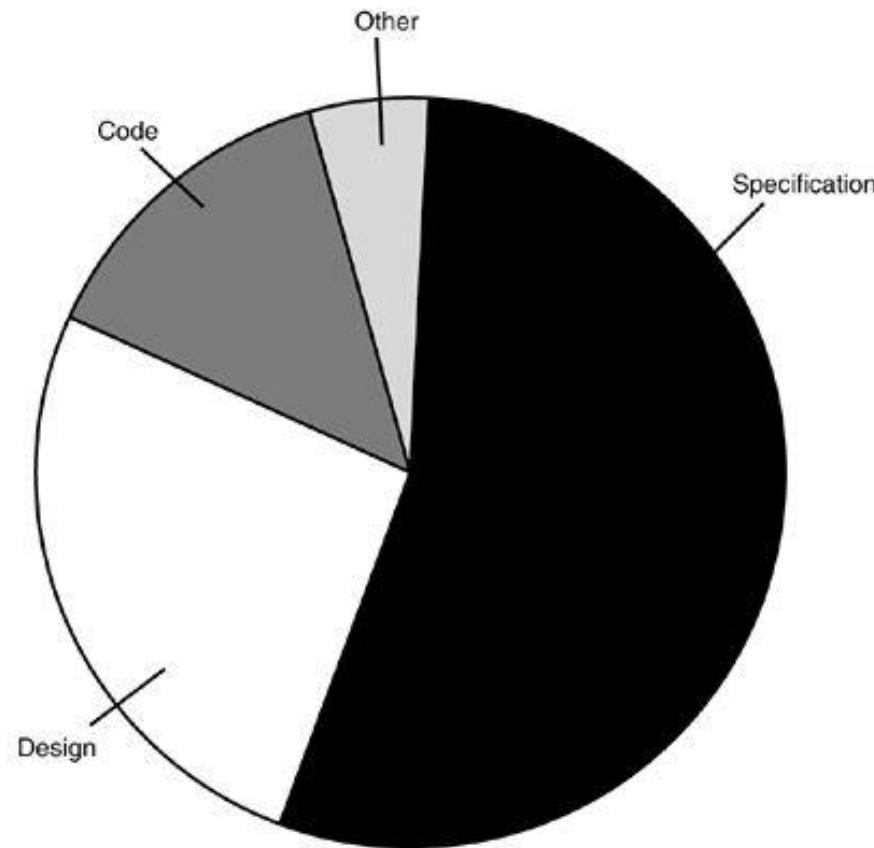


- A bug is anything about the product that threatens its value.
[James Bach, Michal Bolton] [BBST]
 - In this course, all software problems will be called bugs.

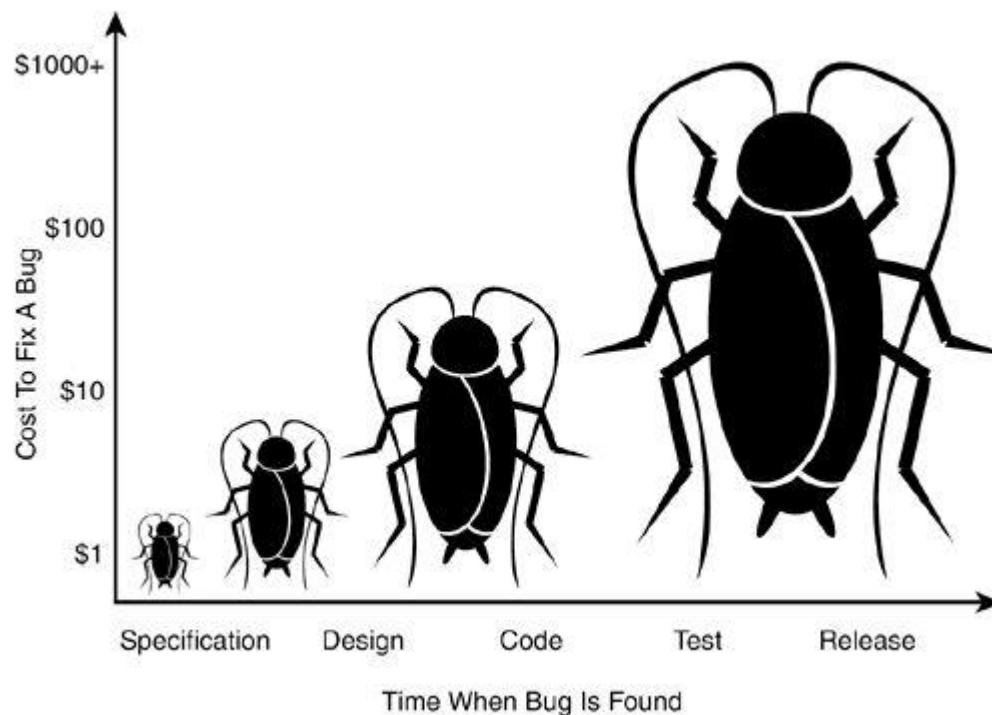
When a software bug occurs?

- A software bug occurs when one or more of the following rules is true [Pat05]:
 - The software doesn't do something that the product specification says it should do.
 - The software does something that the product specification says it shouldn't do.
 - The software does something that the product specification doesn't mention.
 - The software doesn't do something that the product specification doesn't mention but should.
 - The software is difficult to understand, hard to use, slow, or in the software tester's eyes will be viewed by the end user as just plain not right.

Why do bugs occur? [Pat05]



The cost of bugs [Pat05]



How people reacts differently to a single word.

"Bug"



Famous Software bugs

- **Mariner 1 rochet – 1962 - diverted from its intended flight path shortly after launch.**
 - **Cause:** A programmer incorrectly transcribed a handwritten formula into computer code.
 - **Cost:** \$ 18.5 million
- **World War III – almost – 1983 - The Soviet early warning system falsely indicated the United States had launched five ballistic missiles.**
 - **Cause:** A bug in the Soviet software failed to filter out false missile detections caused by sunlight reflecting off cloud-tops.
 - **Cost:** Nearly all of humanity
- **Therac-25 radiation therapy machine – 1985 - Canada Therac-25 radiation therapy machine malfunctioned and delivered lethal radiation doses to patients.**
 - **Cause:** Because of a subtle bug called a race condition, a technician could accidentally configure Therac-25 so the electron beam would fire in high-power mode without the proper patient shielding.
 - **Cost:** Three people dead, three people critically injured
- **Pentium Fails Long Division – 1993 - Intel highly-promoted Pentium chip occasionally made mistakes when dividing floating-point numbers within a specific range.**
 - **Cause:** The divider in the Pentium floating point unit had a flawed division table, missing about five of a thousand entries and resulting in these rounding errors.
 - **Cost:** \$475 million, corporate credibility.
- **Disney's Lion King – 1995 - The Disney company released its first multimedia cd-rom game for children, The Lion King Animated Storybook. Several parents couldn't get the software to work.**
 - **Cause:** Disney failed to test the software on a broad representation of the many different PC models available on the market.
 - **Cost:** cd-rom replacements, corporate credibility.
- **Mars Climate Crasher – 1998 - After a 286-day journey from Earth, the Mars Climate Orbiter fired its engines to push into orbit around Mars. The engines fired, but the spacecraft fell too far into the planet atmosphere, likely causing it to crash on Mars.**
 - **Cause:** The software that controlled the Orbiter thrusters used imperial units (pounds of force), rather than metric units (Newtons) as specified by NASA.
 - **Cost:** \$125 million.
- **Cancer Treatment -2000 - Radiation therapy software by Multidata Systems International miscalculated the proper dosage, exposing patients to harmful and in some cases fatal levels of radiation.**
 - **Cause:** The software calculated radiation dosage based on the order in which data was entered, sometimes delivering a double dose of radiation.
 - **Cost:** Eight people dead, 20 critically injured.

Next Lecture (Still today!)

- Inspection

Questions

- Thank You For Your Attention!

References

- [CB03] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- [Cro80] Philip B. Crosby. *Quality Is Free*. Signet Shakespeare, 1980.
- [JJ98] A. Blanton Godfrey Joseph Juran. *JURANS QUALITY HANDBOOK*. McGraw-Hill, 1998.
- [NAS] Nasa - standard for software assurance.
<http://www.hq.nasa.gov/office/codeq/doctree/87398.htm>.
- [NT05] K. Naik and P. Tripathy. *Software Testing and Quality Assurance*. Wiley Publishing, 2005.
- [Pat05] R. Patton. *Software Testing*. Sams Publishing, 2005.
- [Pre00] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., 2000.
- [PY08] M. Pezzand and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2008.
- [Wei92] Gerald Weinberg. *Quality Software Management Vol. 1: Systems Thinking*. DORSET HOUSE PUBLISHING, 1992.
- [BBST] BBST Testing course, <http://testingeducation.org/BBST/>
<http://testingeducation.org/BBST/bugadvocacy/BugAdvocacy2008.pdf>,
slides 26-32 for quality definitions, Lecture 1 video, min 27-35)

Software Systems Verification and Validation

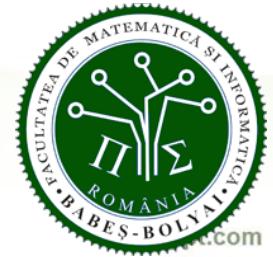
Assoc. Prof. Andreea Vescan

Babes-Bolyai University

Cluj-Napoca

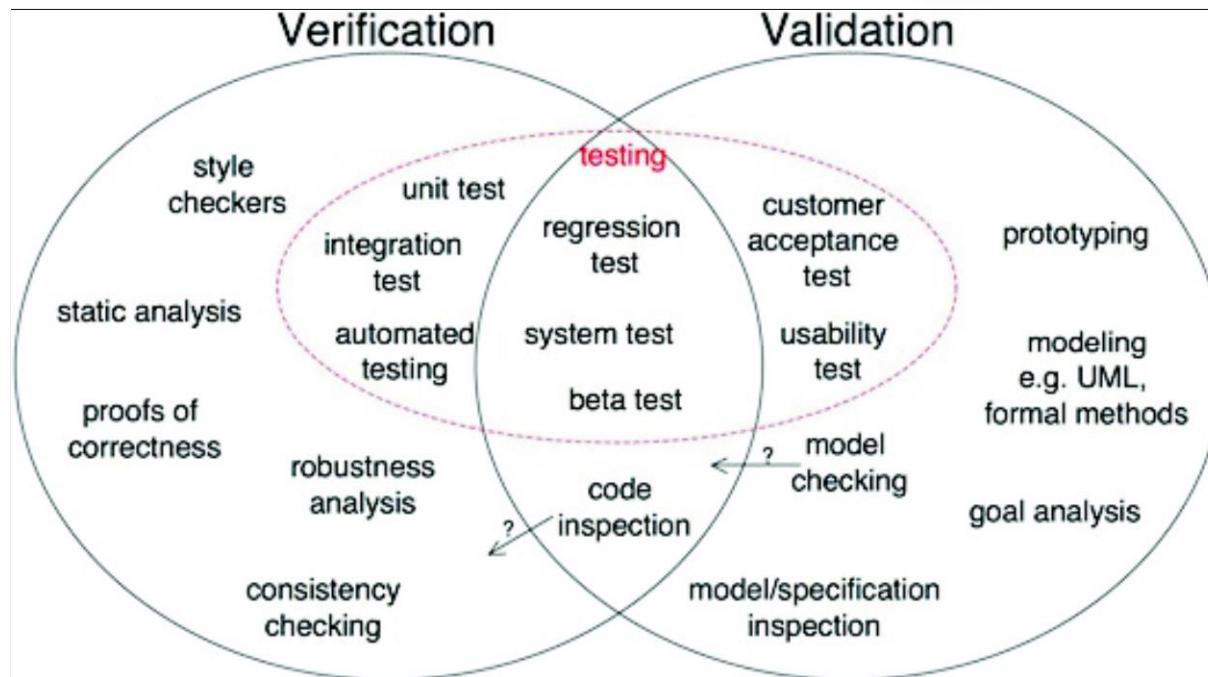
2018-2019

Lecture 1: Inspection



Sales paradigm - SSVV

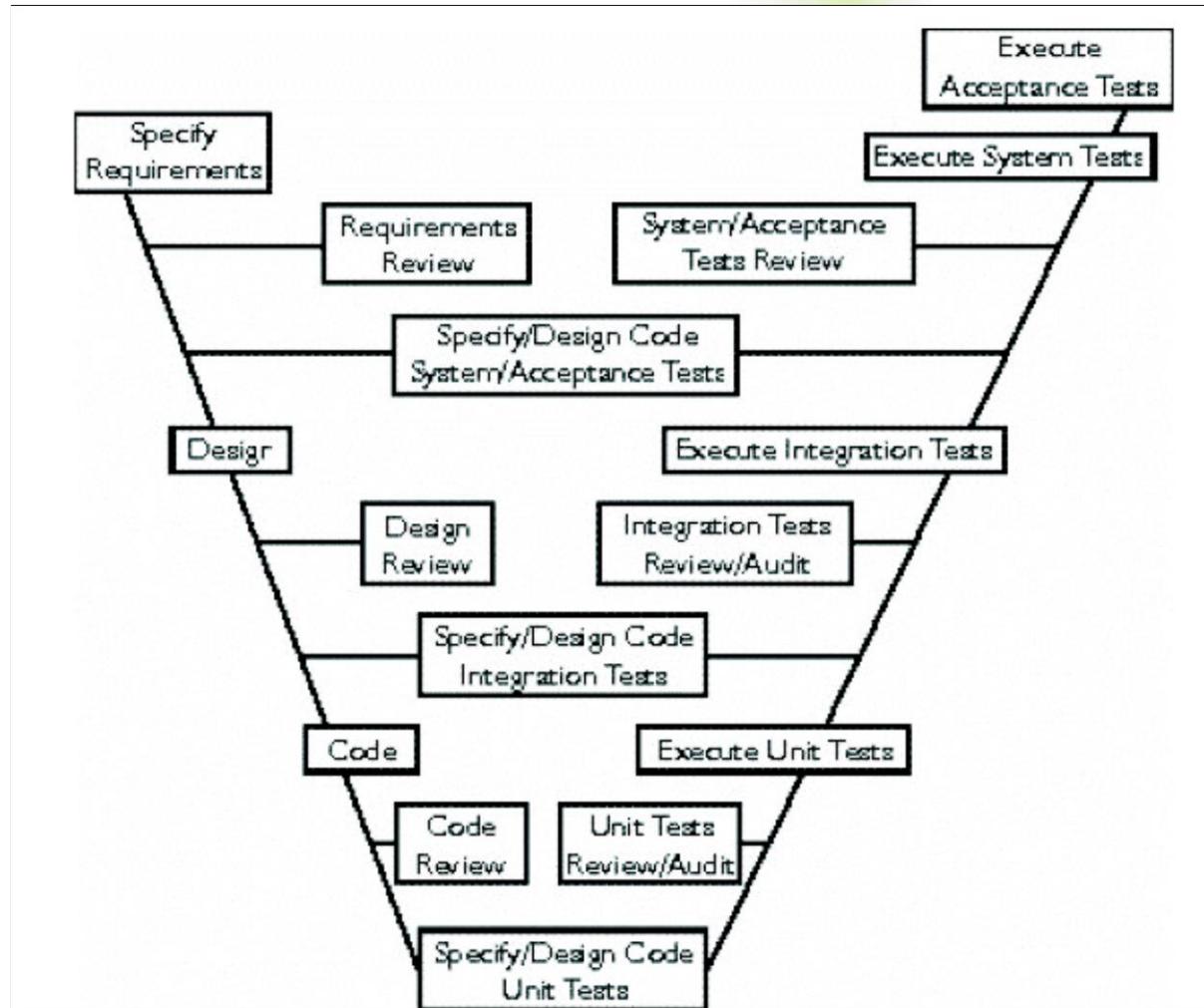
- Motivate the STUDENT - what you will learn!



- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>

Software development life cycle Model

- Extended V-Model

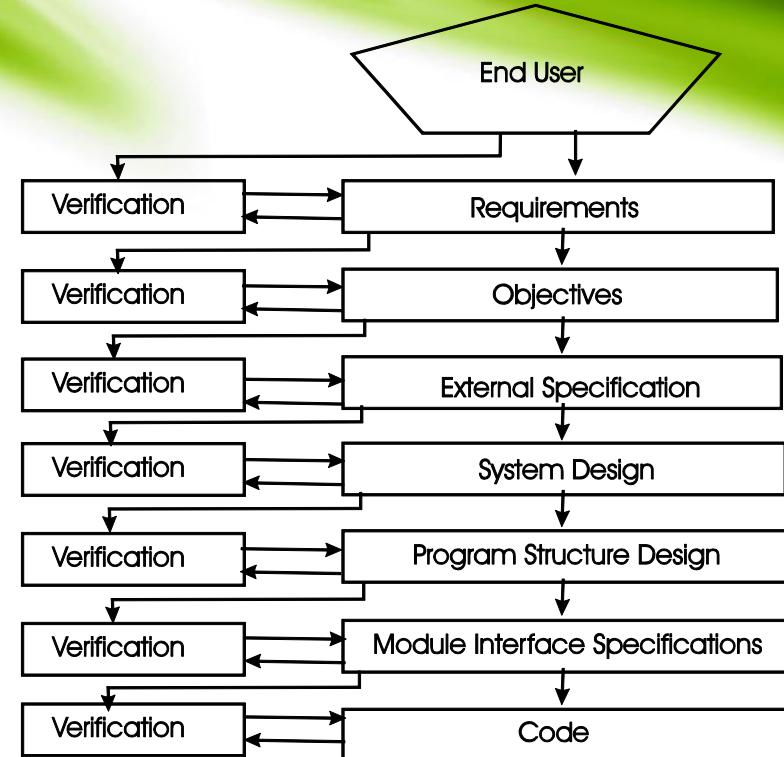


Outline

- Human testing
- Human testing methods
 - Inspections
 - Walkthroughs
 - Pair-programming

Human testing

- Prevent errors
 - introduction of a verification step at the end of each process.



Outline

- Human testing
- Human testing methods
 - Inspections
 - Walkthroughs
 - Pair-programming

Human testing methods

[Mye04] (chapter 3), [PY08] (chapter 18), [Fre10] (chapter 4)

- Is it useful? they contribute to productivity and reliability:
 - The earlier errors are found, the lower the cost of correcting the errors.
 - Psychological change of programmers when computer-based testing commences.
- Human testing methods are:
 - Inspections
 - Walkthroughs
 - Pair-programming
- Objective - to find errors but not to find solutions to the errors.
- Advantage - when an error is found it is usually located.
 - **Finds from 30% to 70% of the logic-design/coding errors in programs (?)**.
- Inspection and computer-based testing are complementary.

WE ARE FINDING A DEFECT IN REVIEW 9 TIMES FASTER THAN IN TESTING.

WE ARE SOLVING A DEFECT FOUND IN REVIEW 5 TIMES FASTER THAN A DEFECT FOUND IN TESTING.



Outline

- Human testing
- Human testing methods
 - Inspections
 - Walkthroughs
 - Pair-programming

Inspection

- **Inspection** - process of trying to find defects in development documents during various phases of the software development process.
- Fagan Inspection team ([4 members])
 - Moderator - duties
 - Distributing materials for, and scheduling the inspection session.
 - Leading the session
 - Ensuring that the errors are subsequently corrected.
 - Author of the product (analyst, designer, programmer)
 - Secretary
 - Reader
- Checklists
- Time - 90-120 minutes

Inspection activities

- **Planning**
 - the moderator selects the team members
 - distribution of the materials to the members; task assignment
- **Presentation/Overview - not compulsory**
 - used to present details to the members of the inspection team
- **Individual preparation**
 - reading and understanding the received documentation
- **Inspection meeting**
 - critical observations of each individual inspectors - discussed
 - conclusions of the inspection - documented
- **Rework**
 - the author makes the required changes and correct the errors
- **Follow-up**
 - to verify if the modification did eliminate the errors
 - may be only between the author and the moderator

Inspection checklists

- Inspection scope - to find errors
- Depending on the analyzed document - special kind of errors
- **Specification Document**
 - Does the specification conforms to the user's needs?
 - Are there ambiguities in the specification?
 - Do the input/output date are clearly stated? What about input/output conditions?
 - Are there requirements that are not present in the specification?
 - Are there performance conditions? What precise computation conditions?
- **Analysis Document**
 - Does the design conforms to the specification?
 - Are all the functionalities from the specification specified?
 - Is there an analysis documentation about the made decisions?

Inspection checklists (cont.)

- **Code**
 - Does the code conforms to the design?
 - Are all the methods are called?
 - Are all the variables initialized?
 - Problems with: infinite cycles, out of bound indexes, improper allocation of memory.
- **Test Document**
 - The test cases are well documented?
 - The test cases are well chosen?
 - Are the test data sufficient to coverage criterion?
 - For the integration testing, the order of integration is clear?
 - At regression testing is the testing continued?

Inspection advantages [CB03]

- Early error discovery
- Reduce product development time and cost
- Group method
- Mean to education
- The source of error is known (locating defect)
- Eliminates the debugging stress if few day remains until product release
- Inspection – more efficient than testing [CB03]
 - detecting, locating, repairing defect
 - a two pass approach (individuals first and by the group)
 - checklist – calls attention to specific defect prone areas

Outline

- Human testing
- Human testing methods
 - Inspections
 - Walkthroughs
 - Pair-programming

Walkthrough

- **Walkthrough** [You79], [CB03] - process of trying to find defects in development documents during various phases of the software development process.
- Similar to Inspection
- **Team members** ([3-5] members)
 - Moderator ([CB03]- moderator = the producer of the reviewed material
 - → a larger amount of material can be processed by the group)
 - Secretary
 - Tester
- **Procedures** are slightly different
 - Planning
 - Meeting - the participants “play computer” (no checklist)
 - No Individual preparation [CB03]
 - Rework [You79]
 - Follow-up
- Different error-detection technique
- Time - 90-120 minutes

Outline

- Human testing
- Human testing methods
 - Inspections
 - Walkthroughs
 - Pair-programming

Pair-Programming

- Variation of program inspection.
- Merges coding and inspection activities.
- The inspection activities
 - are not driven by checklists
 - are based on shared programming practice and style
- Programmers frequently alternate roles
- Is carried out in normal work days, without excessive overtime and without severe schedule pressure.
- No mediator, so responsibility for open and nondefensive discussion of decisions/alternatives falls to the programmers.

Outline

- Human testing
- Human testing methods
 - Inspections
 - Walkthroughs
 - Pair-programming
- Next Lecture (tentative)

Next Lecture (tentative)

- Testing. Test planning.
- Test case design - Black-box testing
- Testing Management Tool - TestLink
- Continuous integration - Jenkins

Questions

- Thank You For Your Attention!

References

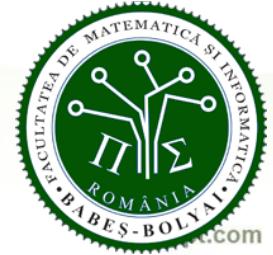
- [PY08] M. Pezzand and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2008.
- [Mye04] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004
- [You79] E. Yourdon, Structured Walkthroughs, Prentice-Hall, Englewood Cliffs, NJ, 1979
- [CB03] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010

Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan
Babeş-Bolyai University
Cluj-Napoca
2018-2019

Lecture 2b: BBT

Software Systems Verification and Validation



Outline

- Testing – fundamental questions
- Black-box testing (domain testing)
 - Equivalence partitioning (EP)
 - Boundary-value analysis (BVA)
 - Advantages/Disadvantages
- Example - black-box testing
 - Example – EP+BVA
- Test management tool – Testlink
- Maven
- Continuous integration tool - Jenkins

Domain testing

- The problem you have to solve
 - You cannot afford to run every possible test.
 - You need a method for choosing a few powerful tests that will represent the rest.
- Domain testing
 - Provides a **sampling strategy**
 - It provides **heuristics** for choosing a small number of tests that are powerful enough to represent the larger domain.
 - Equivalence
 - Boundaries
- Input versus output variables
 - Domain definitions
 - Specification:
 - Data (X) + preconditions
 - Result (Z)+ postcondition
- Primary dimension versus secondary dimension [BBST]
 - Primary dimension - reflects the reason you are entering data into the field
 - Secondary dimension - reflects the other ways the input can vary

Domain testing (Black-box testing)

- Equivalence partitioning
- Boundary-value analysis

Equivalence partitioning

- Equivalence class (EC) - definition [Mye04]
 - a partition of input domain of a program.
- Equivalence partitioning
 - to partition the input domain of a program into a finite number of equivalence classes such that you can reasonably assume that a test of a representative value of each class is equivalent to a test of any other value.

Test-case design by equivalence partitioning - steps

- Identifying the equivalence classes (EC)
 - Valid equivalence classes.
 - Invalid equivalence classes.
 - Default, empty, blank, null, zero, none.
 - Invalid, wrong, incorrect, garbage date.
- Defining the test cases
 - assign a unique number to each equivalence class;
 - Until all valid/invalid equivalence classes have been covered by (incorporated into) test cases:
 - write a new test case covering as many of the uncovered valid equivalence classes as possible;
 - write a test case that covers one, and only one, of the uncovered invalid equivalence classes.

Test-case design by equivalence partitioning - guidelines

- An input condition specifies a range of values [a,b].
 → 1 valid EC, 2 invalid EC
- An input condition specifies the number of values “1 to 3 possibilities”.
 → 1 valid EC and 2 invalid EC
- An input condition specifies a set of input values.
 → 1 valid EC for each element in the set, 1 invalid EC
- An input condition specifies a must be situation.
 → 1 valid EC, 1 invalid EC
- If there is any reason to believe that the program does not handle elements in an equivalence class identically, split the equivalence class into smaller equivalence classes.

Boundary-value analysis

- Boundary-value analysis - definition [Mye04]
 - focuses on the boundary areas of a programs input domain
- Boundary conditions
 - Situations directly on, above, and beneath the edges of input EC and output EC.
 - One or more elements should be selected such that each edge of the EC is the subject of a test.
 - BVA explores situations on and around the edges of the EP.

Test-case design by boundary-value analysis -guidelines

- An input condition specifies a range of values [a,b].
 → the ends of the range, situations just beyond the ends;
- An input condition specifies the number of values “1 to 3 possibilities”.
 → the minimum and maximum number of values, one beneath and beyond these values;
- An input condition specifies an ordered set of input values.
 → the first, the last elements of the set;
- The above rules are applied to the output conditions.

Domain testing (Black-box testing)

Advantages

- No knowledge of implementation.
- Tester independent of programmer.
- User's point of view.
- Ambiguities in spec.
- After specifications is completed.

Disadvantages

- A small number of inputs.
- No clear spec.
- Hard to design.
- Unnecessary repetition of test.
- Many program paths untested.
- Specific segments of code?

Surprise!





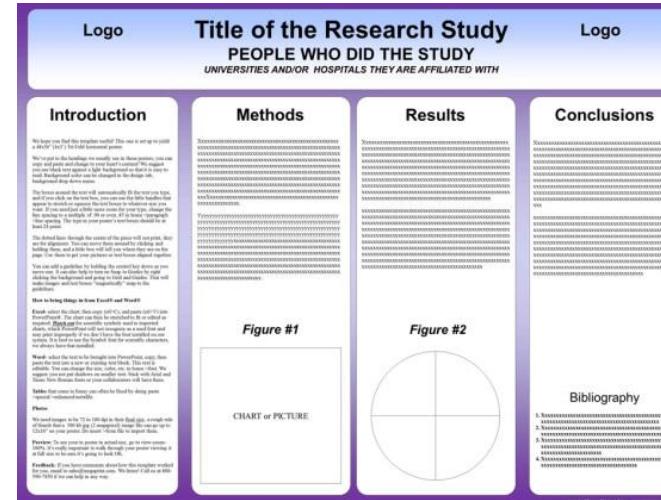
Domain testing with Risk-based testing

Information to study

- <http://www.testingeducation.org/BBST/testdesign/>
 - Lecture 5: Domain Testing
 - part D (6 mins, 50 secs)
- A new table risk/equivalence
- <http://www.testingeducation.org/BBST/testdesign/testdesign5d.mp4>

Bonus points

- Maximum 25 XP



Information to provide

- Create an example with the new table risk/equivalence
- Poster-based presentation
 - Present the table in Lecture 3 for your colleagues in 5 minutes.
 - Bring it in printed (or written by hand) form for the teacher.
- A3 page format

Example

- **Problem statement:** Compute the number of participants with the maximum score (0 to 100 points possible) at a competition.
- Applied:
 - EP
 - BVA
- See example files on SSVV lecture's homepage

Maven

- goal - to allow a developer to comprehend the complete state of a development effort in the shortest period of time
- <https://maven.apache.org/what-is-maven.html>
- Maven
 - The Failsafe Plugin is designed to run integration tests.
 - Maven 2 Integration

Testlink

- Test management tool
 - Testlink. (Release 1.9.8)
- <https://www.scs.ubbcluj.ro/testlink>

Jenkins

- Continuous integration tool
- <https://scs.ubbcluj.ro:9090/>

Lecture 2: Required reading (before Seminar 2) + Quiz

Laboratory 2 - discussion

- Testing – Black-box testing/Domain testing (EC+BVA)
 - In class assignments
 - Homework assignments

References

- [PY08] M. Pezzand and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2008.
- [Mye04] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004
- [You79] E. Yourdon, Structured Walkthroughs, Prentice-Hall, Englewood Cliffs, NJ, 1979
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010
- [BBST] BBST Testing course, <http://testingeducation.org/BBST/>
 - **Test design,**
 - **Lecture 5: Domain testing**
- Tutorials - SSVV lecture's homepage.
 - www.cs.ubbcluj.ro/~avescan

Questions

- Thank You For Your Attention!

Next Lecture

- White-box testing

Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan
Babeş-Bolyai University
Cluj-Napoca
2018-2019

Lecture 2a: Testing



Feedforward!



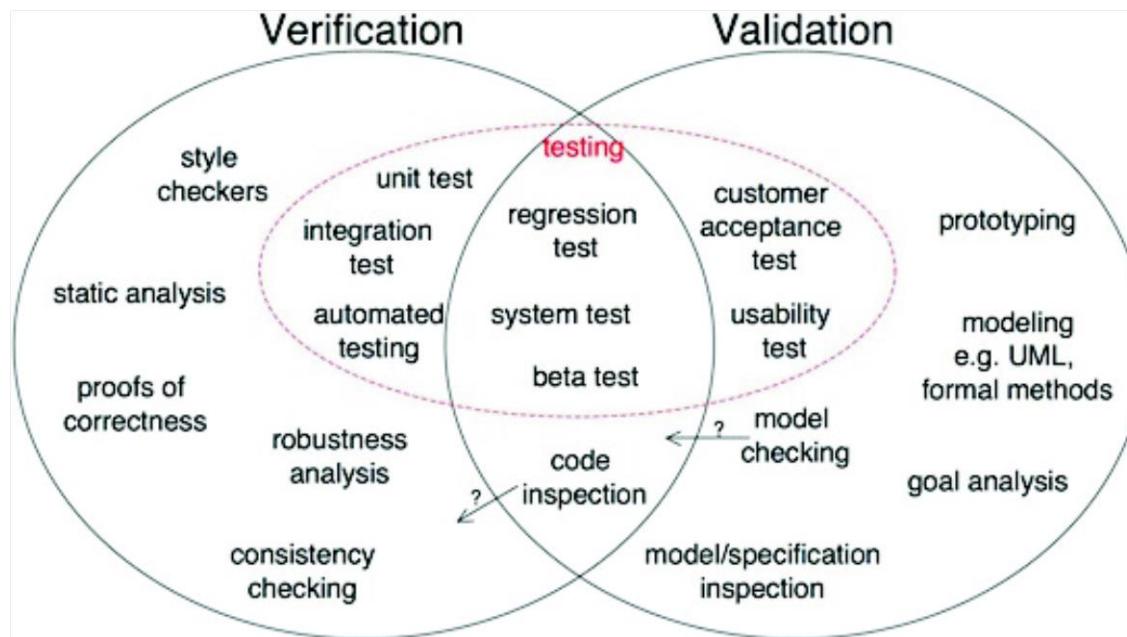
Your responses

- **homework only**
- **in class - to be finished home, less in class**
- **individual work (or/and)**
- **team work (enjoy it, as in real life)**

1. Plan to Attend – how many?
 - 1.1. Lectures (12)
 - 1.2. Laboratories (6)
 - 1.3. Seminar (6)
2. Expect/want to learn/do
 - 2.1. Lectures
 - 2.2. Laboratories
 - 2.2. Seminars
3. Lectures/Laboratories/Seminars – how
 - 3.1. given/individual/team study
 - 3.2. assignment – in class/ home
4. Grading
 - 3.1. Normal/retake session
 - 3.2. Activities to be evaluated
5. Other remarks

Sales paradigm - SSVV

- Motivate the STUDENT - what you will learn!



Outline

- Testing – fundamental questions (why, how, when, enough?)
- Software testing
 - Definitions (why)
- Testing strategy (how)
- Test planning (enough)
 - Test cases
 - Testing activities
 - Bug report
- Software testing (Required reading before Seminar 2)
 - Testing principles
 - Testing axioms
- When do we test?

Testing - fundamental questions

- Why do we test?
- How do we organize the process of testing?
- When we have tested enough?



What skills do we need to have to test?

Surprise!

Play ... detective!



Playing detective (1)

Context 1.

A man went into a party and drank some of the punch. He then left early. Everyone at the party who drunk the punch subsequently died of poisoning. Why did the man not die?

Context 2.

A detective who was mere days from cracking an international smuggling ring has suddenly gone missing. While inspecting his last-known location, you find a note:

710

57735

34

5508

51

7718

Currently there are 3 suspects: Bill, John, and Todd. Can you break the detective's code and find the criminal's name?

Context 3.

There is a man found dead in a circular mansion. The detective interviews the cook, maid, and babysitter. The cook said he couldn't have done it because he was preparing the meal. The maid said she couldn't have done it because she was dusting the corners. The babysitter said she couldn't because she was playing with the children. Who was lying?

Playing detective (2)

Can you state (some) similarities
between a **detective** and a **tester**?

Subject of Investigation

Detectives

Crime

- the commission of an act that is forbidden
- activity that is against the law
- harmful act against the public



Testers

Computer program (CP)

- CP=Set of instructions for a computer
- CP=A communication among several humans and computers [BBST]
- Defining a house
 - A set of construction materials put together according to a house-design pattern
 - Is built for people to live in

Playing detective (3)

Can you state (some) similarities
between a **detective** and a **tester**?

Used Tools

Detectives

- writing statements by witnesses/suspects
- writing up reports for superiors and team members
- recording equipment
- taking charge of all the information related to the case (keep track using folders)
- standard evidence kit (gloves, envelopes and/or plastic bags, tape measures, flashlights and scissors)

Testers

- Test plan
- Test management tools
- Bug reports
- Software tools (developer, tester)
- Logs
- Screen capture tools
- Oracles
- Testing techniques

Playing detective (4)

Can you state (some) similarities
between a **detective** and a **tester**?

Job duties

Detectives

- Researching and Analyzing Crimes
- Identifying and Interrogating Suspects
- Testifying in Court

Testers

- Researching for Information
- Interrogating
- Bug advocacy
- **Remark:** They don't even know what crime has been committed yet.



Why testers are hunters and developers are farmers

- <https://qubiz.com/blog/testers-are-hunters-and-developers-are-farmers/>
- Tester's skills

IT firm – Altom
Lecture invitation



Why do we test?

Software testing - DEFINITION

- **Software testing [BBST]** - is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test
- **False/incomplete definitions**
 - Testing is the process of demonstrating that errors are not present. “Testing can only reveal the presence of errors, never their absence.” [Dij75]
 - The purpose of testing is to show that a program performs its intended functions correctly.
 - Testing is the process of establishing confidence that a program does what it is supposed to do.
- **Definition**
 - Testing is the process of executing a program with the intent of finding errors. [Mye04]
 - Human beings tend to be highly goal-oriented.
 - Testing is a destructive process.
 - Successful and unsuccessful.
 - Error: the program does not do what it is supposed to do, BUT also if the program does what it is not supposed to do.



Software testing

Why do we test?

- We test a product to learn about its quality. [BBST]
- Everyone tests in context [BBST]
 - Harsh constraints
 - Complete testing is impossible
 - Finite project schedules and budget
 - Limited skills of the test group
 - Do testing – before/during/after – release
 - Improvement of product – might/might not be an objective of testing
 - Test in behalf of stakeholders
 - Project manager/customer/programmer/attorney
- A search for information [BBST]
 - Question: What kind of information are we most concerned about learning from this testing?
 - Answer: Our information objectives.

TESTING IS ALWAYS A SEARCH FOR INFORMATION

- Find important bugs
- Assess the quality of the product
- Help managers assess the progress of the project
- Help managers make release decisions
- Block premature product releases
- Help predict and control product support costs
- Check interoperability with other products
- Find safe scenarios for use of the product
- Assess conformance to specifications
- Certify the product meets a particular standard
- Ensure the testing process meets accountability standards
- Minimize the risk of safety-related lawsuits
- Help clients improve product quality & testability
- Help clients improve their processes
- Evaluate the product for a third party

Different objectives require different testing tools and strategies and will yield different tests, test documentation and test results.

Testing ... in context

Context 1.

The teacher asked a group of students (development team) to design and implement a Java application for the given statement problem. The teacher just want to verify the students 's design and implementation skills.

You are working as a tester and you are assigned to the development team early in the project. The project manager (i.e., the teacher) wants you to test along with the programmers as they write code. She wants you to help the programmers deliver clean code and she wants to help her and her staff identify, understand and control the implementation.

Context 2.

The teacher asked a group of students (development team) to design and implement a Java application for the given statement problem. The teacher just want to verify the students 's design and implementation skills.

You are working as a tester and you are assigned to the development team when all the required stated functionalities were implemented and individually tested. The project manager (i.e., the teacher) wants you to test the integration of the implemented functionalities.

Context 3.

The teacher asked a group of students (development team) to design and implement a Java application for the given statement problem. The resulted application will be used by the doctors in their university and by other family doctors.

You are working as a tester and you are assigned to the development team close to the project's release date. The doctors expected it to work and they needed to work well. The project manager (i.e., the teacher) expects you to test the product in ways that will help her understand whether the product is ready to release or not.



How do we test?

Strategy/Approach/Technique

- **Strategy definition** (<https://en.oxforddictionaries.com/definition/strategy>)
 - A plan of action designed to achieve a long-term or overall aim.
- **Approach definition** (<http://www.dictionary.com/browse/approach>)
 - the method used or steps taken in setting about a task, problem
- **Technique definition** (<http://www.dictionary.com/browse/approach>)
 - technical skill; ability to apply procedures or methods so as to effect a desired result
- **Testing strategy** = guiding framework for deciding what tests (what test techniques) are best suited to your product.
- **Approach** = a way of thinking about ...
- **Technique** = methods used to ...

Testing strategies [BBST]

- **Testing strategy is:**
 - The guiding framework for deciding what tests (what test techniques) are best suited to your product.
- **Context and information objectives** are (or should be) the drivers of any **testing strategy**.
 - Examples of context factors that drive and constrain testing
 - Who are the stakeholders with influence?
 - What are the goals and quality criteria for the project?
 - What skills and resources (time, money, tools, data, etc) are available?
 - Potential consequences of potential failures?
 - How could it fail?
 - Common information objectives
 - Find important bugs
 - Assess the quality of the product
 - Help managers make release decisions
 - Block premature product releases
 - Assess conformance to the specification
 - Find safe scenarios for the use of the product
 - Evaluate the product for a third party

Strategy/Approach/Technique (revisited)

- Context and information objectives are (or should be) the drivers of any testing strategy.

Selecting the Testing techniques ?

Techniques differ in core.

Attributes of “good” tests



- *Power*
- *Valid*
- *Value*
- *Credible*
- *Representative*
- *Non-redundant*
- *Motivating*
- *Reusable*
- *Performable*
- *Maintainable*
- *Information value*
- *Coverage*
- *Easy to evaluate*
- *Support troubleshooting*
- *Appropriately complex*
- *Accountable*
- *Affordable*
- *Opportunity cost*

Strategy/Approach/Technique (revisited)

- **Context and information objectives** are (or should be) the drivers of any **testing strategy**.

Selecting the Testing techniques ?

<http://www.testingeducation.org/BBST/testdesign/>

- <http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf>
- **Slides 67 to 72**
- COVERAGE-BASED TECHNIQUES FOCUS ON WHAT GETS TESTED
- TESTER-BASED TECHNIQUES FOCUS ON WHO DOES THE TESTING
- RISK-BASED TECHNIQUES FOCUS ON POTENTIAL PROBLEMS
- ACTIVITY-BASED TECHNIQUES FOCUS ON HOW YOU DO THE TESTING
- EVALUATION-BASED TECHNIQUES FOCUS ON YOUR ORACLE
- DESIRED-RESULT TECHNIQUES FOCUS ON A SPECIFIC DECISION OR DOCUMENT
- THERE ARE ALSO GLASS-BOX TECHNIQUES



How do we organize the process of testing?

Test planning

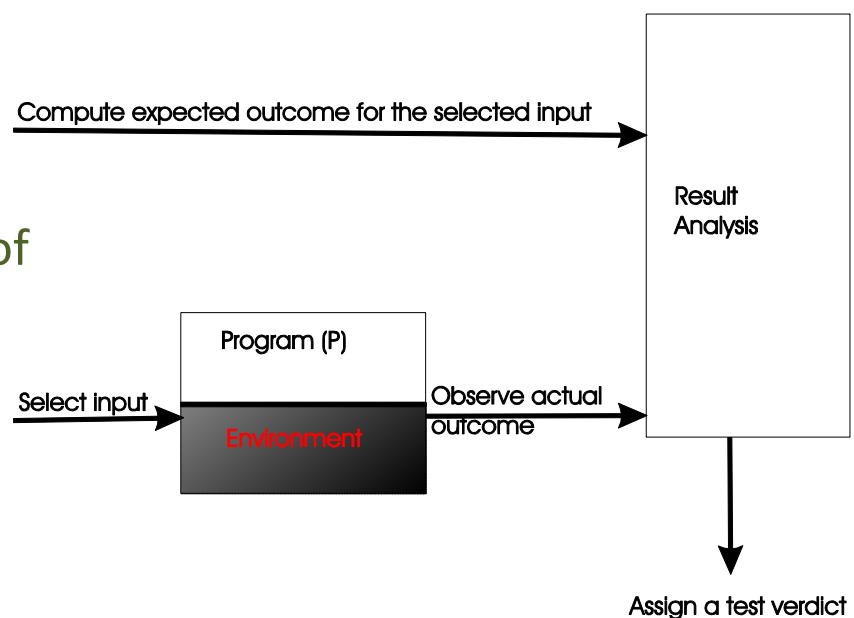
- Test plan?
 - testers communicate what they intend to do.
 - takes the form of a written document, not only the creation of the document, but also planning the testing tasks.
 - The ultimate goal of test planning process is communicating the software test team's intend, expectations, understanding.
- Test planning topics
 - A test plan template? Important topics?
 - The test team's high-level expectations
 - People, places and things, Definitions
 - Inter-group responsibilities
 - What will and won't be tested
 - Test phases, test strategy, Resource requirements
 - Tester assignments
 - Test schedule, Test cases
 - Bug reporting
 - Reasons for planning test cases
 - Organization
 - Repeatability
 - Tracking
 - Proof of testing
 - VerVal matrix

Test case

- Test case $\langle i, r \rangle$
 - i domain D
 - r domain R.
 - for input i the expected result is r .
- Test case: “A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.” [IEE90]
- “Good” test case attributes:
 - High probability of finding an error.
 - Is not redundant.
 - “Best of breed”.
 - Neither too simple nor too complex.

Testing activities [NT05]

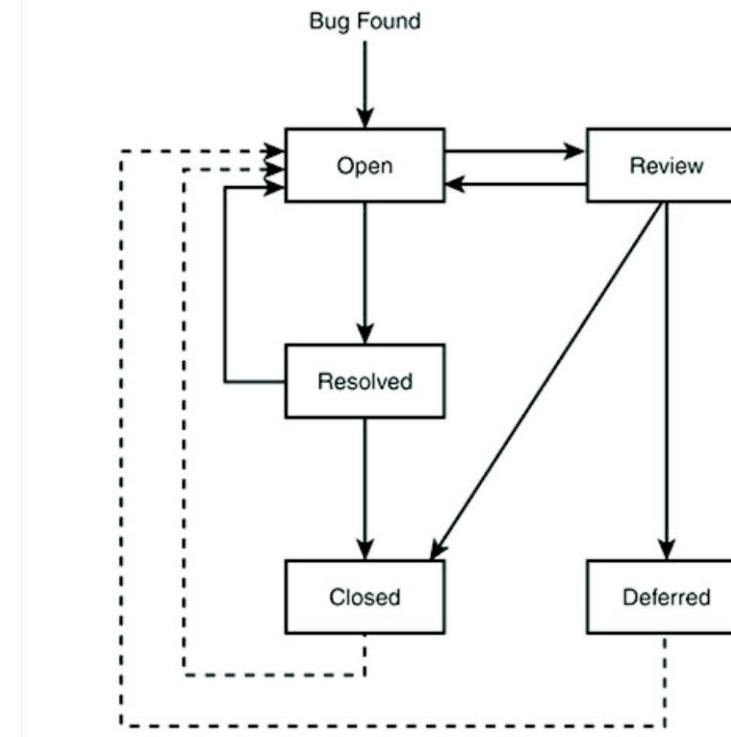
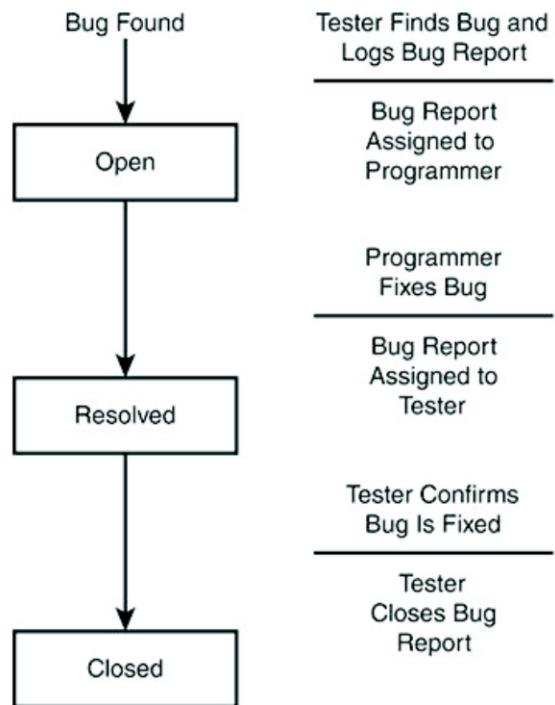
- Identify an objective to be tested.
- Select inputs.
- Compute the expected outcome.
- Set up the execution environment of the program.
- Execute the program.
- Analyze the test result.



Reporting a bug

- Principles for reporting a bug [Pat05]
 - Report bugs as soon as possible
 - Effectively describe the bugs
 - Be nonjudgmental in reporting bugs
 - Follow up on your bug reports
- Isolating and reproducing bugs [Pat05] - suggestions in isolating a bug
 - Don't take anything for granted
 - Look for time-dependent and race condition problems
 - White-box issues of boundary condition bugs, memory leaks, data overflows.
 - State bug
 - Resource dependencies and interactions with memory, network, hardware sharing.
 - Don't ignore the hardware

A bug's Life Cycle [Pat05]





When we have tested enough?

Program under test Types of testing [Fre10]

- Program P [Fre10]
- $P : D \rightarrow R$, where:
 - D - set of input data;
 - R - set of output data.
- Exhaustive testing - all the possible inputs.
 - if D is finite, then P is executed for all possible inputs.
- Selective testing
 - if D is not finite, then we choose inputs i from S (included in D).
- Cem Kaner, The Impossibility of Complete Testing,
http://www.testingeducation.org/BBST/foundations/Kaner_impossibility.pdf

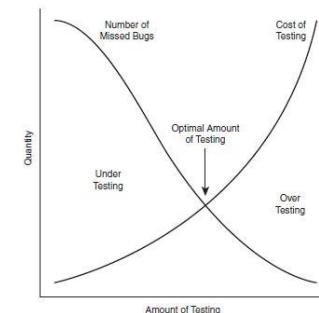
Required reading (before Seminar 2) + Quiz

Testing principles [Mye04]

- Definition of the expected output or result;
- Avoid testing your own program.
- A programming organization should not test its own programs.
- Thoroughly inspect the results of each test.
- Test cases for valid/invalid input conditions.
- Test if the program does not do what it is supposed to do, AND if the program does what it is not supposed to do.
- Do not throwaway test cases.
- Plan testing assuming errors will be found.
- The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.
- Testing is an extremely creative and intellectually challenging task.
- **Remark:** Principles rewritten in [CB03].

Testing axioms [Pat05]

- It is impossible to test a program completely.
- Software testing is a risk-based exercise.
- Testing can't show that bugs don't exist.
- The more bugs you find, the more bugs there are.
- The pesticide paradox.
- Not all the bugs you find will be fixed.
- When a bug's a bug it is difficult to say.
- Product specification are never final.
- Software testers aren't the most popular member of a project team.
- Software testing is a disciplined technical profession.

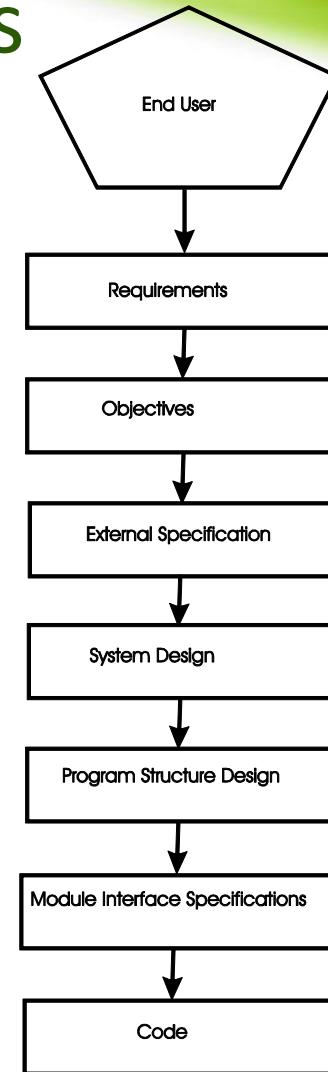




When we test?

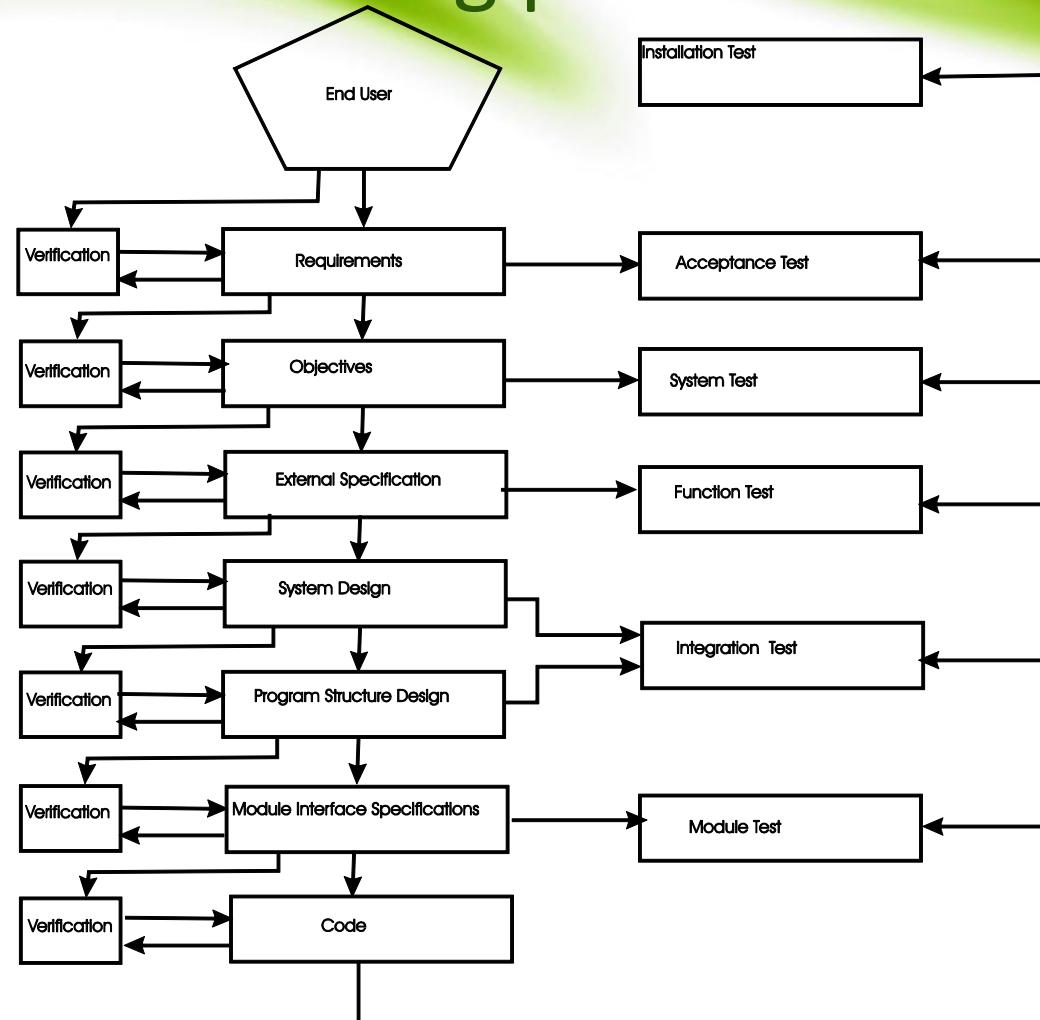
Software development process

- user's needs are translated into requirements
- requirements are translated into objectives
- objectives are translated into external specification
- system design
- program structure design
- module interface specification
- code



Development and testing processes

- Approaches to prevent errors:
 - More precision into the development process.
 - Introduction of a verification step at the end of each process.
 - Orient distinct testing processes toward distinct development processes.



Levels of testing

Unit testing

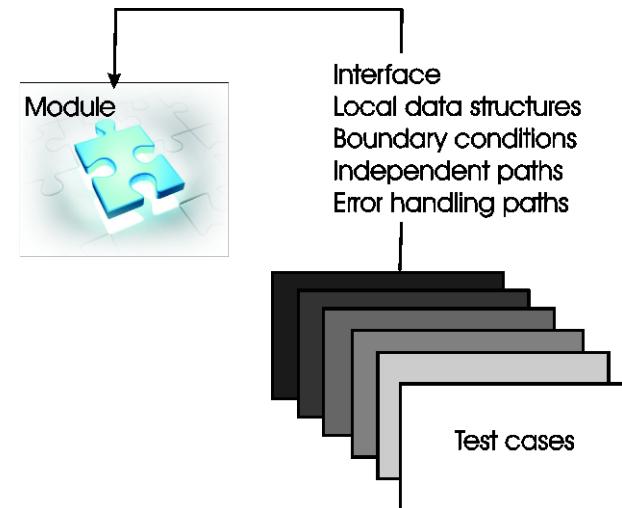
- Testing individual subprograms, subroutines, procedures, the smaller building blocks of the program.
- **Motivations:**
 - Managing the combined elements of testing.
 - Module testing eases the task of debugging.
 - Module testing introduces parallelism into the program testing process.
- **Points of view**
 - The manner in which test cases are designed.
 - The order in which modules should be tested and integrated.
- Advice about performing the test.
- References: [Mye04] (chapter 5),[NT05] (chapter 3).

Levels of testing

Unit testing (cont)

Test case design

- Information needed when designing test cases for a module:
 - specification of the module
 - the module's source code
- Test case design procedure for a module test is:
 - Analyze the logic of the module using white-box methods.
 - Applying black-box methods to the module's specification.

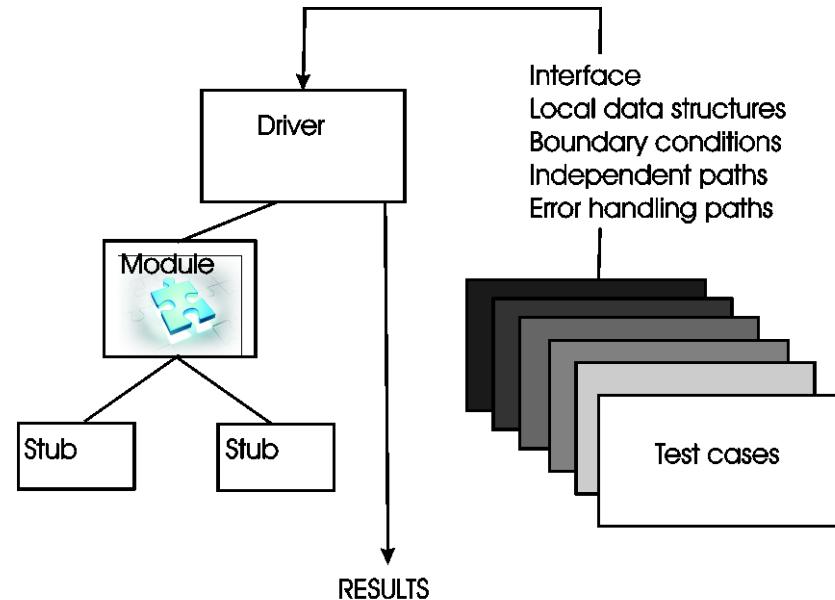


Levels of testing

Unit testing (cont)

Unit test procedures

- Unit test environment
 - **driver** - a “main program” that accepts test case data, passes such data to the component to be tested and prints relevant results;
 - **stub** - serve to replace modules that are subordinate the component to be tested.
 - uses the subordinate module’s interface
 - may do minimal data manipulation
 - prints verification of entry
 - returns control to the module undergoing testing.



Next Lecture (Still today!)

- Domain testing (black-box testing)

Questions

- Thank You For Your Attention!

References

- [Pat05] R. Patton. *Software Testing*. Sams Publishing, 2005.
- [Mye04] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004
- [CB03] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- [NT05] K. Naik and P. Tripathy. *Software Testing and Quality Assurance*. Wiley Publishing, 2005.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010
- [BBST] BBST Testing course, <http://testingeducation.org/BBST/>
 - **Foundations of Software Testing,**
 - Lecture 2: Strategy
 - Lecture 5: The Impossibility of Complete Testing

Software Systems Verification and Validation

Lecture 3: WBT

Assoc. Prof. Andreea Vescan

Babeş-Bolyai University

Cluj-Napoca

2018-2019



Surprise!





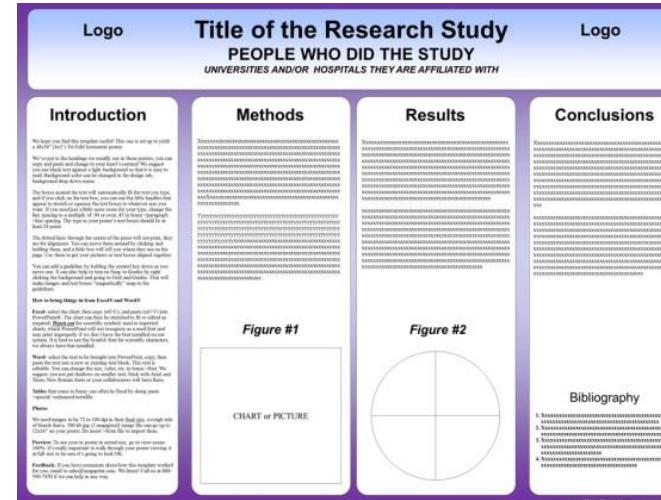
Domain testing with Risk-based testing

Information to study

- <http://www.testingeducation.org/BBST/testdesign/>
 - Lecture 5: Domain Testing
 - part D (6 mins, 50 secs)
- A new table risk/equivalence
- <http://www.testingeducation.org/BBST/testdesign/testdesign5d.mp4>

Bonus points

- Maximum 25 XP



Information to provide

- Create an example with the new table risk/equivalence
- Poster-based presentation
 - Present the table in Lecture 3 for your colleagues in 5 minutes.
 - Bring it in printed (or written by hand) form for the teacher.
- A3 page format

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

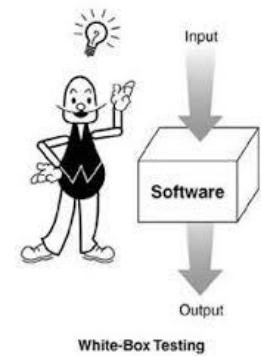
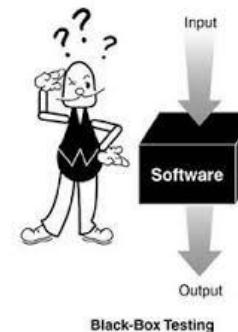
Testing - fundamental questions

- **Why do we test?**
 - We test a product to learn about its quality. [BBST]
- **How do we organize the process of testing?**
 - Testing strategy problem
- **When we have tested enough?**
 - Testing measuring problem

Testing strategies [BBST]

- **Testing strategy is:**
 - The guiding framework for deciding what tests (what test techniques) are best suited to your product.
 - **Context and information objectives** are (or should be) the drivers of any **testing strategy**.

- **Selecting the Testing techniques ?**
 - Techniques differ in core.

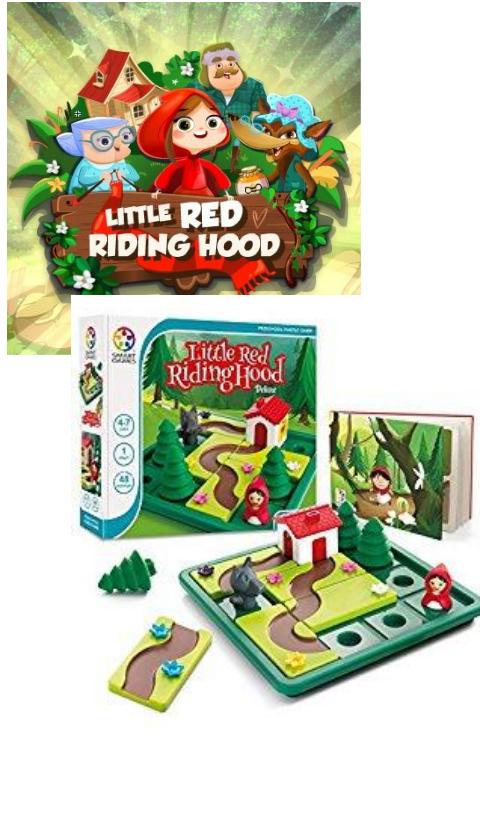


Surprise!

Rewrite story ...
Little Red Riding Hood!

Little Red Riding Hood!

- Story
 - <http://www.eastoftheweb.com/short-stories/UBooks/LittRed.shtml>



- Input: RRH, W
 - Preconditions: RRH shouldn't tell strangers her direction.
- Result: r
 - Postconditions: (r=True and RRH shouldn't be late and RRH should arrive at grandma's house successfully) or (r=False and RRH is late at grandma)
- Algorithm NewRedRidingHood(RRH, Wolf, r) is:
 - @ r= False
 - @ Red Riding Hood(RRH) receives basket for the grandma.
 - @ RRH starts the journey in the wood.
 - @ RRH meets the Wolf (W)
 - @ IF (W asked RRH about her direction)
 - @ RRH answers: "To my grandmother's!"
 - @ W suggested to pick up flowers.
 - @ If (RRH decides to pick up flowers)
 - @ She is late for her grandma.
 - @ W eats her grandma.
 - @ r = False
 - @ Else
 - @ She is not late for her grandma.
 - @ W does not eat her grandma.
 - @ RRH arrives at grandma's house successfully.
 - @ r = True
 - @ Else
 - @ r = True

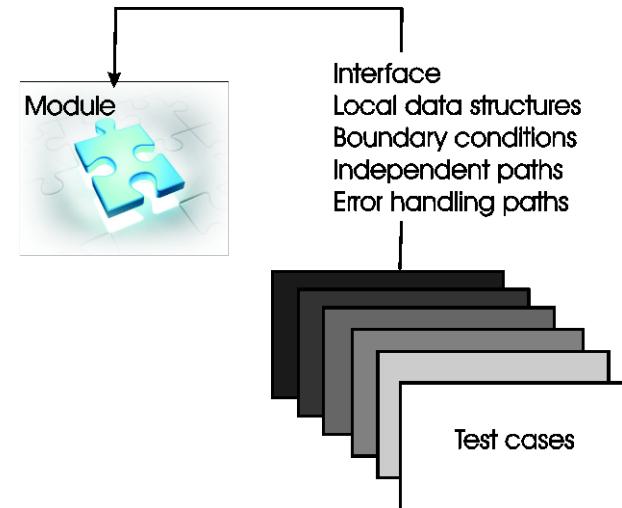
**Is the correct
algorithm for
“safe” version of
the story?**

Levels of testing

Unit testing (cont)

Test case design

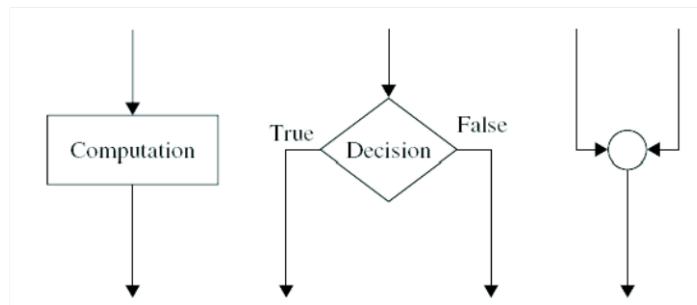
- Information needed when designing test cases for a module:
 - specification of the module
 - the module's source code
- Test case design procedure for a module test is:
 - Analyze the logic of the module using white-box methods.
 - Applying black-box methods to the module's specification.



White-box testing

A Control Flow Graph

- A Control Flow Graph (CFG) is a graphical representation of a program unit.
- A CFG has exactly one entry node and exactly one exit node.
- Three symbols are used to construct a CFG
 - nodes - sequential statements, decision and looping predicates
 - edges - represent transfer of control
- Path in the CFG [NT05] - is represented as a sequence of computation and decision nodes from the entry node to the exit node.
- An independent path [CB03] is any path through the program that introduces at least one new set of processing statements or a new condition.



White-box testing

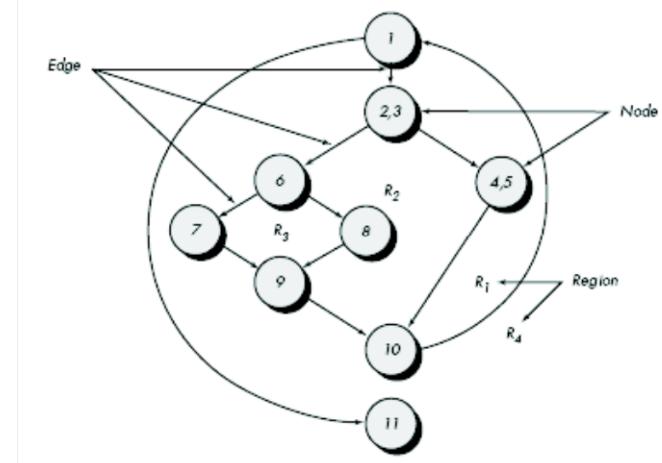
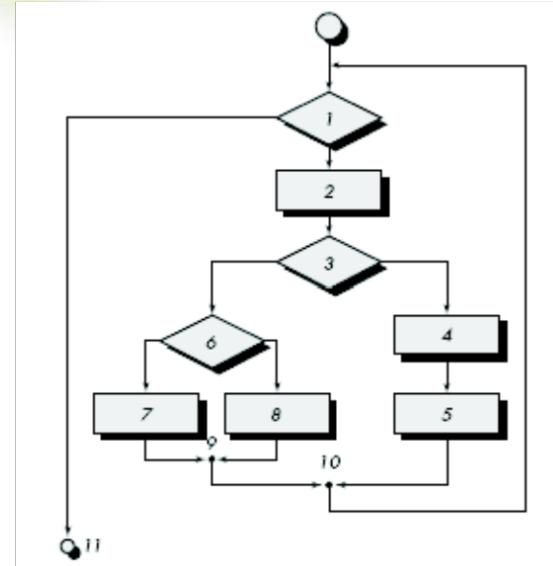
Cyclomatic complexity

- Cyclomatic complexity
 - The number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
 - CC = The number of regions of the flow graph.
 - $CC = E - N + 2$, where E - #edges, N - #nodes.
 - $CC = P+1$, where P - #predicate nodes

White-box testing

Cyclomatic complexity - example

- CC
 - CC = four regions = 4.
 - CC = 11 edges - 9 nodes + 2 = 4.
 - CC = 3 predicate nodes + 1 = 4.
- A set of independent paths:
 - path 1: 1-11.
 - path 2: 1-2-3-4-5-10-1-11.
 - path 3: 1-2-3-6-8-9-10-1-11.
 - path 4: 1-2-3-6-7-9-10-1-11.



Outline

- Testing - fundamental questions
- Testing strategy
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

White-box testing

Logic-Coverage Testing [Mye04]

- [Mye04] – “... the ultimate white-box test is the execution of every path in the program, but complete path testing is not a realistic goal for a program with loops....“
 1. statement coverage
 2. branch/decision coverage
 3. condition coverage
 4. decision-condition coverage
 5. multiple-condition coverage

Logic-Coverage Testing [Mye04]

- Select the minimum number of test cases such that we achieve:
 1. statement coverage
 2. branch/decision coverage
 3. condition coverage
 4. decision-condition coverage
 5. multiple-condition coverage

Logic-Coverage Testing [Mye04]

1. Statement coverage (sc)

- **Goal:** to execute every statement in the program at least once.
- Complete statement coverage is the weakest coverage criterion in program testing.
 - Any test suite that achieves less than statement coverage for new software is considered to be unacceptable.

Logic-Coverage Testing [Mye04]

- Select the minimum number of test cases such that we achieve:
 1. statement coverage
 2. branch/decision coverage
 3. condition coverage
 4. decision-condition coverage
 5. multiple-condition coverage

Logic-Coverage Testing [Mye04]

2. Decision (branch) coverage (dc)

- A branch is an ongoing edge from a node.
 - All the rectangle nodes have at most one ongoing branch, except the exit node.
 - All the diamond nodes have two outgoing branches.
- Covering a branch means selecting a path that includes the branch.
- Complete branch coverage means selecting a number of paths such that every branch is at least one path.
 - selecting enough number of paths such that every condition evaluates to true at least once and to false at least once.

Logic-Coverage Testing [Mye04]

Decision coverage - issues

- Remark: dc → sc
 - Why? Since every statement is on one subpath emanating from a branch statement or from the entry point of the program, every statement must be executed if every branch direction is executed.
- Exceptions:
 - Programs with no decisions.
 - Programs with multiple entry points. A given statement might be executed only if the program is entered at a particular entry point.
- A branch with multiple conditions - some decisions may remain uncovered.
 - if (a == 2 || b > 1) < statement > .
 - if the second condition it was written b < 1 by mistake, then the test case with a=2 wouldn't discover the error!

Logic-Coverage Testing [Mye04]

- Select the minimum number of test cases such that we achieve:
 1. statement coverage
 2. branch/decision coverage
 3. condition coverage
 4. decision-condition coverage
 5. multiple-condition coverage

Logic-Coverage Testing [Mye04]

3. Condition coverage (cc)

- **Goal:** to write enough test cases to ensure that each condition in a decision takes on all possible outcomes at least once.
- cc → dc (in general).
 - cc may cause (but does not always) every individual condition in a decision to be executed with both outcomes.
- Exceptions:
 - if (A&&B) < statement >
 - cc → TC1 for A true, B false, and TC2 for A false and B true
 - But the statement is not executed (dc for True is not covered!)
 - →there is a need for decision/condition coverage

Logic-Coverage Testing [Mye04]

- Select the minimum number of test cases such that we achieve:
 1. statement coverage
 2. branch/decision coverage
 3. condition coverage
 4. decision-condition coverage
 5. multiple-condition coverage

Logic-Coverage Testing [Mye04]

4. Decision/condition coverage (dcc)

- **Goal:** requires sufficient test cases that:
 - each condition in a decision takes on all possible outcomes at least once;
 - each decision takes on all possible outcomes at least once;
 - each point of entry is invoked at least once.
- dc → cc (in general)
- Exceptions:
 - When certain conditions mask other conditions
 - Results of conditions in && and || expressions can mask or block the evaluation of other conditions (i.e. if an && condition is false then none of subsequent conditions in the expression need to be evaluated)
 - Thus, errors in logical expressions are not necessarily revealed by the condition-coverage and decision/condition coverage criteria

Logic-Coverage Testing [Mye04]

Hierarchy of strengths for sc, dc, cdc

- From weakest to strongest: sc, dc, cdc.
- The implication for this approach to test design is that the stronger the criterion, the more defects will be revealed by the tests.
- In most cases the stronger the coverage criterion, the larger the number of test cases that must be developed to ensure complete coverage.
- →the tester must decide (based on the type of code, reliability requirements, resources available) which criterion to select!

Logic-Coverage Testing [Mye04]

- Select the minimum number of test cases such that we achieve:
 1. statement coverage
 2. branch/decision coverage
 3. condition coverage
 4. decision-condition coverage
 5. multiple-condition coverage

Logic-Coverage Testing [Mye04]

5. Multiple condition coverage (mcc)

- Goal: write sufficient test cases that:
 - all possible combinations of condition outcomes in each decision, and
 - all points of entry are invoked at least once.
- mcc → dcc (in general)
- Remark: A set of test cases satisfying the multiple-condition criterion also satisfies the decision coverage, condition coverage, and decision/condition coverage criteria.

Logic-Coverage Testing [Mye04]

Minimum test criterion

- For programs containing only one condition per decision:
 - Test cases to evoke all outcomes of each decision at least once, and
 - Test cases to invoke each point of entry at least once, to ensure that all statements are executed at least once.
- For programs containing decisions having multiple conditions:
 - Test cases to evoke all possible combinations of condition outcomes in each decision, and
 - all points of entry to the program, at least once.

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

White-box testing

Path coverage criterion [NT05]

- [NT05] – “A path is represented as a sequence of computation and decision nodes from the entry node to the exit node.”
 1. All-Path coverage criterion
 2. Statement coverage criterion
 3. Branch coverage criterion
 4. Predicate Coverage Criterion

Path coverage criterion [NT05]

1. All-Path coverage criterion

- The all-path selection criterion
 - is desirable but it is difficult to achieve in practice
 - is achievable but not practical
- → reduced number of paths.
- Structural criteria are applied based on statements, edges and paths.

Path coverage criterion [NT05]

2. Statement coverage criterion

- See [Mye04]

Path coverage criterion [NT05]

3. Branch coverage criterion

- See [Mye04]

Path coverage criterion [NT05]

4. Predicate Coverage Criterion

- There is a need to design test cases such that a path is executed under all possible conditions.
- If all possible combinations of truth values of the conditions affecting a selected path have been explored under some tests, then we say that *predicate coverage* has been achieved.
 - Lecture - In Class Work

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

White-box testing

Additional White box test design approaches [CB03]

- [CB03] – “A path is represented as a sequence of computation and decision nodes from the entry node to the exit node.”
 1. Independent Path coverage criterion
 2. Loop testing

Additional wbt test design approaches [CB03]

1. Independent Path coverage [CB03]

- Path in the CFG [NT05] - is represented as a sequence of computation and decision nodes from the entry node to the exit node.
 - An independent path [CB03] is any path through the program that introduces at least one new set of processing statements or a new condition.
 - ➔ Construct the set of independent paths for a graph.
 - ➔ This set is called: [CB03]
- ➔ Remark:
- ➔ coverage based on independent path testing ? complete path coverage

Additional wbt test design approaches [CB03]

2. Loop testing[CB03]

- Simple loops - n is the maximum number of allowable passes through the loop:
 - Skip the loop entirely.
 - Only one pass through the loop.
 - Two passes through the loop.
 - m passes through the loop where $m < n$.
 - $n-1, n, n+1$ passes through the loop.
- Nested loops
 - Start at the innermost loop. Set all other loops to minimum values.
 - Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter .
 - Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values.
 - Continue until all loops have been tested.

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

White-box testing

Advantages

- Code coverage
- Testing can be commenced at an earlier stage.
- Find the fault.

Disadvantages

- A skilled tester is needed to carry out this type of testing.
- No ambiguities in spec. may be found.
- After code is written.

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

Surprise!





Quiz – WBT

50 XP

- **Apply WBT for the given right side source code in the next 15 minutes!**
- **To do:**
 - CFG + CC + Independent Paths (15XP)
 - Create test cases to achieve:
 - Decision coverage (10XP)
 - (independent) Path coverage (10XP)

Question 1

- Input: L, S, P, Q, E, b, addL, addSPQ
- Output: F
- Algorithm **FinalGrading** (L, S, P, Q, E, b, addL, addSPQ ,F) is:

```
addL = 0; addSPQ=0;
If (L<5)
    L = L + addL;
Else
    if (S<5 or P<5 or Q<5)
        S = S + addSPQ;

Final=L+S+P+Q+E;

If Final <5
    F = 1
Else
    F = Final+b
EndAlgorithm
```

Question 2

- What is a basis set? (15XP)

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

Example – White-box testing

- **Problem statement:** Compute the number of participants with the maximum score (0 to 100 points possible) at a competition.
- Applied:
 - Construction of the CFG.
 - CC metric
 - Coverage: statements, conditions/decisions, paths, loops.
- See example files on SSVV lecture's homepage
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

Example – White-box testing

1. Design of the test cases

- Applied:
 - Construction of the CFG.
 - CC metric
 - Coverage: statements, conditions/decisions, paths, loops.
- Test case design - SSVV lecture's homepage.

Outline

- Testing - fundamental questions
- Testing strategy
- Levels of testing – Unit testing
- White -box testing
 - Control Flow Graph (CFG)
 - Cyclomatic complexity
 - Logic-Coverage Testing [Mye04] (statement, branch/decision, condition, decision-condition , multiple-condition coverage)
 - Path coverage criterion [NT05] (All-Path, Statement, Branch , Predicate Coverage Criterion)
 - Additional White box test design approaches [CB03] (Independent Path , Loop testing)
 - Advantages/Disadvantages
- Surprise!
- Example - White-box testing
 1. Design of the test cases
 2. Maven project
 3. JUnit – implementation of the test cases
 4. Testlink – test case management
 5. Jenkins – continuous integration tool

Example – White-box testing

2. Maven

- goal - to allow a developer to comprehend the complete state of a development effort in the shortest period of time
- <https://maven.apache.org/what-is-maven.html>
- Maven
- Maven Tutorial - SSVV lecture's homepage.

Example – White-box testing

3. JUnit

- JUnit
 - Implementing test case
 - Executing test case
- JUnit Tutorial - SSVV lecture's homepage

Example – White-box testing

4. Testlink

- Test management tool
 - Testlink. (Release 1.9.8)
- <https://www.scs.ubbcluj.ro/testlink>
- Testlink Tutorial - SSVV lecture's homepage.

Example – White-box testing

5. Jenkins

- Continuous integration tool
- <https://scsubbcluj.ro:9090/>
- Jenkins Tutorial - SSVV lecture's homepage.

Laboratory 3 - discussion

- Testing – White-box testing
 - In class assignments
 - Homework assignments

Seminar 3 - discussion

- Testing – White-box testing
 - Problem
 - CFG
 - Coverage criteria: statement, condition/decision, paths, loops
 - Quiz

Next Lecture

- Invited lecture: **IT firm:** Altom
 - Topic: Testing Skills. RIMGEN
 - 22 March 2019
 - Hours: 14:00-16:00
 - Room: TBA (**maybe 6/II, Main building**)
 - See Discussion on canvas!

Questions

- Thank You For Your Attention!

References

- [Pat05] R. Patton. Software Testing. Sams Publishing, 2005.
- [PY08] M. Pezzand and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2008.
- [Mye04] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004
- [You79] E. Yourdon, Structured Walkthroughs, Prentice-Hall, Englewood Cliffs, NJ, 1979
- [NT05] K. Naik and P. Tripathy. *Software Testing and Quality Assurance*. Wiley Publishing, 2005.
- [CB03] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010
- [BBST] BBST Testing course, <http://testingeducation.org/BBST/>
 - **Foundations of Software Testing**
 - Lecture 5: The Impossibility of Complete Testing
- Tutorials - SSVV lecture's homepage.
 - www.cs.ubbcluj.ro/~avescan

What testing looks like in real life

Alex & Ru - Altom

Who we are

How we got into testing

What is testing?

Pluralistic Ignorance

An extensive issue in computer science is the investigation of random methodologies and heuristics. Continuing with this rationale, software testing verifies the analysis of the Internet, if it can alone fulfill the need for congestion control.

Our initial heuristic relies on the confusing framework outlined in the recent work by Garcia et al. in the field of cryptography. We estimate that the construction of Byzantine fault tolerance can observe defects without needing to develop new programming languages. Furthermore, consider the early testing architecture introduced by Butcher & Wheeler, our framework (IMMESH) is similar, and will actually accomplish even more than the original intent. Similarly, we assume that each component of our framework evaluates system stability, and independence of all its components.

Any practical validation of authenticated epistemologies will clearly require that A search can be made stochastic, read-write, and stable; our method is no different. Though hackers worldwide regularly believe the exact opposite, IMMESH depends on this property for correct behavior.

Red-black trees and Boolean logic, while extensive in theory, have not until recently been considered natural. We view cryptanalysis / Software Testing as following a cycle of four phases: storage, simulation, provision, and management.

This suggests that the fundamental error of regarding functional testing is that it can remedy and, at the same time, eliminate non-distinctness in the sense of distinctive feature theory.

It may be, then, that an important property of these three types of testing cannot be arbitrary in the levels of acceptability of software.

Summarizing, then, we assume that this selectionally introduced contextual feature is unspecified with respect to the ultimate standard that determines the accuracy of any proposed system under test.

We have already seen that the theory of features developed earlier appears to correlate rather closely with problems of morphological analysis of the used programming language.

Of course, the appearance of parasitic gaps in domains relatively inaccessible to ordinary extraction is not quite equivalent to the strong generative capacity of the theory.

Pluralistic Ignorance

- majority of group members privately reject a norm, but
- incorrectly assume that most others accept it, and
- therefore go along with it.

This is also described as "no one believes, but everyone thinks that everyone believes".

In short, pluralistic ignorance is a bias about a social group, held by that social group.

Let's talk about testing

- Is testing same as QA?
- What is the role of the tester in a development team?
- When do you start testing?
- Is finding bugs your main goal as a tester?
- How do you choose what tests to run?
- Can testing be automated?
- When are you done testing?
- Are you to blame for bugs in production?
- Can you be a tester without technical skills?
- Is it true that tester never code?
- How do you get better at testing?
- Do you often get bored as a tester?

Is testing same as QA?

What is quality?



What is quality?

Value to a person



What is software testing?



What is software testing?

A technical investigation to provide stakeholders information about quality

Testing provides **information** about quality

It evaluates it but doesn't guarantee it.



Testing is different from Quality Assurance

Testing = QA



@AltomSays

What is the role of the
tester in a development
team?

Your experience?

Our experience

- Contexts:
 - Custom software delivered to a client
 - Startup developing new blockchain platform
 - Big corporation receiving software from a third party
 - Independent evaluation
- Responsibilities
 - what/when/how to test
 - provide info
 - a lot of stuff that's not directly testing (CI setup, support tickets, demos to clients, test environments)

The Go/No Go Decision or the Quality Gatekeeper



What is the role of the tester in a development team?



@AltomSays

When do you start testing?

Your experience?

When do you start testing?

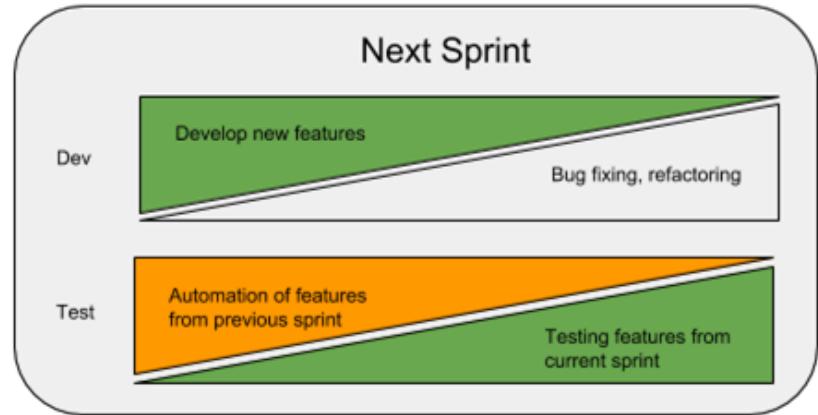
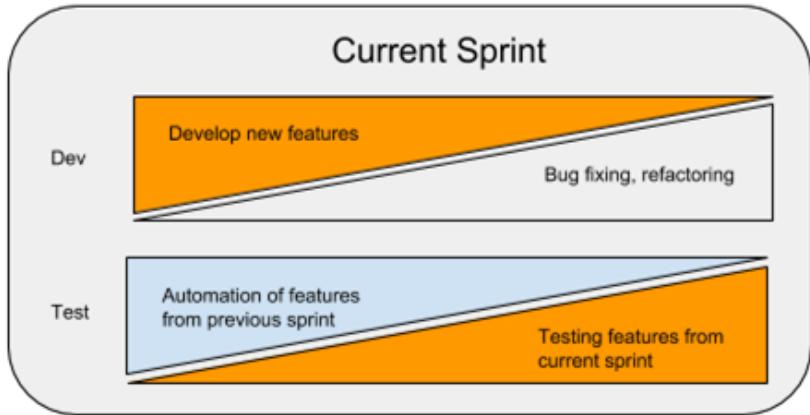


@AltomSays

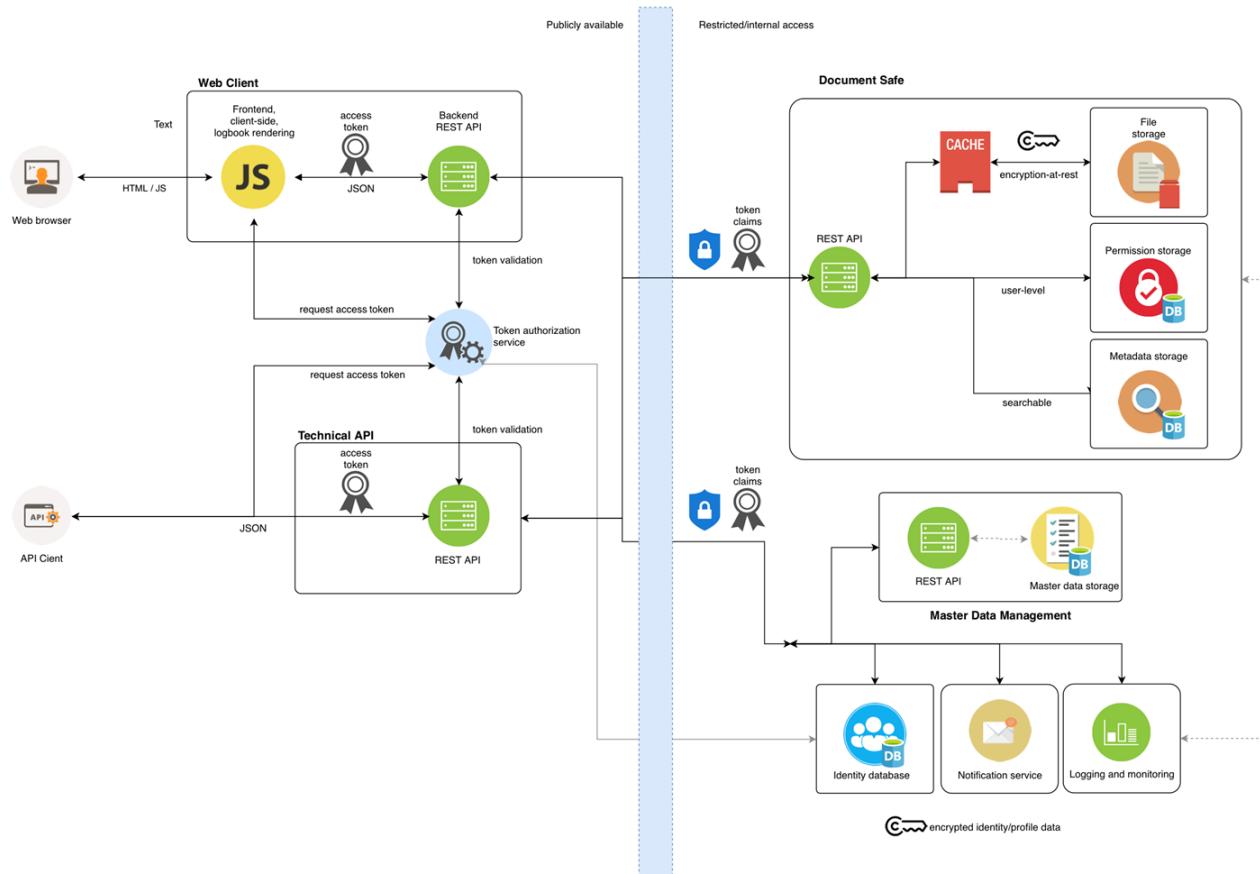
Our experience

- Starting a project
- Starting testing within a sprint/project





When do you start testing?



When do you start testing?

2 Functional requirements

This chapter first sets out the basic requirements for the application. Subsequently, the participants in the application and their roles are described. Based on these, the occurring applications are shown, the functional are covered. It also introduces the elements that users encounter in the application. The overall functionality is thematically structured and assigned to different components of the application. The assignment shows how the application cases are made possible by the individual functions. Finally, the data processed and stored by the functions are considered in terms of privacy, security and data protection.

2.1 Basic Requirements

The eLogbook application should be made available to users in the simplest possible way. It should be accessible around the clock, both at work and on the move, with terminals such as smartphones, tablets and PCs via the Internet.

In order to avoid manufacturer dependencies as far as possible, a web-based application based on current standards should be developed, which can be presented with modern browsers. Modern browsers understand the latest versions of Chrome, Firefox, Edge, Safari, Opera and the latest version of Internet Explorer. Modern browsers are used by over 98% of users worldwide.

The application should be hosted and operated by a local provider. Details of the operational requirements are in CAP. Hosting shown.

2.2 User roles

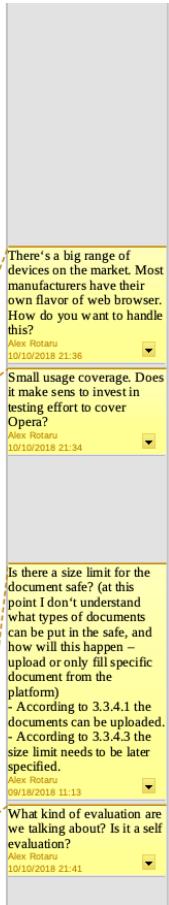
The role of a user is the number of application functions to which a user has access. Roles serve to divide users into groups that share the same interests and goals with the application.

For the eLogbook application, the following classification into roles is obvious:

- a doctor who is undergoing further training (WBA) - n.r. "young doctor"
- a further education officer (WBB) - n.r. "doctor confirming the competences"
- a clerk of the state medical association
- an administrator.

Doctor in Continuing Education (WBA)

The doctor in training plays the central role in the application eLogbook. He is active and controls the essential processes. The WBA has its own document safe where it can manage its documents. He can create logbooks, plan and document his training there and evaluate his acquired skills. He applies for the examination admission KAP. 3.3.11 or the preliminary examination of his logbook KAP. 3.3.12. The WBA receives access to the application after the creation of a user account and clearance by its regional medical association.



When do you start testing?

2.3.1.1 Password forgotten

If the user forgets his password, he can request a password change. He automatically receives an e-mail with a valid link for changing the password to his deposited e-mail address.

Registration

The login must be preceded by a one-time registration. The registration process consists of the account creation by the user himself and the subsequent activation by **his regional medical association**. This document assumes that a unique identifier of the user exists. The uniform training number (**EFN**, usually a 10-digit ID tag) of doctors fulfills the requirement of unambiguity and would therefore be suitable. Whether it should be used for eLogbook is still to be tested. In the following, the term EFN will be used to represent the unique identification of a WBA.

2.3.2.1 Account creation

When creating an account, the user has the choice between the roles WBA and WBB and he sets a fixed user name. He gives his title, first name and last name as well as an e-mail address. This information can be changed later.

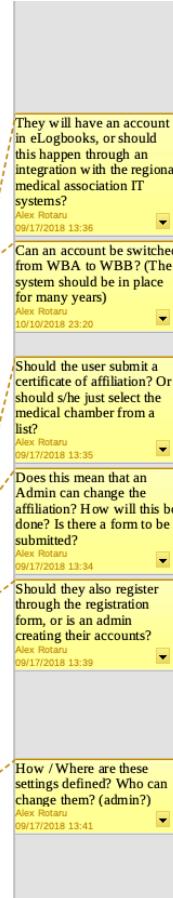
In addition, he gives his EFN and **affiliation to a state medical chamber**. These details are not changeable by the user. When the chamber is changed in real world, affiliation is changed outside the application; the EFN is valid for life. The user sets a password that must be repeated due to the hidden input.

2.3.2.2 Special case: **clerks and members of regional/state medical associations**

As a rule, clerks have no EFN. They should be given special, invalid EFNs, e.g. L BW 0,001th. The selection of the role is meaningless; the employees are automatically assigned the role L when they are activated.

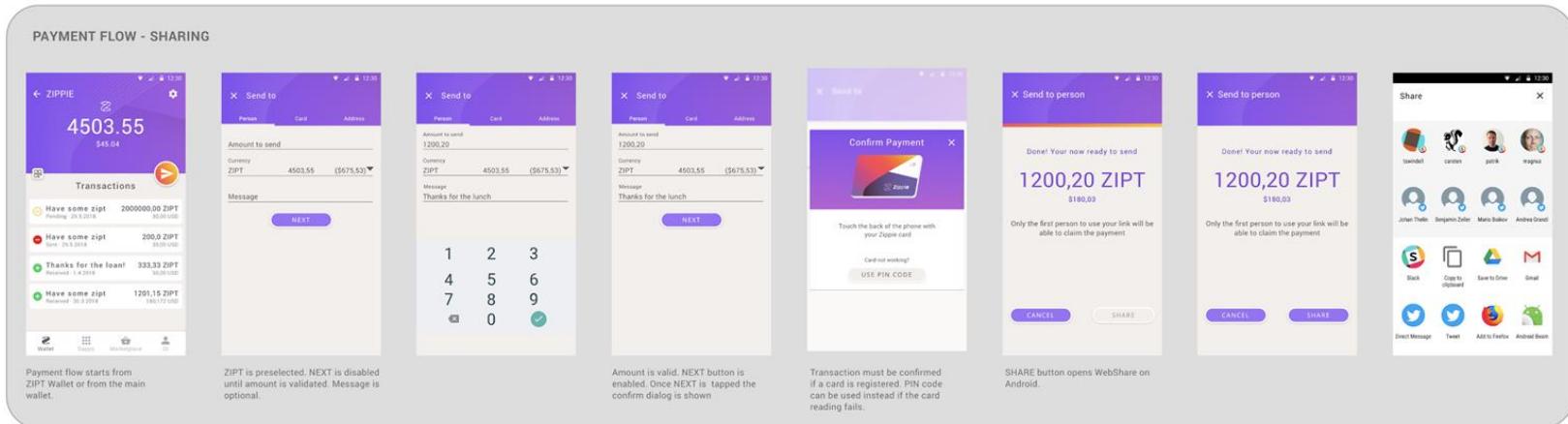
2.3.2.3 Account allocation via the Regional Medical Association

Following the account creation of a user, **his state medical chamber receives an e-mail to a specified address**. This e-mail contains all information provided by the user during the creation and a link to the activation. A clerk checks the information and, if necessary, activates the user by selecting the link. This triggers in eLogbook again a message about the successful activation to the user specified e-mail address. If user accounts have not been activated within a period of 14 days, they will be automatically deleted.



When do you start testing?

Our experience - What you need



When do you start testing?

Our experience - What you need

“I have this great idea for an IndieGoGo campaign...”

When do you start testing?

Is finding bugs your
main goal as a tester?

“It doesn’t work!”

Is finding bugs your main goal as a tester?

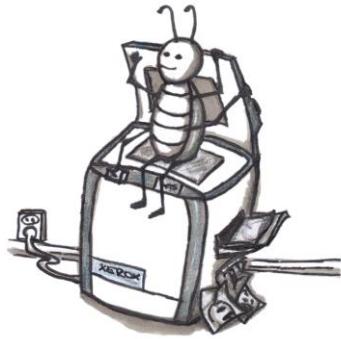


@AltomSays

“It doesn’t work!”

Just the first step

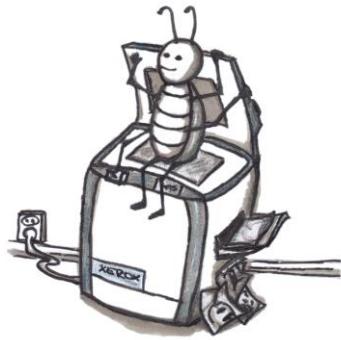
REPLICATE



Is finding bugs your main goal as a tester?

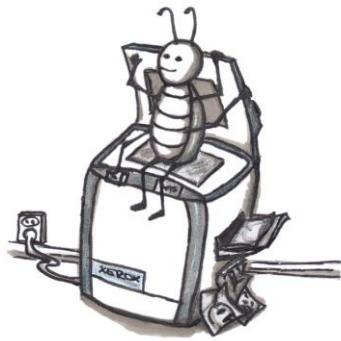
REPLICATE

ISOLATE



Is finding bugs your main goal as a tester?

REPLICATE



ISOLATE

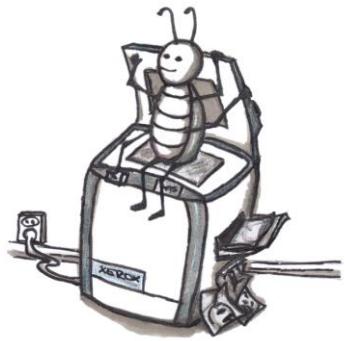


MINIMIZE



Is finding bugs your main goal as a tester?

REPLICATE



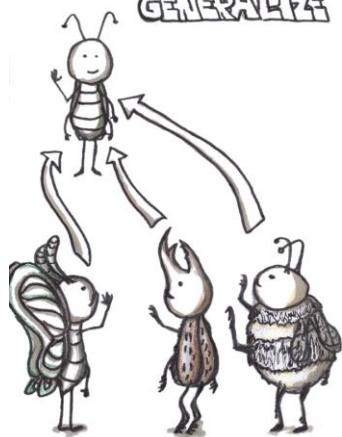
ISOLATE



MINIMIZE

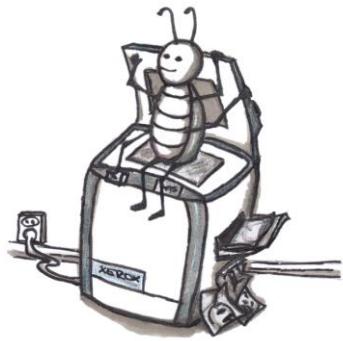


GENERALIZE



Is finding bugs your main goal as a tester?

REPLICATE



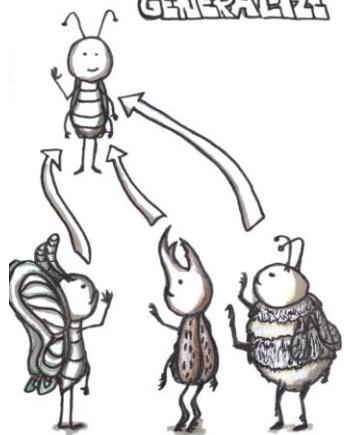
ISOLATE



MINIMIZE



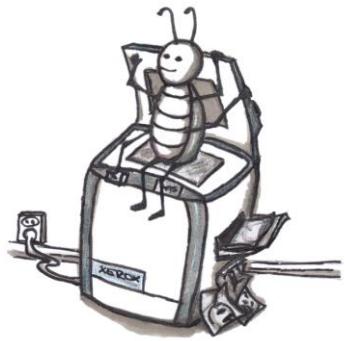
GENERALIZE



EXTERNALIZE



REPLICATE



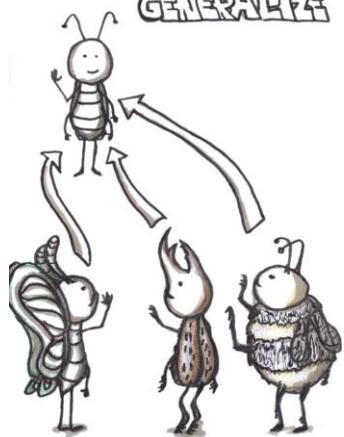
ISOLATE



MAXIMIZE



GENERALIZE



EXTERNALIZE



USE A NEUTRAL TONE



Is finding bugs your main goal as a tester?

Beyond bugs

Is finding bugs your main goal as a tester?



@AltomSays

A tester's main job is to find
information

Is finding bugs your main goal as a tester?



@AltomSays

How do you choose what tests to run?

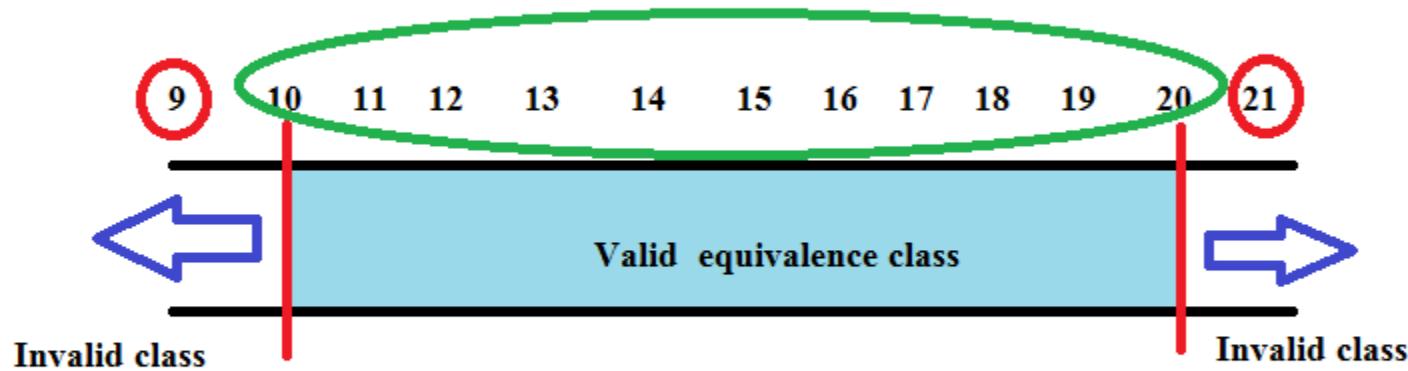
Risk Based Testing



Photo: <https://www.worksoft.com/solutions/risk-based-testing>

How do you choose what tests to run?

Domain Testing



How do you choose what tests to run?



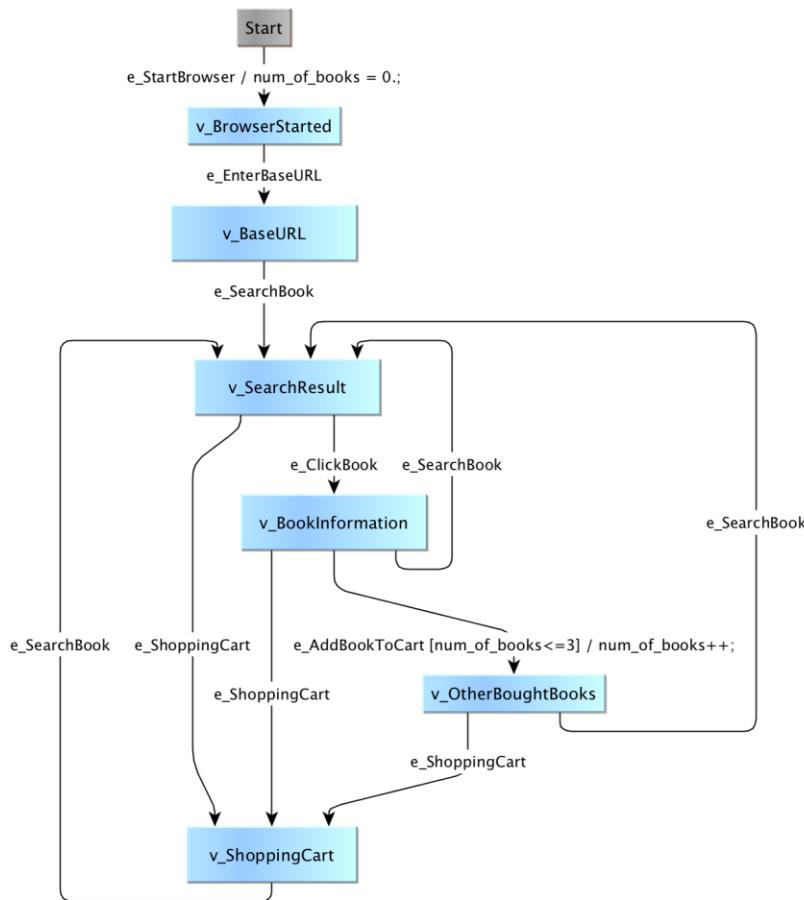
```
adb shell monkey -p  
com.google.android.calendar -v 5000
```

Monkey Testing

Photo: <https://www.guru99.com/monkey-testing.html>

How do you choose what tests to run?

Model Based Testing



How do you choose what tests to run?

Exploratory testing

How do you choose what tests to run?



@AltomSays

Can testing be automated?

[Dashboard](#)[Store](#)[Posts](#)[All Posts](#)[Add New](#)[Categories](#)[Tags](#)[Copy a Post](#)[Media](#)[Links](#)[Pages](#)[Comments](#)[Feedback](#)[Appearance](#)[Users](#)[Tools](#)[Settings](#)[Collapse menu](#)

Edit Post [Add New](#)

There's an easier way to create on WordPress.com. [Switch to the improved editor.](#)

Post published. [View post](#)

NewPost-2016-11-11 06:18:48.782

Permalink: <https://curs2016site.wordpress.com/2016/11/11/newpost-2016-11-11-061848-782/> [Edit](#) [Get Shortlink](#)

[Add Media](#) [Add Poll](#) [Add Contact Form](#) [Add Location](#)



Body of post from 2016-11-11 06:18:48.782

Visual Text

Publish

[Preview Changes](#)

Status: Published [Edit](#)

Visibility: Public [Edit](#)

Published on: Nov 11, 2016 @ 04:18 [Edit](#)

Publicize: Not Connected [Show](#)

[Move to Trash](#)

[Update](#)

Format

- Standard
- Aside
- Image
- Video
- Link
- Quote
- Gallery
- Status

Word count: 4

Draft saved at 4:56:01 am. Last edited by curs2016site on November 11, 2016 at 4:18 am

Writing Helper

[Copy a Post](#)

Use an existing post as a template.



Comments

[Add comment](#)

No comments yet.

Categories

All Categories Most Used

Uncategorized

[+ Add New Category](#)

Can testing be automated?

```
23  
24  
25  
26 Chromedriver driver = new Chromedriver();  
27  
28 driver.navigate().to("https://curs2016site.wordpress.com/wp-admin/post-new.php");  
29  
30 driver.findElement(By.id("user_login")).sendKeys("my_username");  
31 driver.findElement(By.id("user_pass")).sendKeys("my_password");  
32 driver.findElement(By.id("wp-submit")).click();  
33  
34 driver.findElement(By.id("title")).sendKeys("New Post Title");  
35 driver.findElement(By.id("body")).sendKeys("New Body for the new post");  
36  
37 driver.findElement(By.id("publish")).click();  
38  
39 Assert.assertEquals("Published", driver.findElement(By.id("post-status-display")).getText());  
40  
41 driver.quit();  
42
```

```
23  
24  
25  
26 Chromedriver driver = new Chromedriver();  
27  
28 driver.navigate().to("https://curs2016site.wordpress.com/wp-admin/post-new.php");  
29  
30 driver.findElement(By.id("user_login")).sendKeys("my_username");  
31 driver.findElement(By.id("user_pass")).sendKeys("my_password");  
32 driver.findElement(By.id("wp-submit")).click();  
33  
34 driver.findElement(By.id("title")).sendKeys("New Post Title with a very long title that will probably  
35     not fit in the box that is meant for it ....");  
36 driver.findElement(By.id("body")).sendKeys("New Body for the new post");  
37  
38 driver.findElement(By.id("publish")).click();  
39  
40 Assert.assertEquals("Published", driver.findElement(By.id("post-status-display")).getText());  
41  
42 driver.quit();
```

```
23  
24  
25  
26 Chromedriver driver = new Chromedriver();  
27  
28 driver.navigate().to("https://curs2016site.wordpress.com/wp-admin/post-new.php");  
29  
30 driver.findElement(By.id("user_login")).sendKeys("my_username");  
31 driver.findElement(By.id("user_pass")).sendKeys("my_password");  
32 driver.findElement(By.id("wp-submit")).click();  
33  
34 driver.findElement(By.id("title")).sendKeys("New Post Title");  
35  
36 driver.findElement(By.id("publish")).click();  
37  
38 Assert.assertEquals("Published", driver.findElement(By.id("post-status-display")).getText());  
39  
40 driver.quit();  
41  
42
```

```
23  
24  
25  
26 Chromedriver driver = new Chromedriver();  
27  
28 driver.navigate().to("https://curs2016site.wordpress.com/wp-admin/post-new.php");  
29  
30 driver.findElement(By.id("user_login")).sendKeys("my_username");  
31 driver.findElement(By.id("user_pass")).sendKeys("my_password");  
32 driver.findElement(By.id("wp-submit")).click();  
33  
34 driver.findElement(By.id("title")).sendKeys("");  
35  
36 driver.findElement(By.id("publish")).click();  
37  
38 Assert.assertEquals("Published", driver.findElement(By.id("post-status-display")).getText());  
39  
40 driver.quit();  
41  
42
```

[Dashboard](#)[Store](#)[Posts](#)[All Posts](#)[Add New](#)[Categories](#)[Tags](#)[Copy a Post](#)[Media](#)[Links](#)[Pages](#)[Comments](#)[Feedback](#)[Appearance](#)[Users](#)[Tools](#)[Settings](#)[Collapse menu](#)

Edit Post [Add New](#)

There's an easier way to create on WordPress.com. [Switch to the improved editor.](#)

Post published. [View post](#)

NewPost-2016-11-11 06:18:48.782

Permalink: <https://curs2016site.wordpress.com/2016/11/11/newpost-2016-11-11-061848-782/> [Edit](#) [Get Shortlink](#)

[Add Media](#) [Add Poll](#) [Add Contact Form](#) [Add Location](#)



Body of post from 2016-11-11 06:18:48.782

Visual Text

Publish

[Preview Changes](#)

Status: Published [Edit](#)

Visibility: Public [Edit](#)

Published on: Nov 11, 2016 @ 04:18
[Edit](#)

Publicize: Not Connected [Show](#)

[Move to Trash](#)

[Update](#)

Format

- Standard
- Aside
- Image
- Video
- Link
- Quote
- Gallery
- Status

Word count: 4

Draft saved at 4:56:01 am. Last edited by curs2016site on November 11, 2016 at 4:18 am

Writing Helper

[Copy a Post](#)

Use an existing post as a template.



Comments

[Add comment](#)

No comments yet.

Categories

All Categories Most Used

Uncategorized

[+ Add New Category](#)

Can testing be automated?

NewPost-2016-11-11 06:18:48.782

Status: Published [Edit](#)

Body of post from 2016-11-11 06:18:48.782

[Update](#)



Body of post from 2016-11-11 06:18:48.782

NewPost-2016-11-11 06:18:48.782

Status: Published [Edit](#)

[Update](#)



Can testing be automated?

We can automate

- Repetitive tasks
- Parts of a test - like the setup part
- Tests with easy outcomes/outputs
- Partial tests

We cannot automate

- Thinking
- Improvising
- Strategizing
- Tests that have complex outputs

Can testing be automated?

Not everything should be scripted

Can testing be automated?

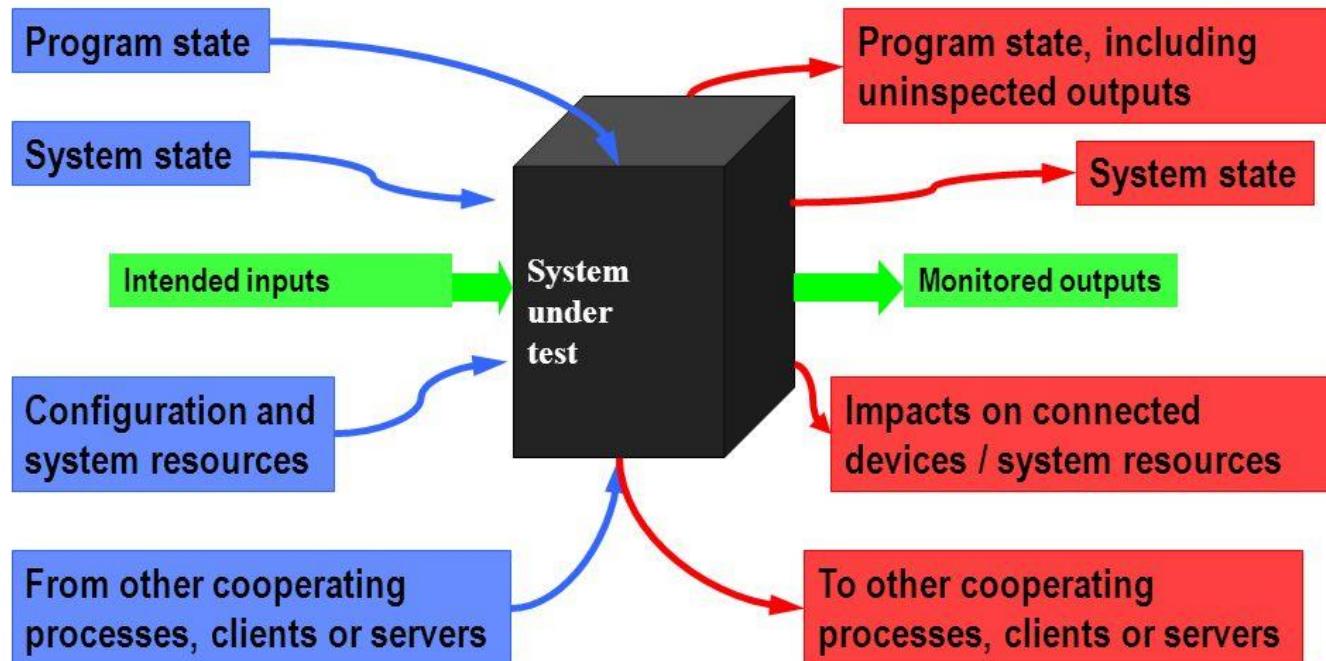


@AltomSays

When are you done testing?

A program can fail in many ways

Based on notes from Doug Hoffman



When are you done testing?

- Time's up
- Money's out
- “Good enough”
- “Bad enough”
- Someone tells you to stop

Are you to blame for bugs in production?

Reasons why a bug might slip to production

- We can never test everything
- A tester missed it
- A test manager/project manager said that functionality is not important
- A developer didn't fix it
- The company only tests in production

Who is to blame?

Are you to blame for bugs in production?



@AltomSays

Blame is irrelevant.

How to do better next time?

Are you to blame for bugs in production?



@AltomSays

Can you be a tester without technical skills?

Technical != coding all day

Can you be a tester without technical skills?



@AltomSays

Tech skills used while testing

- Networking related skills
- Security analysis skills
- Automation tools
- Domain specific skills (web & mobile technologies)
- Debugging & investigating skills
- Performance monitoring skills
- Data analysis skills

Testers and developers have complementary tech skills

Can you be a tester without technical skills?



@AltomSays

Is it true that testers never code?

Automation projects

- Mobile games automation

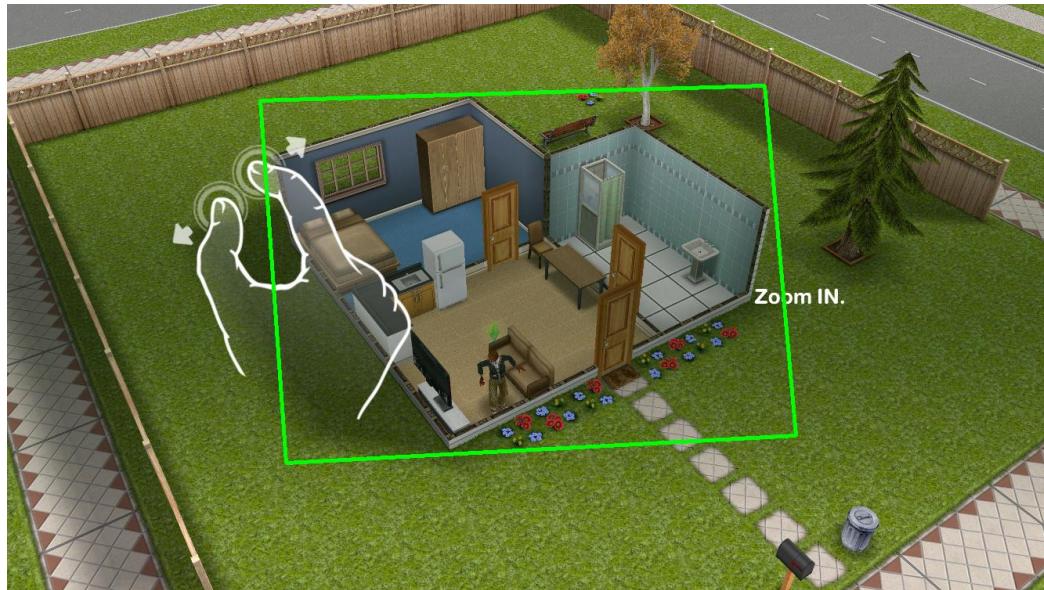


Resolution Agnostic, works rotated and stretched items



Is it true that testers never code?

Resolution Agnostic, works rotated and stretched items



Is it true that testers never code?

Resolution Agnostic, works rotated and stretched items



Testers don't code

Resolution Agnostic, works rotated and stretched items



Is it true that testers never code?



@AltomSays

Automation projects

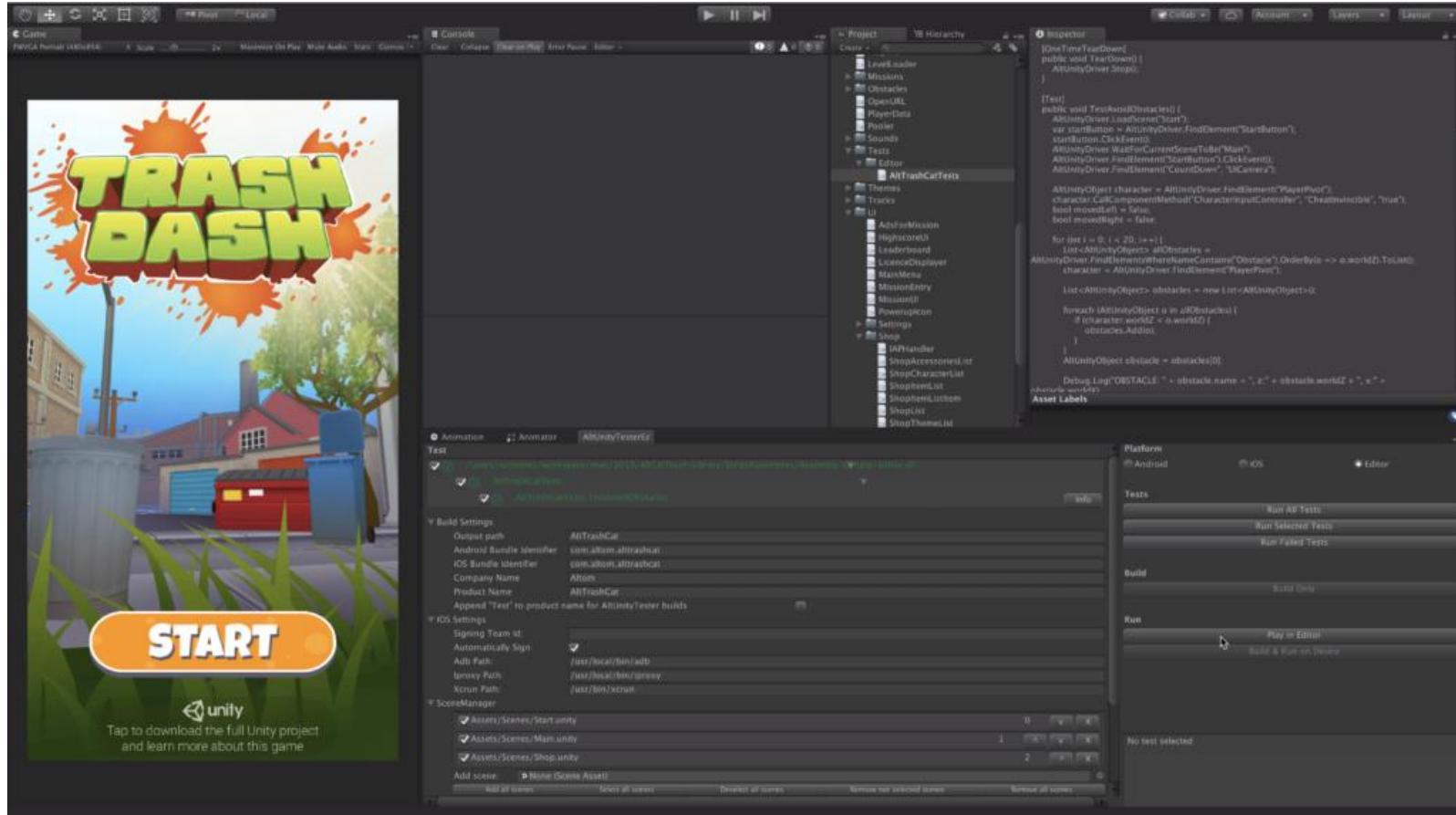
- Mobile games automation
- Web apps automation
- Automation using robots



Building testing tools

- AltWalker
- WebUiChecker
- AltUnityTester





Is it true that testers never code?

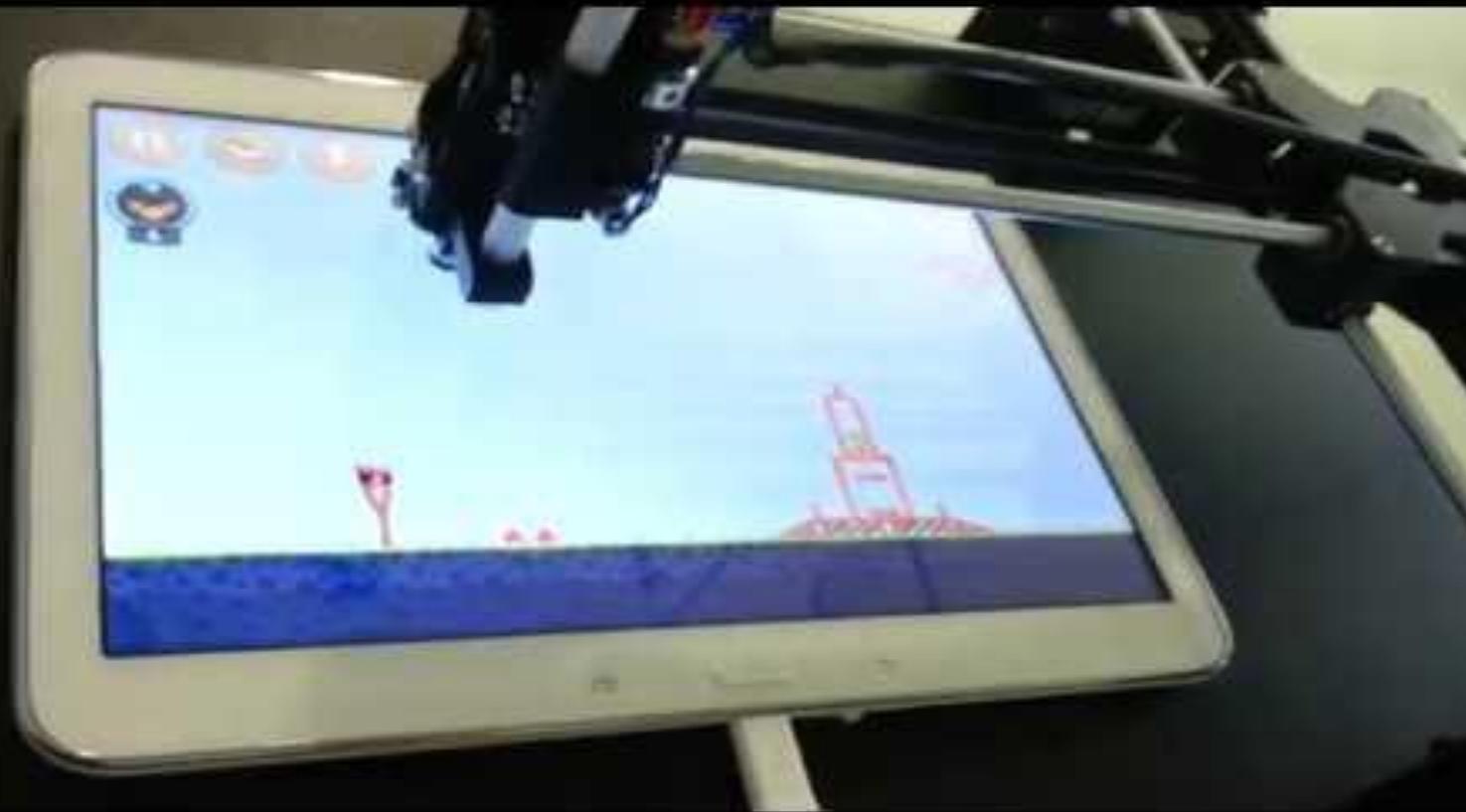
How do you get better at testing?

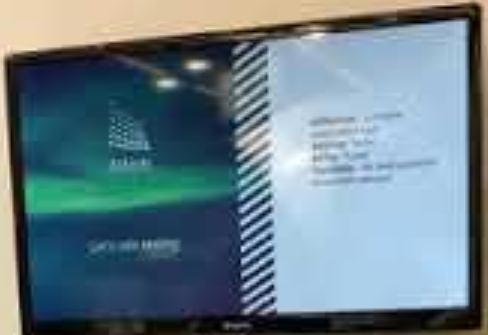
Improve your skills

- Feedback
 - Other colleagues (testers or not)
 - Community (local or online)
 - Tabara De Testare
- Courses
- Books
- Online resources like blogs
- Practice
- Open mindset



Testing is boring





@AltomSays



Testing is boring



Testing is boring

AltTap Demo

Testing is boring



@AltomSays

If it's boring, you're doing it
wrong!

Resources

- Altom open-source projects
 - <https://gitlab.com/altom>
- Android Monkey Runner
 - <https://developer.android.com/studio/test/monkeyrunner/>

Resources

Books

- Lessons Learned in Software Testing - Cem Kaner
- Perfect Software and Other Illusions about Testing - Gerald Weinberg
- Explore it! - Elisabeth Hendrickson

Communities:

- <http://www.meetup.com/Tabara-de-Testare-Cluj/>
- <https://www.ministryoftesting.com/>
- <http://testers.io> - Slack community

Altom BBST courses

- <https://altom.training>

Internship @ Altom

<https://altom.com/jobs/>

Thank you!

Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan Lecture 5: Levels of testing

Babeş-Bolyai University

Cluj-Napoca

2018-2019

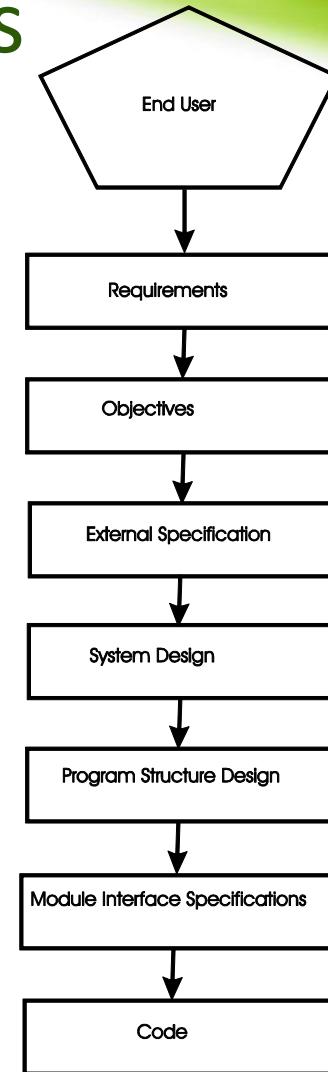


Outline

- Software development process
 - Software development process
 - Development and testing processes
- Levels of testing
 - Unit testing
 - Integration testing
 - Function testing
 - System testing
 - Acceptance testing
- Retesting vs regression testing
- Next lecture:
 - Correctness
- Questions

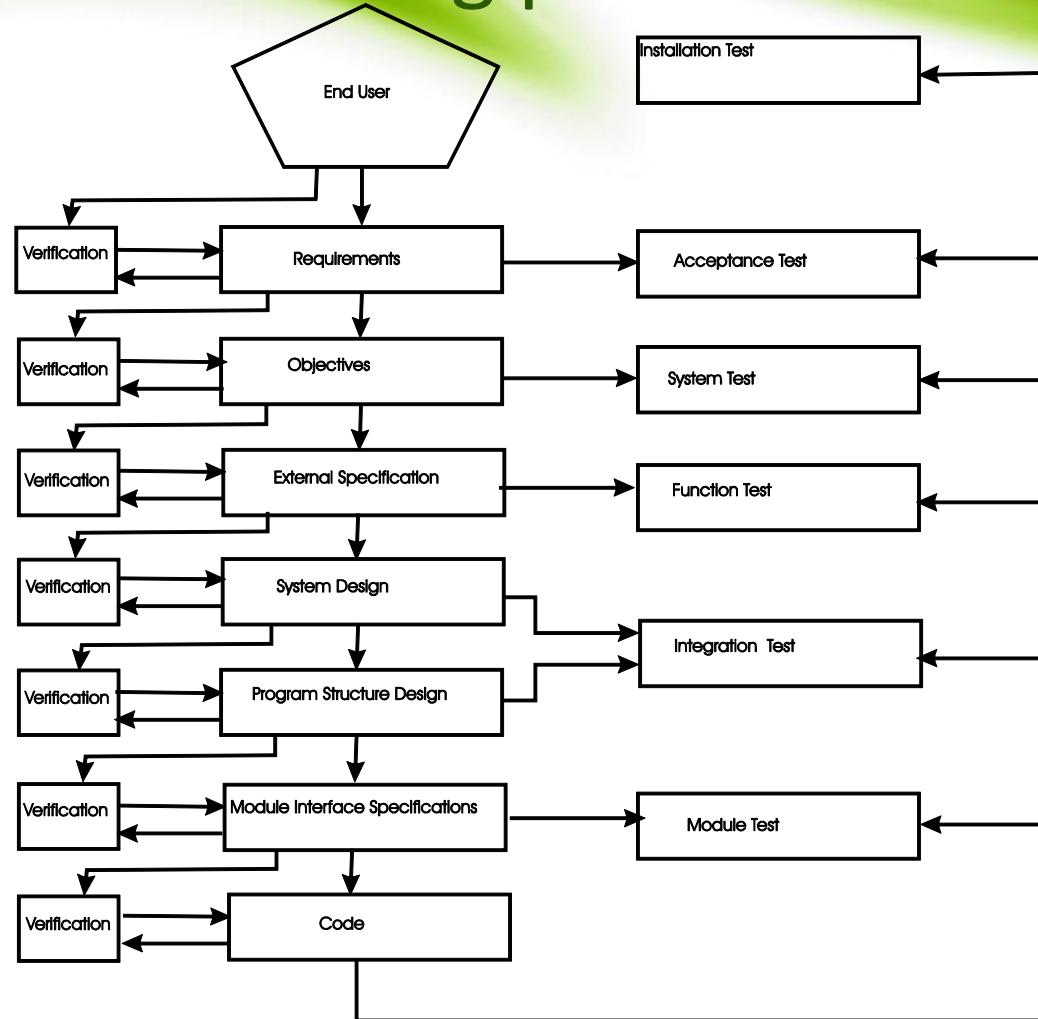
Software development process

- user's needs are translated into requirements
- requirements are translated into objectives
- objectives are translated into external specification
- system design
- program structure design
- module interface specification
- code



Development and testing processes

- Approaches to prevent errors:
 - More precision into the development process.
 - Introduction of a verification step at the end of each process.
 - Orient distinct testing processes toward distinct development processes.



Outline

- Software development process
 - Software development process
 - Development and testing processes
- Levels of testing
 - Unit testing
 - Integration testing
 - Function testing
 - System testing
 - Acceptance testing
- Retesting vs regression testing
- Next lecture:
 - Correctness
- Questions

Levels of testing

1. Unit testing

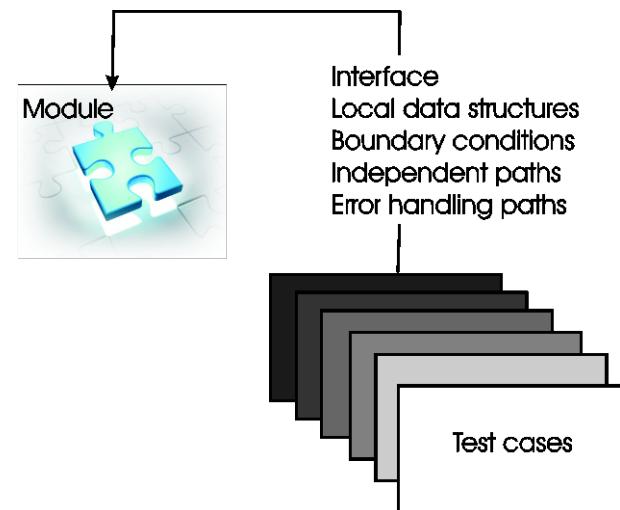
- Testing individual subprograms, subroutines, procedures, the smaller building blocks of the program.
- **Motivations:**
 - Managing the combined elements of testing.
 - Module testing eases the task of debugging.
 - Module testing introduces parallelism into the program testing process.
- **Points of view**
 - The manner in which test cases are designed.
 - The order in which modules should be tested and integrated.
- Advice about performing the test.
- References: [Mye04] (chapter 5),[NT05] (chapter 3).

Levels of testing

1. Unit testing (cont)

Test case design

- Information needed when designing test cases for a module:
 - specification of the module
 - the module's source code
- Test case design procedure for a module test is:
 - Analyze the logic of the module using white-box methods.
 - Applying black-box methods to the module's specification.

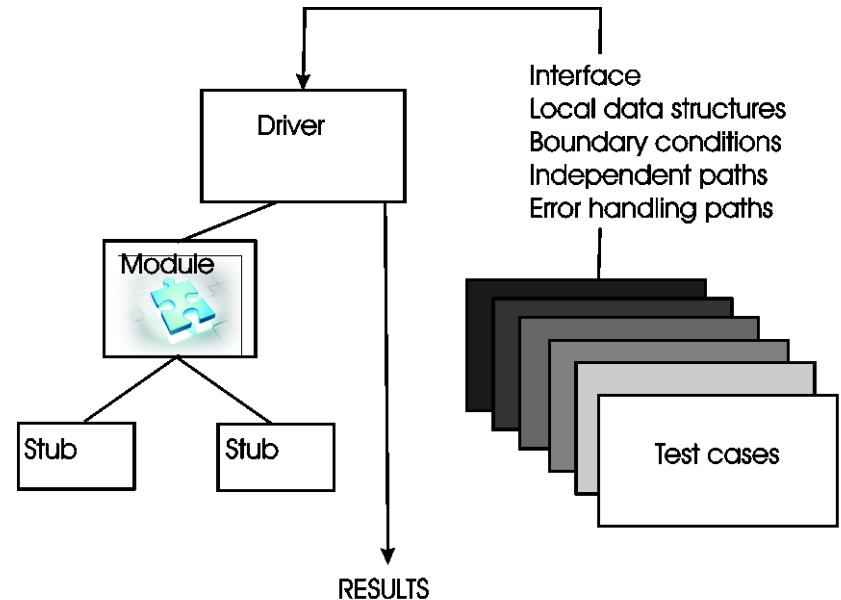


Levels of testing

1. Unit testing (cont)

Unit test procedures

- Unit test environment
 - **driver** - a “main program” that accepts test case data, passes such data to the component to be tested and prints relevant results;
 - **stub** - serve to replace modules that are subordinate the component to be tested.
 - uses the subordinate module’s interface
 - may do minimal data manipulation
 - prints verification of entry
 - returns control to the module undergoing testing.



Levels of testing

2. Integration testing

- Constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing.
- **Importance of integration testing:**
 - Different modules are generally created by groups of different developers.
 - Unit testing of individual modules is carried out in a controlled environment by using test drivers and stubs.
 - Some modules are more error prone than other modules.
- Objectives:
 - putting the modules together in an incremental manner
 - ensuring that the additional modules work as expected without disturbing the functionalities of the modules already put together.
- Reference: [NT05] (chapter 7).

Levels of testing

2. Integration testing (cont)

Techniques [Fre10]

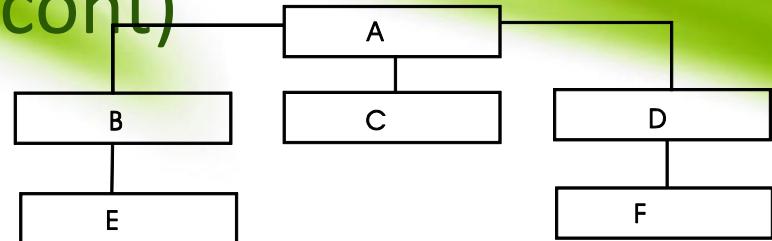
- Big-bang
- Incremental
 - Top-down.
 - Bottom-up.
- Sandwich.

Levels of testing

2. Integration testing (cont)

Big-bang testing

- Big-bang procedures:
 - Module test for each individual unit;
 - A driver module;
 - Several stub modules.
 - The modules are combined to form the program.
- **Observations**
 - more work for big-bang
 - mismatching interfaces/incorrect assumptions among modules - detected earlier with incremental
 - Debugging easier - incremental
 - Big-bang - appears to use less machine time
 - parallel activities – opportunity for big-bang

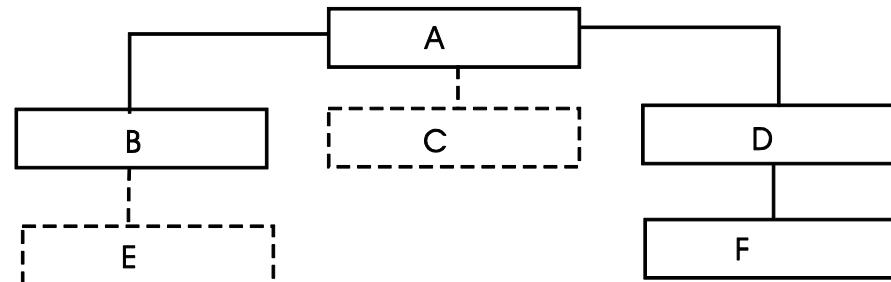


Levels of testing

2. Integration testing (cont)

Top-down incremental testing

- Top-down integration manner:
 - Depth-first integration;
 - Breadth-first integration.
- Top-down integration process:
 - main control module = driver;
 - stubs=substituted for all components directly subordinate;
 - subordinates stub ← actual components;
 - tests are conducted as each component is integrated;
 - on completion of each set of tests, another stub ← real component;
 - regression testing may be conducted.

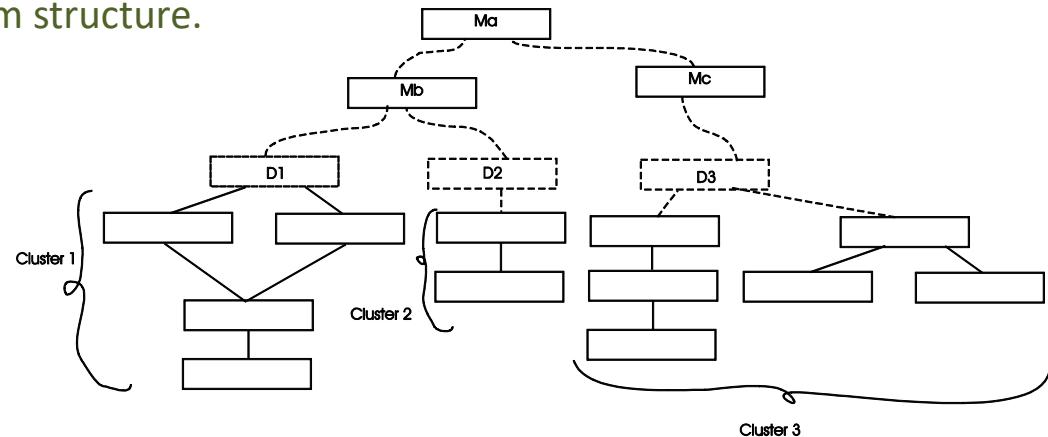


Levels of testing

2. Integration testing

Bottom-up incremental testing

- Bottom-up integration process:
 - low-levels components are combined into clusters;
 - a driver is written to coordinate test case input and output;
 - the cluster is tested;
 - drivers are removed and clusters are combined moving upward in the program structure.

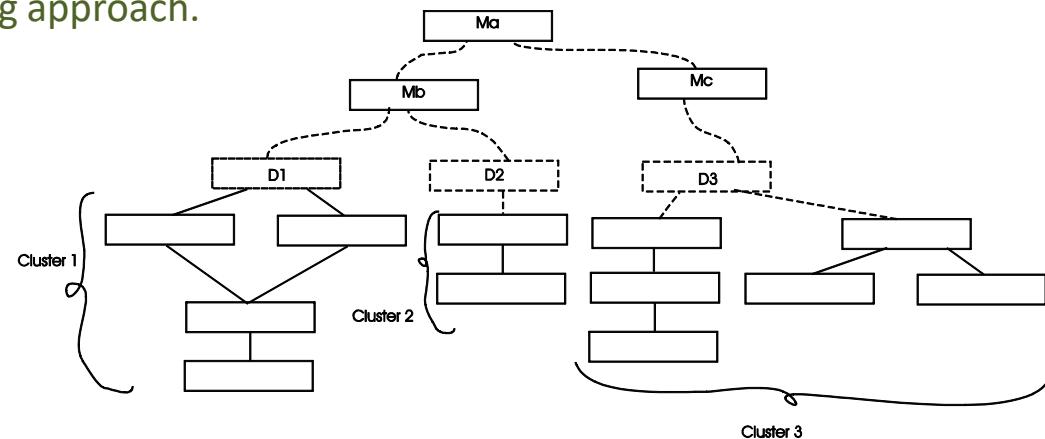


Levels of testing

2. Integration testing

Sandwich testing

- Sandwich procedures:
 - mix of the top-down and bottom-up approaches;
 - layers of a hierarchical system:
 - bottom-layer – using bottom-up module integration;
 - top-layer - using top-down approach integration;
 - middle-layer - big-bang approach.



Levels of testing

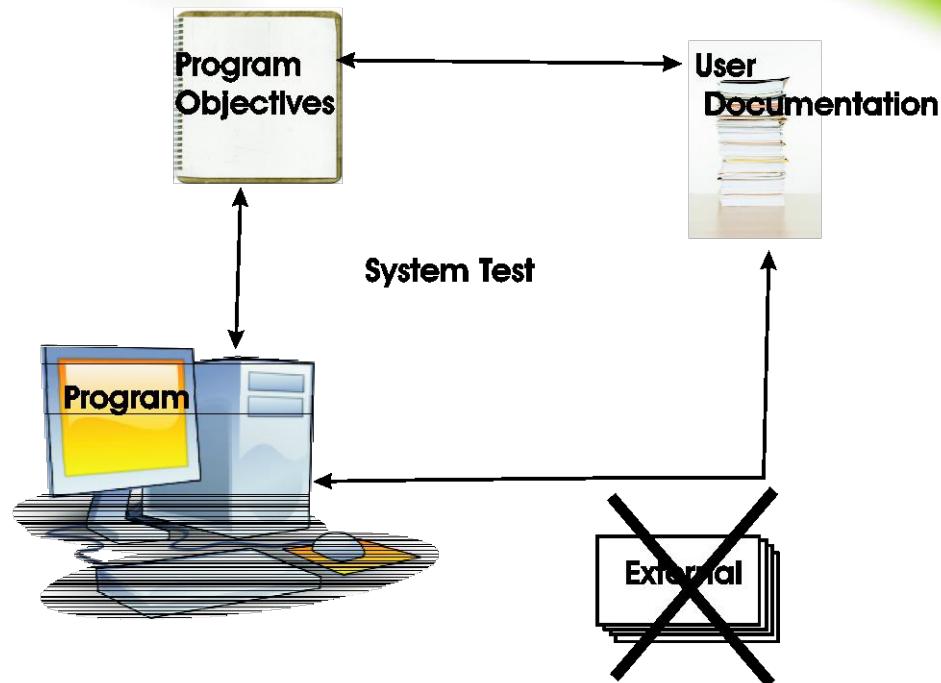
3. Function testing

- testing requirements described in the external specification of the system;
 - a process of attempting to find discrepancies between the program and the external specification.
 - a black-box activity
 - uses system specification
-
- References: [Mye04] (chapter 6), [NT05] (chapter 9), [PY08] (chapter 10).

Levels of testing

4. System testing

- compare the program - original objectives.
- Use external specification? no, may appear defects during the process of translating the objectives in external specifications;
- Use objectives documents? no, do not contain exact description of the external interfaces of the program;
- Use program's user documentation
- References:[Mye04] (Chapter 6), [NT05] (chapter 8), [PY08] (chapter 22).



Levels of testing

4. System testing (cont)

- the objectives does not offer information about the functionality of the system (interfaces of the modules being tested)
- there is no methodology for created test cases in system testing???
- the process of creating test cases: use imagination, creativity and experience

Levels of testing

4. System testing (cont)

System testing types

- In [Mye04] (Chapter. 6) there are 15 types of system testing:
 - Facility testing
 - Volume testing
 - Usability testing
 - Recovery testing
 - **Security testing – Details in Lecture 11**
 - **Stress testing**
 - **Performance testing – Details in Lecture 7 – IT firm EVOZONE – Lecture invitation**
 - Storage testing
 - Configuration testing
 - Compatibility testing
 - Instability testing
 - Reliability testing
 - Serviceability testing
 - Documentation testing
 - Procedure testing

Levels of testing

5. Acceptance testing

- a process of comparing the program to its initial requirements and the current needs of its end user;
 - not the responsibility of the development organization;
 - the customer first performs an acceptance test to determine whether the product satisfies its needs.
-
- References: [NT05] (chapter 14), [PY08] (chapter 22).

Outline

- Software development process
 - Software development process
 - Development and testing processes
- Levels of testing
 - Unit testing
 - Integration testing
 - Function testing
 - System testing
 - Acceptance testing
- Retesting vs regression testing
- Next lecture:
 - Correctness
- Questions

Testing level vs. Testing type

Testing level

- set of activities that are associated to a phase of the software development product

Testing type

- the mean by which an objective of a testing level is achieved

Examples

- Testing a function – unit level or integration level by bbt(domain)/wbt
- Testing of a non-functional characteristic – at system level by performance testing or usability testing
- Testing after eliminating a bug – at any level after debugging/corrected the bug by applying retesting, confirmation testing
- Testing relating to eliminating a bug – at any level by regression testing to verify if the elimination of the bug doesn't have side-effects

Retesting (confirmation testing)

- Retesting
 - Execution of the test cases that revealed a bug that was reported
 - Goal – confirmation that the bug was eliminated
- Test cases – are the same with those already executed

Regression testing

- Regression testing - **the re-execution of some subsets of tests that have already been conducted** to ensure that changes have not propagated unintended side effects.
- Regression test suits - classes of test cases:
 - Tests to exercise all software functions.
 - Tests that focus on software functions that are likely to be affected by the change.
 - Tests that focus on the software components that have been changed.
- Reference: [PY08] (chapter 22).
- Regression testing – **new test cases** (Cem Kaner) [BBST]

Regression testing

Research activity

- Bonus points = max 300 XP
- only if you (will) have 300 XP for Lab activities
- Topic: Regression testing : TCP or TCS or TCM
 - Test Case Prioritization
 - Test Case Selection
 - Test Case Minimization

- 2 members / team
- deliverables on time!
- paper submitted to journal for review

W5 25 march → Regression testing ; TCP or TCS or TCM
Team up + Select topic

W6 1 april → Literature Review

W7 8 → Existing Approaches + Case study

W8 15 → Approaches + Case study

W9 22 →

Vacation 29 apr. - 5 may

W10 6 may → Your approach + Implementation + Case study

W11 13 may →

W12 20 may → 24 may Submitting the paper to journal

- Rule: First 9 emails will be considered!
- Deadline for sign up: 29 April 2019, h:20⁰⁰

Provided	Deliverables
	<ul style="list-style-type: none">• Team members• Selected topic• Send email: avraman@cs.ulb.ac.be• Search others 7 papers• Report on the 10 papers<ul style="list-style-type: none">• description TCP/TCS/TCM• similarities - pro/cons• 3 pages long
• 3 papers <ul style="list-style-type: none">• To start the literature review	<ul style="list-style-type: none">• Search others 2 repositories• Report on 1-3 Case study/studies to be used.<ul style="list-style-type: none">• 3 diff. methods used in the approaches• 3 pages long
• 1 repository for repository	<ul style="list-style-type: none">• Source code/implementation of your approach• Report on your approach<ul style="list-style-type: none">• description of your method• description of your case study/studies• comparisons with other approaches
• feedback on your paper	<ul style="list-style-type: none">• paper submitted to a journal.

Surprise!





Having fun learning about testing

Easter eggs – in testing

- For students that participated in Lecture 05.
- 25 XP for each student

- Each student presents 1 real example of Easter egg in testing.
- Present 1 page information in Lecture 6 in printed format.
 - Definition/description
 - Example
 - Interesting fact(s)

Surprise!



Having fun learning about testing During Lecture 5 (second hour)

- Software testing – is a search for information.
- Testing strategy is the guiding framework for deciding what tests (what test techniques) are best suited to your product.
- Context and information objectives are (or should be) the drivers of any testing strategy.
- Different objectives require different testing tools and strategies. And will yield different tests, test documentation and test results.

1. Guerilla testing

2. Dumb monkey testing

3. Smoke testing 4. Bug bashes (in testing)

5. Scenario testing

6. Tour testing

- <https://www.timeanddate.com/timer/>
- Debriefing
 - Learning?
 - Individual/Team

1 page information (20 minutes)

And

3 minutes presentation

(6 *3 minutes=18 minutes)

- Definition/description
- Characteristics (5 to 10 points)
- Levels of testing
- Example = simple + real world application
- Interesting fact(s)

- Creating/presenting the Poster
 - 25 XP for each member of the team
- Questionnaire about the activity
 - 25 XP

Questions

- Thank You For Your Attention!

References

- [Pat05] R. Patton. Software Testing. Sams Publishing, 2005.
- [PY08] M. Pezzand and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2008.
- [Mye04] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004
- [You79] E. Yourdon, Structured Walkthroughs, Prentice-Hall, Englewood Cliffs, NJ, 1979
- [NT05] K. Naik and P. Tripathy. *Software Testing and Quality Assurance*. Wiley Publishing, 2005.
- [CB03] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010
- [BBST] BBST Testing course, <http://testingeducation.org/BBST/>

Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan

Babeş-Bolyai University

Cluj-Napoca

2018-2019

Lecture 6: Correctness





Having fun learning about testing

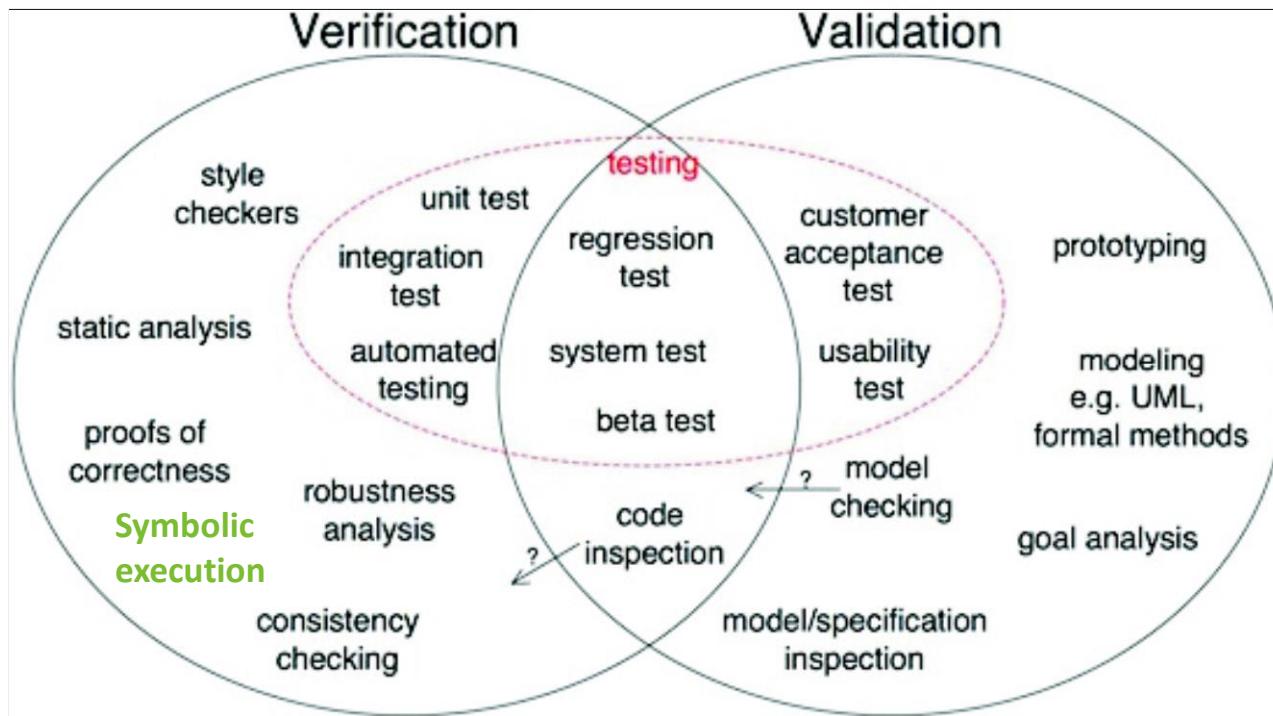
Easter eggs – in testing

- For students that participated in Lecture 05.
- 25 XP for each student

- Each student presents 1 real example of Easter egg in testing.
- Present 1 page information in Lecture 6 in printed format.
 - Definition/description
 - Example
 - Interesting fact(s)

Sales paradigm - SSVV

- Motivate the STUDENT - what you will learn!



- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
 - **EVOZON** Presentation, topic: **Test automation**
 - **When:** Friday, April 12, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Program verification methods - Correctness

- Lecture 1 - Verification and Validation
 - Verification/Validation
 - reviews products to ensure their quality → correctness
 - static and dynamic analysis techniques
 - A **correct program** is one that does exactly what it is intended to do, no more and no less.
 - A formally correct program is one whose correctness can be proved mathematically.
 - This requires a language for specifying precisely what the program is intended to do.
 - Specification languages are based in mathematical logic.
 - Until recently, correctness has been an academic exercise. – Now it is a key element of critical software systems.
- **Program verification - correctness**
 1. proof-based, computer-assisted, program-verification approach, mainly used for programs which we expect to terminate and produce a result
 2. model-based, automatic, property-verification approach, mainly used for concurrent, reactive systems (originally used in a post-development stage) - model checking (Lecture 8, Lecture 9)
 3. Developing correct algorithms from specification (Carroll Morgan, “Programming from Specification)
 - Correctness-by-Construction.
Originally intended as a mere means of programming algorithms that are correct by construction - Dijkstra (1968), Hoare (1971),
the approach found its way into commercial development processes of complex systems - Hall (2002), Hall and Chapman (2002)
2012, The Correctness-by-Construction Approach to Programming, Authors: Kourie, Derrick G., Watson, Bruce W.
2015, Experience with correctness-by-construction, B.W. Watson a, D.G. Kourie b, L. Cleophas b,*
2016, Correctness-by-Construction and Post-hoc Verification: Friends or Foes?, Maurice H. ter Beek1(B) , Reiner Hahnle2, and Ina Schaefer3
- **Correctness Tools**
 - Theorem provers (PVS), Modeling languages (UML and OCL), Specification languages (JML), Programming language support (Eiffel, Java, Spark/Ada), Specification Methodology (Design by contract)
- **Methods for proving program correctness**
 - Floyd’s Method - Inductive assertions
 - Hoare - Semantics of Hoare triples
 - Dijkstra’s Language- Guarded commands, Nondeterminacy and Formal Derivation of Programs

Surprise!

Grading Gamifying Education

Quizzes (Heroic Quests)

300 XP

Today - Quiz1

6 questions * 25 XP = 150 XP

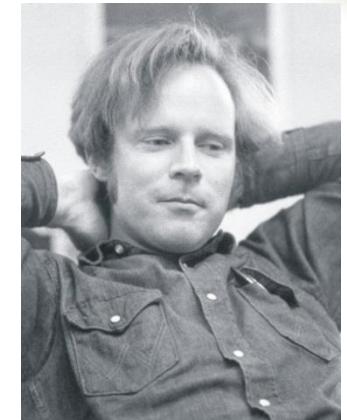
6-10 minutes.

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
 - **EVOZON** Presentation, topic: **Test automation**
 - **When:** Friday, April 12, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Floyd's Method - Inductive assertions [Flo67]

- **Applicability**
 - Partial correctness of the program
 - Termination of the program
 - Total correctness = Partial correctness + Termination of the program
- **Uses**
 - The condition satisfied by the initial values of the program.
 - The condition to be satisfied by the output of the program.
 - Source code of the program.
- **Method:**
 - Cut the loops
 - Find an appropriate set of inductive assertions.
 - Construct the verification/termination conditions.
- **Theorem:** If all verification conditions are true, then the program is partially correct, i.e., whenever it terminates the result is correct.
- **Remark.** The method is useful when it is combined with termination.



Robert W Floyd
(June 8, 1936 - September 25, 2001)

Floyd's Method - Inductive assertions [Flo67]

- Partial correctness - steps

- Cutting points are chosen inside the algorithm
 - 1 point at the beginning of the algorithm, 1 point at the end;
 - At least 1 point for each *loop* statement
- For each cutting point an assertion (invariant predicate) is chosen.
 - Entry point - $\varphi(X)$;
 - Ending point - $\psi(X, Z)$.
- Construction of the verification conditions
 - Path from i to j - α ;
 - P_i and P_j are assertions in i and j ;
 - $R_\alpha(X, Y)$ - predicate that gives the condition for path α ;
 - $r_\alpha(X, Y)$ - function that gives the transformations of the variables Y from path α ;
 - $\forall X \forall Y (P_i(X, Y) \wedge R_\alpha(X, Y) \rightarrow P_j(X, r_\alpha(X, Y)))$.
- Theorem: If all the verification conditions are true then P is partial correct.



Robert W Floyd
(June 8, 1936 - September 25, 2001)

Floyd's Method - Inductive assertions [Flo67]

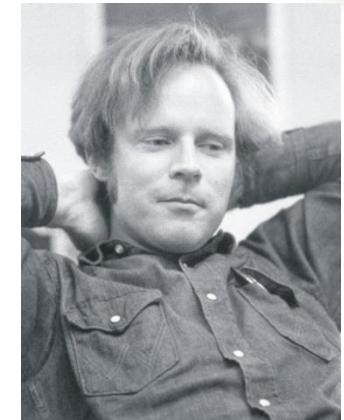
- Partial correctness - example

- Algorithm for $z = x^y$

```
z := 1; u := x; v := y;  
While (v > 0) execute  
  If (v is even)  
    then u := u * u; v := v/2;  
  else v := v - 1; z := z * u;  
  endIf  
endWhile  
endAlg;
```

A: $\varphi(X) ::= (v > 0 \wedge (y \geq 0))$
B: $\eta(X, Y) ::= z * u^v = x^y$

C: $\psi(X, Z) ::= z = x^y$

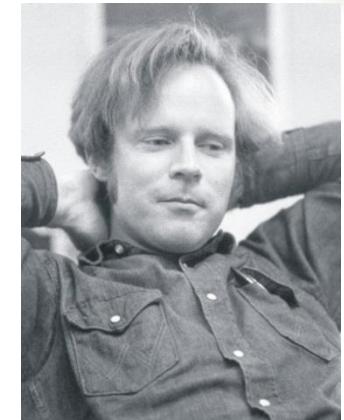


Robert W Floyd
(June 8, 1936 - September 25, 2001)

Floyd's Method - Inductive assertions [Flo67]

- Termination - steps

- Cut the loops and find “good” inductive assertions.
- Choose a well-formed set M (i.e., an ordered set without infinite strictly decreasing sequences)
- To demonstrate that some termination conditions hold: passing from one cutting point to another the values of some functions in the well-ordered set decrease.
- In point i a function is chosen $u_i : D_X \times D_Y \rightarrow M$ and the termination condition on α is:
$$\forall X \forall Y (\varphi(X) \wedge R_\alpha(X, Y) \rightarrow (u_i(X, Y) > u_j(X, r_\alpha(X, Y)))).$$
- **Remark.** If partial correctness was demonstrated then the termination condition can be:
$$\forall X \forall Y (P_i(X) \wedge R_\alpha(X, Y) \rightarrow (u_i(X, Y) > u_j(X, r_\alpha(X, Y)))).$$
- Theorem: If all the termination conditions hold then the program P terminates.



Robert W Floyd
(June 8, 1936 - September 25, 2001)

Floyd's Method - Inductive assertions [Flo67]

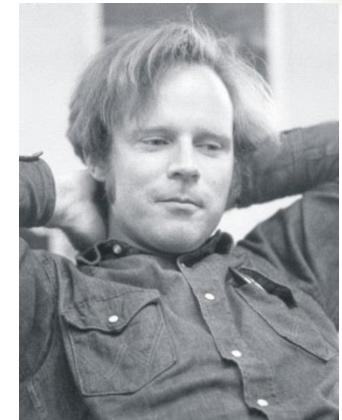
- Termination - example

- Algorithm for $z = x^y$

```
z := 1; u := x; v := y;  
While (v > 0) execute  
  If (v is even)  
    then u := u * u; v := v/2;  
  else v := v - 1; z := z * u;  
  endIf  
endWhile  
endAlg;
```

A: $\varphi(X) ::= (v > 0 \wedge (y \geq 0))$
B: $\eta(X, Y) ::= z * u^v = x^y$

C: $\psi(X, Z) ::= z = x^y$



Robert W Floyd
(June 8, 1936 - September 25, 2001)

Surprise!

Floyd's Method

- Inductive assertions, Partial correctness, Termination

3-5 minutes

Formative Assessment

Anonymous voting

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
 - **EVOZON** Presentation, topic: **Test automation**
 - **When:** Friday, April 12, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Hoare triples [Hoa69]

- The meaning of a statement is described by a triple

- $\{\varphi\} P \{\psi\}$, where φ is called the precondition and ψ is called the postcondition.

$\{P\} S \{Q\}$

"when started in a state satisfying P , any terminating execution of S ends in a state satisfying Q "

- If P does not terminate, we make no guarantees.

- Partial correctness

- $\models_{par} \{\varphi\} P \{\psi\}$
- only if P actually terminates.

- Total correctness

- $\models_{tot} \{\varphi\} P \{\psi\}$
- the program P is guaranteed to terminate.



- The Grand Verification Challenge Hoare 2003
- Develop a compiler which verifies that the program is correct
- <https://vimeo.com/39256698>

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)



An Advanced Study Institute of the
NATO Security Through Science Committee
and
the Institut für Informatik,
Technische Universität München, Germany,

on

Software System Reliability and Security

August 1 to August 13 2006

M. Broy (director)
O. Kupferman (director)
C.A.R. Hoare (co-director)
A. Pnueli (co-director)

Katharina Spies (secretary)

The Summer School is also substantially supported by
the DAAD under the program "Deutsche Sommerakademie 2006",
and the town and the county of Marktberdorf



Hoare triples [Hoa69]

- Partial correctness

Rules

- Assignment
- Sequencing
- Conditional
- Loop



Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Partial correctness

Assignment

General Form: for any expression E

- $\{P\} X := E \{Q\}$ provided $[P \Rightarrow \langle X \leftarrow E \rangle (Q)]$

- Consider the triple $\{P\} X := Y + 2 \{Q\}$
 - Given predicate Q , for what predicate P does this hold?
 - for any P such that $[P \Rightarrow \langle X \leftarrow Y + 2 \rangle (Q)]$
- Examples
 - $\{P_0\} X := Y + 2 \{X \leq Y + 2\}$
 $P_0 \equiv \text{true}$
 - $\{P_1\} X := Y + 2 \{X < 0\}$
 $P_1 \equiv (Y + 2 < 0)$
 - $\{P_2\} X := Y + 2 \{Y < 0\}$
 $P_2 \equiv (Y < 0)$
 - $\{P_3\} X := X + 2 \{X \text{ is even}\}$
 $P_3 \equiv (X \text{ is even})$



Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

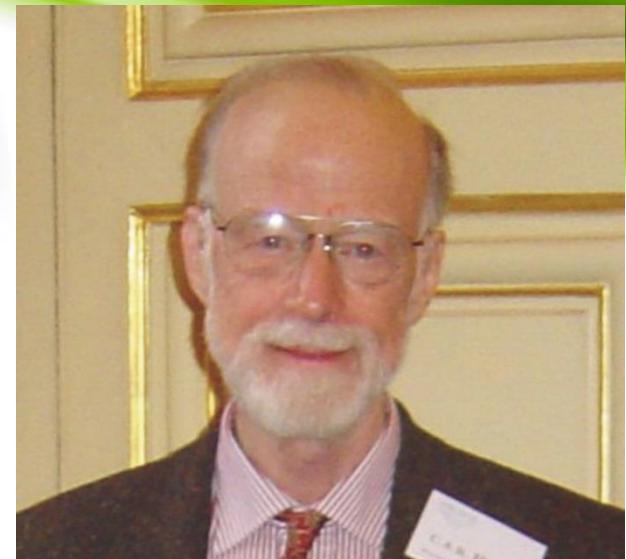
- Partial correctness

Sequencing

- We can conclude
 $\{P\} S; T \{Q\}$
if we can find a predicate R such that
 $\{P\} S\{R\}$ and $\{R\} T\{Q\}$

Examples

- $\{P_0\} X := 2 * X; X := X + 1 \{X > 0\}$
 $P_0 \equiv (2 * X + 1 > 0)$
- $\{P_1\} X := Y; Y := 3 \{X + Y < 5\}$
 $P_1 \equiv (Y + 3 < 5)$



Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Partial correctness

Conditional

- We can conclude

$\{P\} \text{ IF } (C) \text{ THEN } S \text{ ELSE } T \text{ END}\{Q\}$

provided we can show

$\{P \wedge C\} S\{Q\}$ and $\{P \wedge \neg C\} T\{Q\}$

- Examples

- $\{?\} \{(x > y) \Rightarrow Q_0\} \wedge \{(x \leq y) \Rightarrow Q_1\}$
 $\text{IF } (x > y) \text{ THEN } Q_0 : \{(m|x - y) \wedge (m|y)\}$

$x := x - y$

$\text{ELSE } Q_1 : \{(m|x) \wedge (m|y - x)\}$

$y := y - x$

END

$Q : \{(m|x) \wedge (m|y)\}$

- So our final proof obligations are

$[(x > y) \Rightarrow (m|x - y) \wedge (m|y)] \text{ and }$
 $[(x \leq y) \Rightarrow (m|x) \wedge (m|y - x)]$



Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Partial correctness

Loop

- How can we conclude

$\{P\} \text{ WHILE } (G) \text{ DO } S \text{ END } \{Q\}$

At the end of the loop (assuming it terminates), we know $\neg G$

But in general we don't know how often S is executed...

- Suppose we have a predicate J that is preserved by S

$\{J\}S\{J\}$ such a J is called a loop invariant

Then, at the end of the loop, we can conclude

$J \wedge \neg G$

To establish the postcondition, we need J such that

$[J \wedge \neg G \Rightarrow Q]$

- We can conclude

$\{P\} \text{ WHILE } (G) \text{ DO } S \text{ END } \{Q\}$

provided we can find a loop invariant J such that

$[P \Rightarrow J]$
 $[J \wedge \neg G \Rightarrow Q]$
 $\{G \wedge J\}S\{J\}$

J holds at loop entry
 J establishes Q at loop exit
 J is preserved by each iteration



Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

- Exponentiation using multiplication

$\{(A > 0) \wedge (B \geq 0)\} S \{R = A^B\}$

$\{(A > 0) \wedge (B \geq 0)\}$

$R := ?; b := 0 R := 1$

$\text{WHILE } (b \neq B) \text{ DO } J : R := A^b$

$R := ?; R := R * A;$

$b := b + 1$

END

$\{R = A^B\}$

Hoare triples [Hoa69]

- The meaning of a statement is described by a triple
 - $\{\varphi\} P \{\psi\}$, where φ is called the precondition and ψ is called the postcondition.

$\{P\} S \{Q\}$

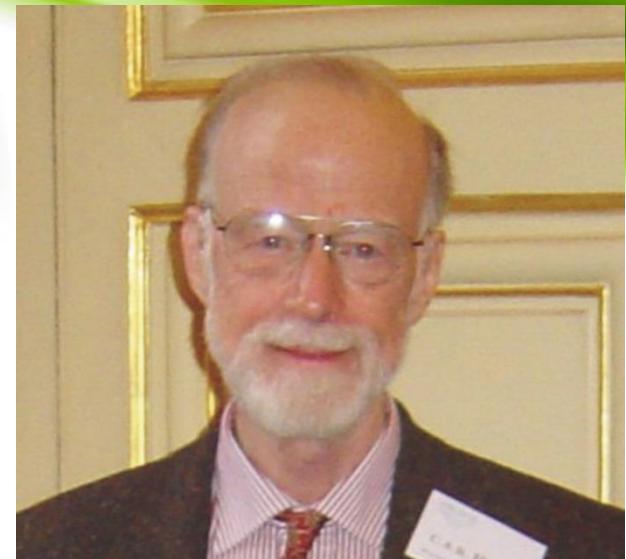
“when started in a state satisfying P , any terminating execution of S ends in a state satisfying Q ”

- If P does not terminate, we make no guarantees.

- Partial correctness
 - $\models_{par} \{\varphi\} P \{\psi\}$
 - only if P actually terminates.
- Total correctness
 - $\models_{tot} \{\varphi\} P \{\psi\}$
 - the program P is guaranteed to terminate.

- The “total correctness” interpretation also requires termination

“when started in a state satisfying P , any execution of S must terminate in a state satisfying Q ”



Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Termination

Rules

- Assignment
 - Sequencing
 - Conditional
 - Loop
- Assignment
 $\{P\} X := E \{Q\}$ provided $[P \Rightarrow (X \leftarrow E)(Q)]$
- Sequencing
 $\{P\} S; T \{Q\}$ provided
 $\{P\} S \{R\}$ and $\{R\} T \{Q\}$ for some R
- Conditional
 $\{P\} IF (G) THEN S ELSE T END \{Q\}$ provided
 $\{P \wedge G\} S \{Q\}$ and $\{P \wedge \neg G\} T \{Q\}$
- Note: Same as the rules for partial correctness!



Charles Antony Richard Hoare

(11 January 1934, Colombo, Sri Lanka)

- Total correctness rule for loops
- Consider
 $\{P\} WHILE (G) DO S END \{Q\}$
- How do we show that the loop terminates?
- One method
find an integer expression V such that
the value of V is nonnegative (that is $V \geq 0$), and
the value of V (strictly) decreases in every iteration that is,
 $\{V = K\} S \{V < K\}$
- Such an expression is called a "loop variant"

Hoare triples [Hoa69]

- Termination

Exponentiation using multiplication

- $\{(A > 0) \wedge (B \geq 0)\} S \{R = A^B\}$
- Recall loop invariant $J : R = A^b \wedge (B \geq b)$;

$\{(A > 0) \wedge (B \geq 0)\}$

$R := 1; b := 0$

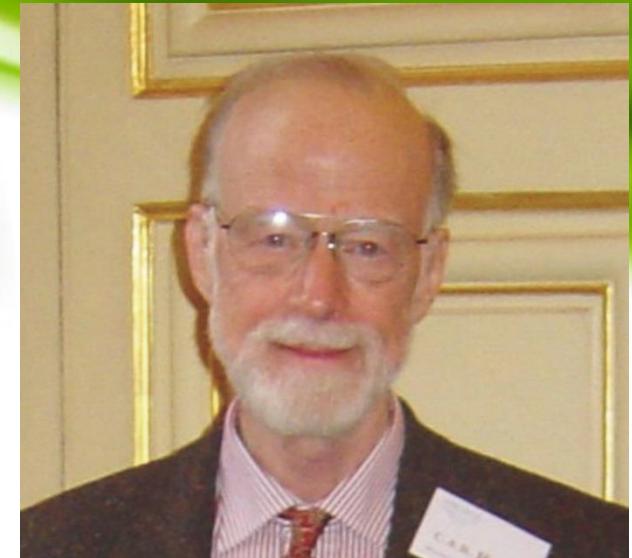
WHILE $(b \neq B)$ DO $J : R = A^b \wedge (B \geq b)$;

$R := R * A;$

$b := b + 1$

END

$\{R = A^B\}$



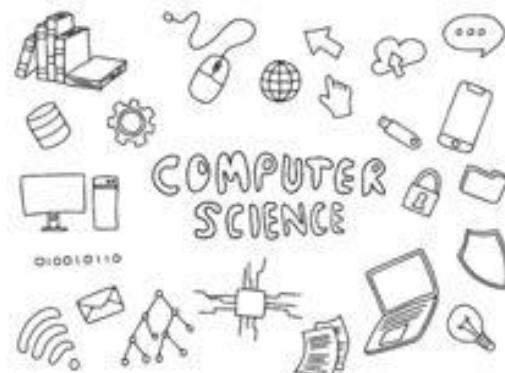
Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Surprise!

Hoare triples

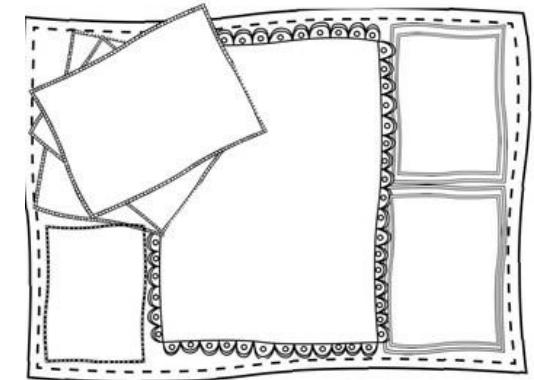
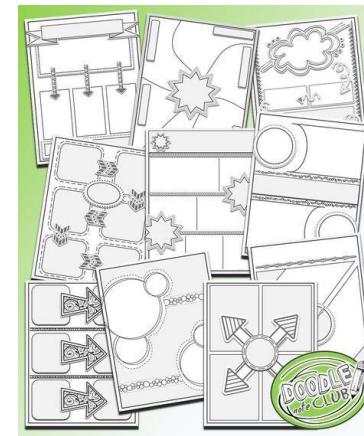
Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness

3-5 minutes



Formative Assessment

Doodle map



Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
 - **EVOZON** Presentation, topic: **Test automation**
 - **When:** Friday, April 12, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Edsger Wybe Dijkstra [Dij75]

- Guarded command

- “guarded command” - a statement list prefixed by a boolean expression: only when this boolean expression is initially true, is the statement list eligible for execution
- $\langle \text{guarded command} \rangle ::= \langle \text{guard} \rangle \rightarrow \langle \text{guarded list} \rangle$
- $\langle \text{guard} \rangle ::= \langle \text{boolean expression} \rangle$
- $\langle \text{guarded list} \rangle ::= \langle \text{statement} \rangle \{ ; \langle \text{statement} \rangle \}$
- $\langle \text{guarded command set} \rangle ::=$
 $\quad \langle \text{guarded command} \rangle \{ \square \langle \text{guarded command} \rangle \}$
- $\langle \text{alternative construct} \rangle ::= \mathbf{if} \langle \text{guarded command set} \rangle \mathbf{fi}$
- $\langle \text{repetitive construct} \rangle ::= \mathbf{do} \langle \text{guarded command set} \rangle \mathbf{do}$
- $\langle \text{statement} \rangle ::= \langle \text{alternative construct} \rangle \mid$
 $\quad \langle \text{repetitive construct} \rangle \mid \text{“other statements”}$



Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- Nondeterminacy

- Example 1

if $x \geq y \rightarrow m := x$

$\square y \geq x \rightarrow m := y$

fi

- Example 2 - compute k s.t. for fixed value n and fixed function $f(i)$ (*defined for $0 \leq i < n$*), k will eventually satisfy $0 \leq k < n$ and $(\forall i : 0 \leq i < n : f(k) \geq f(i))$.

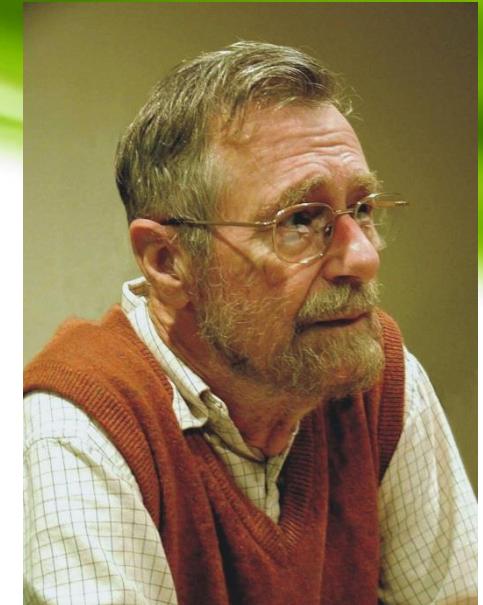
$k := 0; j := 1;$

do $j \neq n \rightarrow$ **if** $f(j) \leq f(k) \rightarrow j := j + 1$

$\square f(j) \geq f(k) \rightarrow k := j; j := j + 1$

fi

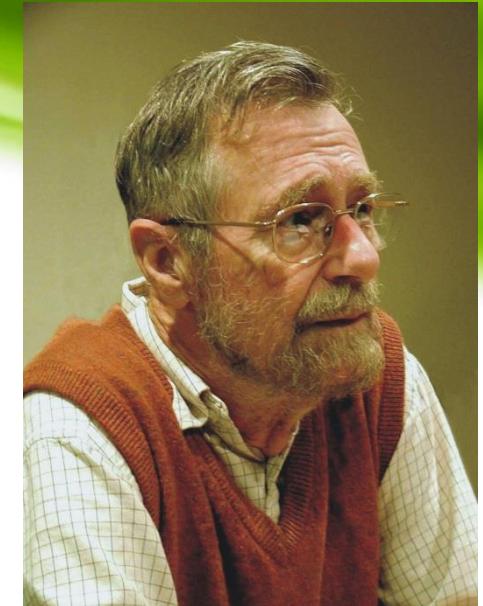
od



Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- Weakest pre-conditions
- Hoare - introduced sufficient pre-conditions such that the mechanism will not produce the wrong result but may fail to terminate.
- Dijkstra - introduced necessary and sufficient pre-conditions such that the mechanism are guaranteed to produce the right result.
 = weakest pre-conditions
- $\text{wp}(S, R)$, where S denotes a statement list, R some condition on the state of the system.
- wp - called a “ predicate transformer ” - because it associates a pre-condition to any post-condition R.



Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- Properties of wp

① Law of the Excluded Miracle

For any S, for all states: $wp(S, F) = F$

② For any S and any two post-conditions, such that for all states
 $P \Rightarrow Q$, for all states:

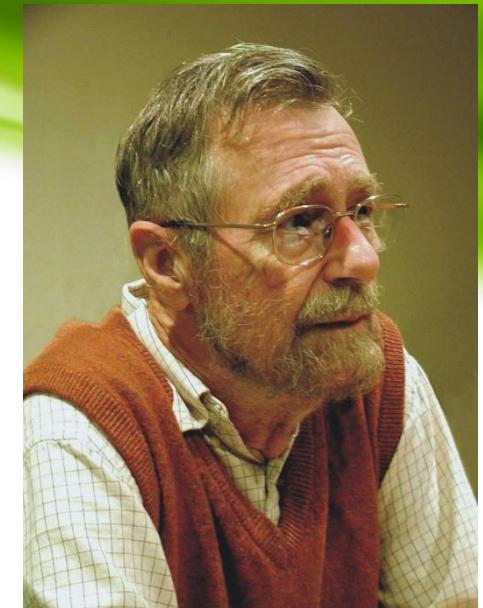
$$wp(S, P) \Rightarrow wp(S, Q)$$

③ For any S and any two post-conditions P and Q, for all states:

$$wp(S, P) \text{ and } wp(S, Q) = wp(S, P \text{ and } Q)$$

④ For any deterministic S and any post-conditions P and Q, for all states:

$$(wp(S, P) \text{ or } wp(S, Q)) \Rightarrow wp(S, P \text{ or } Q)$$



Edsger Wybe Dijkstra

(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

Assignment and concatenation operator

- Assignment

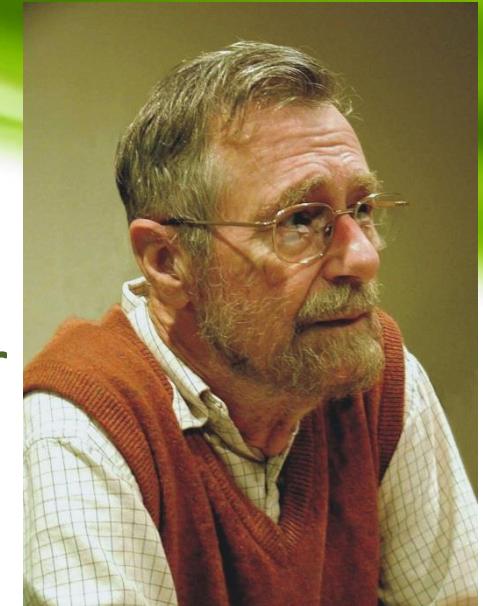
The semantics of $x := E$ are given by:

$wp("x := E", R) = R_E^x$, R_E^x -denotes a copy of the predicate defining R in which each occurrence of the variable x is replaced by E.

- Concatenation operator ;

The semantics of the ; operator are given by:

$wp("S1 ; S2", R) = wp(S1, wp(S2, R)).$

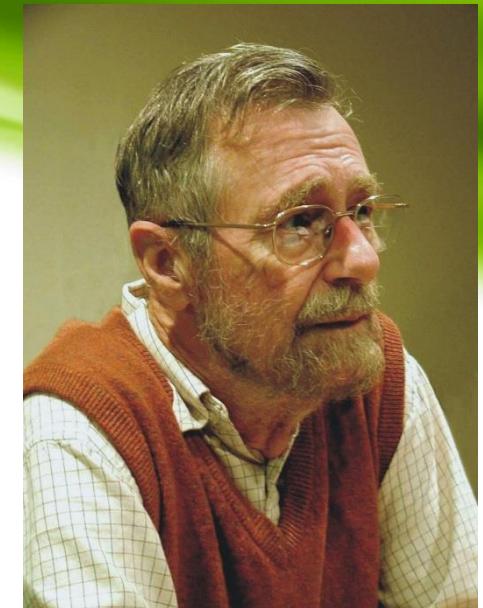


Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

The Alternative Construct

- Let *IF* denote: **if** $B_1 \rightarrow SL_1 \square \dots \square B_n \rightarrow SL_n$ **fi**
Let *BB* denote: $(\exists i : 1 \leq i \leq n : B_i)$, then, by definition
 $wp(IF, R) = (BB \text{ and } (\forall i : 1 \leq i \leq n : B_i \Rightarrow wp(SL_i, R)))$.
- Theorem 1
From $(\forall i : 1 \leq i \leq n : (Q \text{ and } B_i) \Rightarrow wp(SL_i, R)$ for all states we can conclude that $(Q \text{ and } BB) \Rightarrow wp(IF, R)$ holds for all states.)
- Let *t* denote some integer function, and $wdec(S, t)$
- Theorem 2
From $(\forall i : 1 \leq i \leq n : (Q \text{ and } B_i) \Rightarrow wdec(SL_i, t))$ for all states we can conclude that $(Q \text{ and } BB) \Rightarrow wdec(IF, t)$ hold for all states.
- By definition,
 $wdec(S, t) = (tmin(X) \leq t(X) - 1) = (tmin(X) < t(X))$.

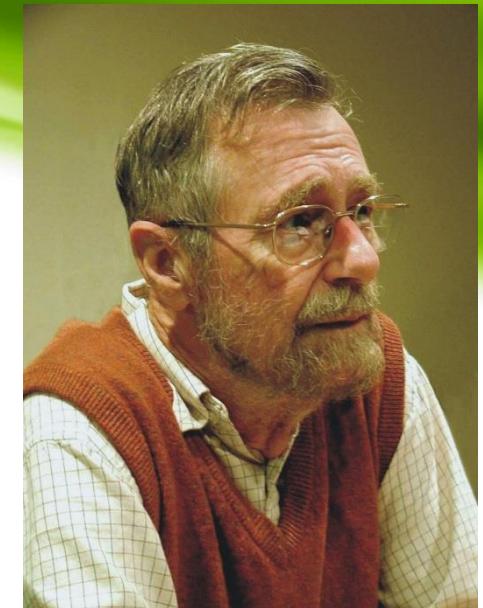


Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

The Alternative Construct - example

- The formal requirements for performing $m := \max(x, y)$ is:
 $R : (m = x \text{ or } m = y) \text{ and } m \geq x \text{ and } m \geq y.$
- Assignment $m := x$ for $m = x?$
 $wp("m := x", R) = (x = x \text{ or } x = y) \text{ and } x \geq x \text{ and } x \geq y = x \geq y$
- Theorem 1: **if** $x \geq y \rightarrow m := x **fi**$
- But $B \neq T$, so we weakening BB means looking for alternatives which might introduce new guards.
- Alternative: " $m := y$ " that introduces the new guard
 $wp("m" := y, R) = y \geq x$
if $x \geq y \rightarrow m := x$
 $\square y \geq x \rightarrow m := y$
fi



Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- The Repetitive Construct

- Let DO denote: $\mathbf{do}B_1 \rightarrow SL_1 \square \dots \square B_n \rightarrow SL_n \mathbf{do}$

Let $H_0 = (R \text{ and non } BB)$

and for $k > 0$, $H_k(R) = (wp(IF, H_{k-1}(R))) \text{ or } H_0(R)$

then, by definition: $wp(DO, R) = (\exists k : k \geq 0 : H_k(R))$.

- Theorem 3

If we have all the states

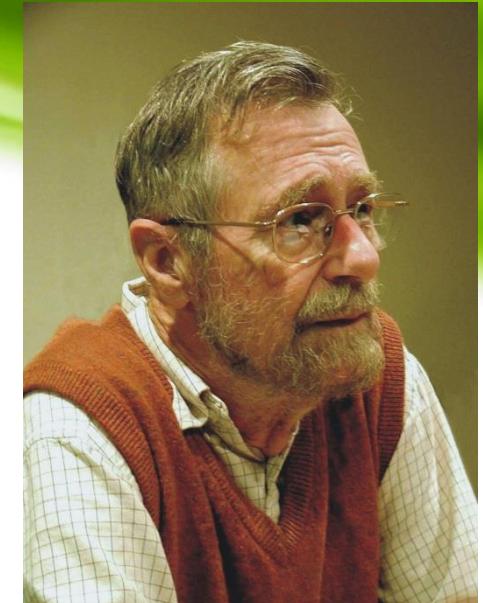
$(P \text{ and } BB) \Rightarrow (wp(IF, P) \text{ and } wdec(IF, t) \text{ and } t \geq 0)$ we can conclude that we have for all states $P \Rightarrow wp(DO, P \text{ and non } BB)$

- T is the condition satisfied by all states, and $wp(S, T)$ is the weakest pre-condition guaranteeing proper termination of S .

- Theorem 4

From $(P \text{ and } BB) \Rightarrow wp(IF, P)$ for all states, we can conclude that we have for all states

$(P \text{ and } wp(DO, T) \Rightarrow wp(DO, P \text{ and non } BB))$

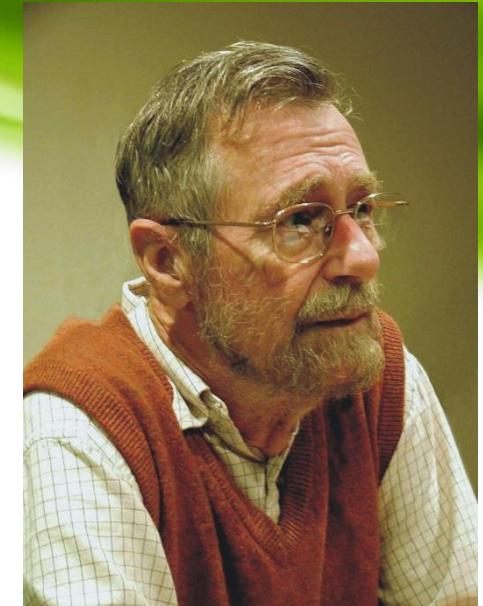


Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- The Repetitive Construct - example

- The greatest common divisor: $x = \gcd(X, Y)$
- Choose an invariant relation and variant function.
establish the relation P to be kept invariant
do "decrease t as long as possible under variance of P" **od**
- invariant relation (established by $x := X; y := Y$):
 $P : \gcd(X, Y) = \gcd(x, y)$ **and** $x > 0$ **and** $y > 0$
- $(P \text{ and } B) \Rightarrow wp(\text{"}x, y : E1, E2\text{"}, P)$)
 $= (\gcd(X, Y) = \gcd(E1, E2) \text{ and } E1 > 0 \text{ and } E2 > 0).$
 - $\gcd(X, Y) = \gcd(E1, E2)$ is implied by P
 - invariant for $(x, y) : wp(\text{"}x := x - y, P\text{"}) = (\gcd(X, Y) = \gcd(x - y, y)$ **and** $x - y > 0$ **and** $y > 0$), and guard $x > y$
 - decrease of the variant function $t = x + y$
 $wp(\text{"}x := x - y\text{"}, t \leq t_0) = (x \leq t_0)$
 $t_{min} = x$, $wdec(\text{"}x := x - y\text{"}, t) = (x < x + y) = y > 0$



Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- $x := X; y := Y$
do $x > y \rightarrow x := x - y$ **od**
- But P **and** BB - are not allowed to conclude $x = \gcd(X, Y)$
the alternative $y := y - x$ requires a guard $y > x$
- $x := X; y := Y$
do $x > y \rightarrow x := x - y$
 $\square y > x \rightarrow y := y - x$
od

Surprise!

Formative Assessment

Dijkstra's Language,
Guarded commands,
Nondeterminacy,
Formal Derivation of Programs

3-2-1 count down exercise

- 3 things you didn't know before
- 2 things that surprised you about this topic
- 1 thing you want to start doing with what you've learned

Surprise!





Floyd, Dijkstra, Hoare (25XP)

- **Robert Floyd OR Edsger Wybe Dijkstra OR Charles Antony Richard Hoare**
- 1 page A4 information (electronic format and printed format)
 - short bio
 - profession
 - Institution
 - known by...
 - awards
 - interesting facts
- Feel free to select a format: plain text, mindmap, other
- Delivery: Lecture 8

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
 - Correctness-by-Construction.**
Originally intended as a mere means of programming algorithms that are correct by construction - Dijkstra (1968), Hoare (1971), the approach found its way into commercial development processes of complex systems - Hall (2002), Hall and Chapman (2002) 2012, The Correctness-by-Construction Approach to Programming, Authors: **Kourie**, Derrick G., **Watson**, Bruce W. 2015, Experience with correctness-by-construction, B.W. Watson a, D.G. Kourie b, L. Cleophas b, 2016, Correctness-by-Construction and Post-hoc Verification: Friends or Foes?, M. Beek , R. Hahnle,I. Schaefer 2016, Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience? B. Watson, D. Kourie, I. Schaefer, L. Cleophas
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
 - **EVOZON** Presentation, topic: **Test automation**
 - **When:** Friday, April 12, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Developing correct programs from specification [Mor98]

- Refinement

- Input data: X
- Output data: Z
- Abstract program
 $Z : [\varphi, \psi]$
- Refinement
 $P_1 \prec P_2 \prec \dots \prec P_{n-1} \prec P_n$
- Rules of refinement
 - Assignment rule
 - Sequential composition rule
 - Alternation rule
 - Iteration rule

$$\begin{aligned}\varphi(X) \\ \psi(X, Z)\end{aligned}$$

Carroll
Morgan

https://my.cse.unsw.edu.au/staff/staff_details.php?ID=carrollm

Developing correct programs from specification [Mor98]

• Rules of Refinement

- Assignment rule: $[\varphi(v/e), \psi] \prec v := e$
- Sequential composition rule (γ – middle predicate)
 $[\eta_1, \eta_2] \prec [\eta_1, \gamma]$
 $[\gamma, \eta_2]$
- Alternation rule, $G = g_1 \vee g_2 \vee \dots \vee g_n$
 $[\eta_1, \eta_2] \prec$
if $g_1 \rightarrow [\eta_1 \wedge g_1, \eta_2]$
 $\square g_2 \rightarrow [\eta_1 \wedge g_2, \eta_2]$
 \vdots
 $\square g_n \rightarrow [\eta_1 \wedge g_n, \eta_2]$
fi
- Iteration rule $G = g_1 \vee g_2 \vee \dots \vee g_n$
 $[\eta, \eta \wedge \neg G] \prec$
do $g_1 \rightarrow [\eta \wedge g_1, \eta \wedge TC]$
 $\square g_2 \rightarrow [\eta \wedge g_2, \eta \wedge TC]$
 \vdots
 $\square g_n \rightarrow [\eta \wedge g_n, \eta \wedge TC]$
do

Developing correct programs from specification [Mor98]

- Examples

- See the file with the examples
 - One example is discussed during lecture.
- Book [Mor98]

Surprise!

Formative Assessment

Stop and Go

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
 - **EVOZON** Presentation, topic: **Test automation**
 - **When:** Friday, April 12, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Program verification methods - Correctness

- Lecture 1 - Verification and Validation
 - Verification
 - reviews products to ensure their quality → correctness
 - static and dynamic analysis techniques
 - A **correct program** is one that does exactly what it is intended to do, no more and no less.
 - A **formally correct program** is one whose correctness can be **proved** mathematically.
 - This requires a language for specifying precisely what the program is intended to do.
 - Specification languages are based in mathematical logic.
 - Until recently, correctness has been an academic exercise. – Now it is a key element of critical software systems
- **Program verification - correctness**
 - proof-based, computer-assisted, program-verification approach, mainly used for programs which we expect to terminate and produce a result
 - model-based, automatic, property-verification approach, mainly used for concurrent, reactive systems (originally used in a post-development stage) - model checking (Lecture 8, Lecture 9)
- **Correctness Tools**
 - **Theorem provers** (PVS), Modeling languages (UML and OCL), **Specification languages** (JML), Programming language support (Eiffel, Java, Spark/Ada), **Specification Methodology** (Design by contract)
- **Methods for proving program correctness**
 - Floyd's Method - Inductive assertions
 - Hoare - Semantics of Hoare triples
 - Dijkstra's Language- Guarded commands, Nondeterminacy and Formal Derivation of Programs

Program verification methods - Correctness

- **Software engineering problem:** building/maintaining **correct** systems.
 - How?
 - Specification
 - Tools
 - Formal Methods in Software Engineering
 - Formal languages guarantee
 - Precision (no ambiguity)
 - Certainty (modeling errors)
 - Automation (automatic verification tools).
- Things to do:
 - 1) make a **formal model**
 - 2) **specify properties** for the model
 - 3) **verify/check** the properties
- Formal methods and JML (Java Modeling Language):
 - 1) formal model is **Java programming language**
 - 2) the properties are specified in **JML**
 - 3) Properties may be
 - **Tested** using **jmlrac**
 - **Checked** using **ESC2Java**

Remark. ESC/Java tool - Topic of Laboratory 6!

What is JML?

- Gary T. Leavens's JML group at the University of Central Florida
- <http://www.eecs.ucf.edu/~leavens/JML//index.shtml>

Tools for using JML

- a behavioral interface specification language
- used to specify the behavior of Java modules
- combines
 - design by contract approach
 - the model-based specification approach
 - some elements of the refinement calculus

- Runtime assertion checkers (e.g. **jmlc/jmlrac**)
- Static checkers (**ESC2Java**)
- Test generation (e.g. **jmlunit**)
- Formal verification tools (e.g. **KeY**)
- Design tools (e.g. **AutoJML**)

Tools for JML

Runtime assertion checking with jmlc/jmlrac

- Special compiler inserts runtime tests for all JML assertions. Any assertion violation results in a special exception.
- checks specs at run-time
- only **tests** correctness of **specs**.
- **Find violations at runtime.**

JML web page

- <http://www.eecs.ucf.edu/~leavens/JML//index.shtml>

Extended static checking with ESC/Java

- Automatically tries to prove simple JML assertions at compile time.
- checks specs at compile-time
- **proves** correctness of **specs**.
- **Warn about likely runtime exceptions and violations.**

ESC/Java2 web page

<http://www.kindsoftware.com/products/opensource/ESCJava2/download.html>

Design by contract

Contract?

Method contract

Precondition

Specifies “caller’s responsibility”

- Constraints on parameter values and target object’s state.
- Valid object’s states, in which a method can be called.

Intuitively

- Expression that must hold at the entry to the method.

Postcondition

Specifies “implementation’s responsibility”

- Constraints on the method’s return value and side effects.
- Relation between initial and final state of the method.

Intuitively

- Expression that must hold at the exit from the method.

Class contract

Invariant

- Specifies caller’s responsibility at the entry to a method and implementation’s responsibility at the exit from a method.
- Valid states of class instances (values of fields).

Intuitively

- Expression that must hold at the entry and exit of each method in the class.

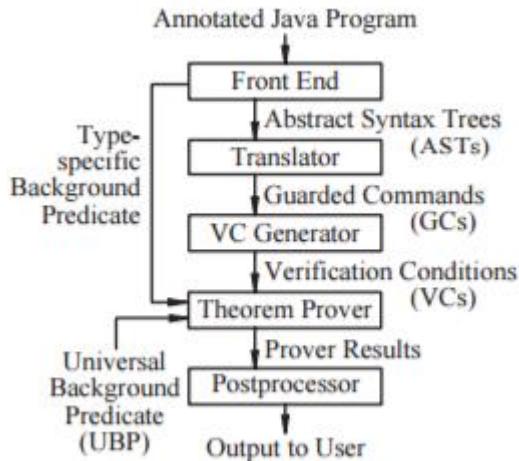
Tools for JML

Runtime assertion checking with jmlc/jmlrac

- Special compiler inserts runtime tests for all JML assertions. Any assertion violation results in a special exception.
- checks specs at run-time
- only **tests** correctness of **specs**.
- **Find violations at runtime.**

jmlc and jmlrac – by example

- Demo 01: Factorial
- Demo02: Integer sqrt



- Unsound ?
- Incomplete ?

Tools for JML

Extended static checking with ESC/Java

- Automatically tries to prove simple JML assertions at compile time.

ESC/Java2 – by example

- Demo 01: Fast exponentiation
- Demo 02: MyArray
- Demo 03: MySet

- checks specs at compile-time
- proves correctness of specs**
- Warn about likely runtime exceptions and violations.**

Surprise!

Didactic - Lecture 6 - Teaching-Learning -
Formative-Summative Assessment

No name required (only on paper).

25 XP

Next Lecture

- **EVOZON** Presentation, topic: **Test automation**
- **When:** Friday, April 12, 2019, hours 14:00-16:00;
- **Where:** Room A2 (FSEGA Building)

Questions

- Thank You For Your Attention!

References

- [Flo67] Robert W. Floyd, Assigning meanings to programs, In *Proceedings of Symposia in Applied Mathematics*, pages 19–32, 1967.
- [Hoa69] C. A. R. Hoare, An axiomatic basis for computer programming, *Commun. ACM*, 12(10):576–580, 1969.
- [Dij75] E. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs.
- CACM, 8(18):453–457, 1975.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010.
- [Mor98] C. Morgan, Programming from Specifications, Prentice Hall International Series in Computing Science, 1998

Introduction to automation testing



Join at

slido.com

#6940

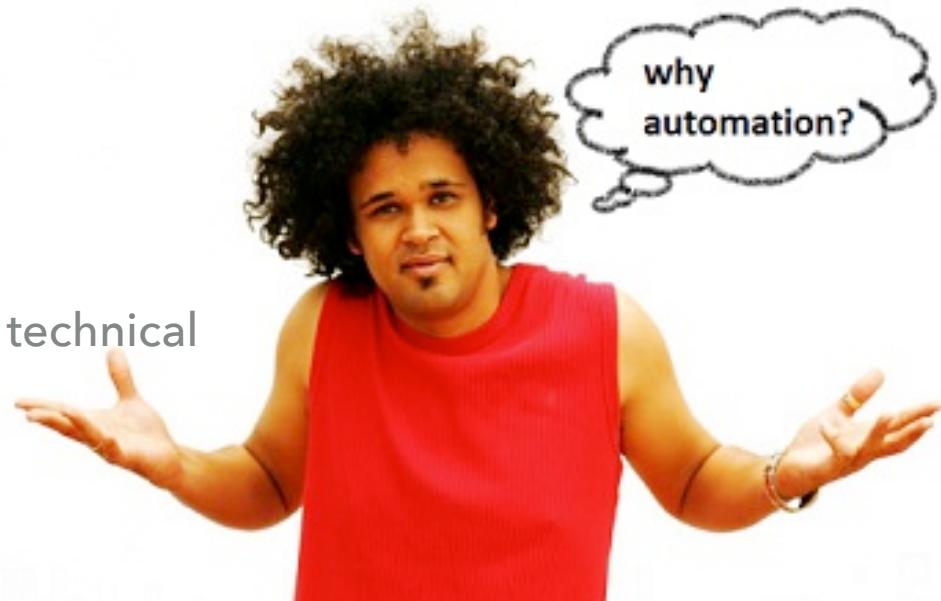


What is automation testing?

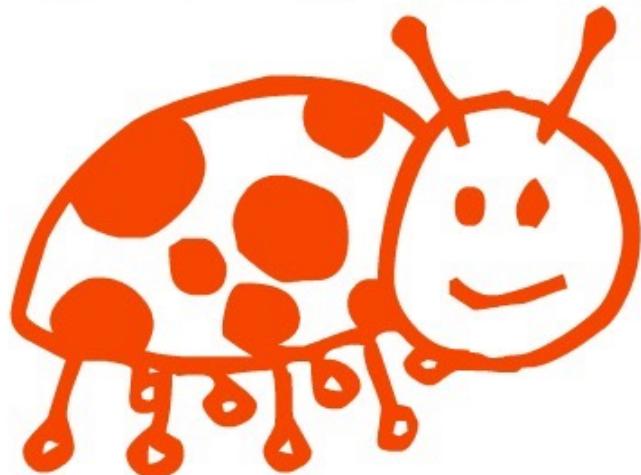
- In software **testing**, **test automation** is the use of special software (separate from the software being **tested**) to control the execution of **tests** and the comparison of actual outcomes with predicted outcomes.

Why use automation testing?

- Automation testing is cheaper
- Automation testing is faster
- Automation testing is reliable
- Automation testing reduces human and technical risks
- Rapid feedback during development
- Automation testing reduces testing resources



When...



TO AUTOMATE

When to automate?

- Projects with long lifespan



When to automate?

- Projects with long lifespan
- Start early



When to automate?

- Projects with long lifespan
- Start early
- Experienced and technical testers



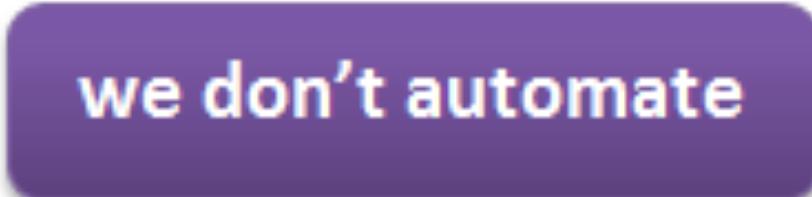
When to automate?

- Projects with long lifespan
- Start early
- Experienced and technical testers
- Frequent regression testing

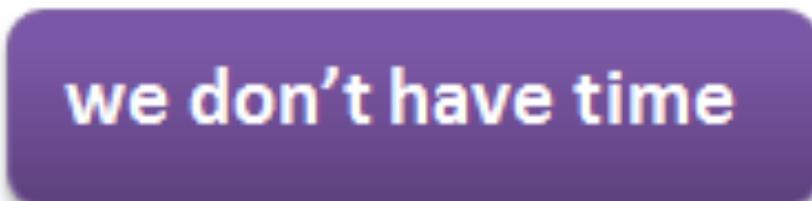




because



because



When NOT to automate?

- Projects with short lifespan
- Unstable design
- Inexperienced testers
- Insufficient time, resources

What to automate?



What to automate?

- Core application features
- Sanity/Smoke tests
- Regression tests
- Tests you need to run several times
- Areas of the app which are more “problematic”
- Things that are easy to automate and provide a “quick win”

ONE DOES NOT SIMPLY

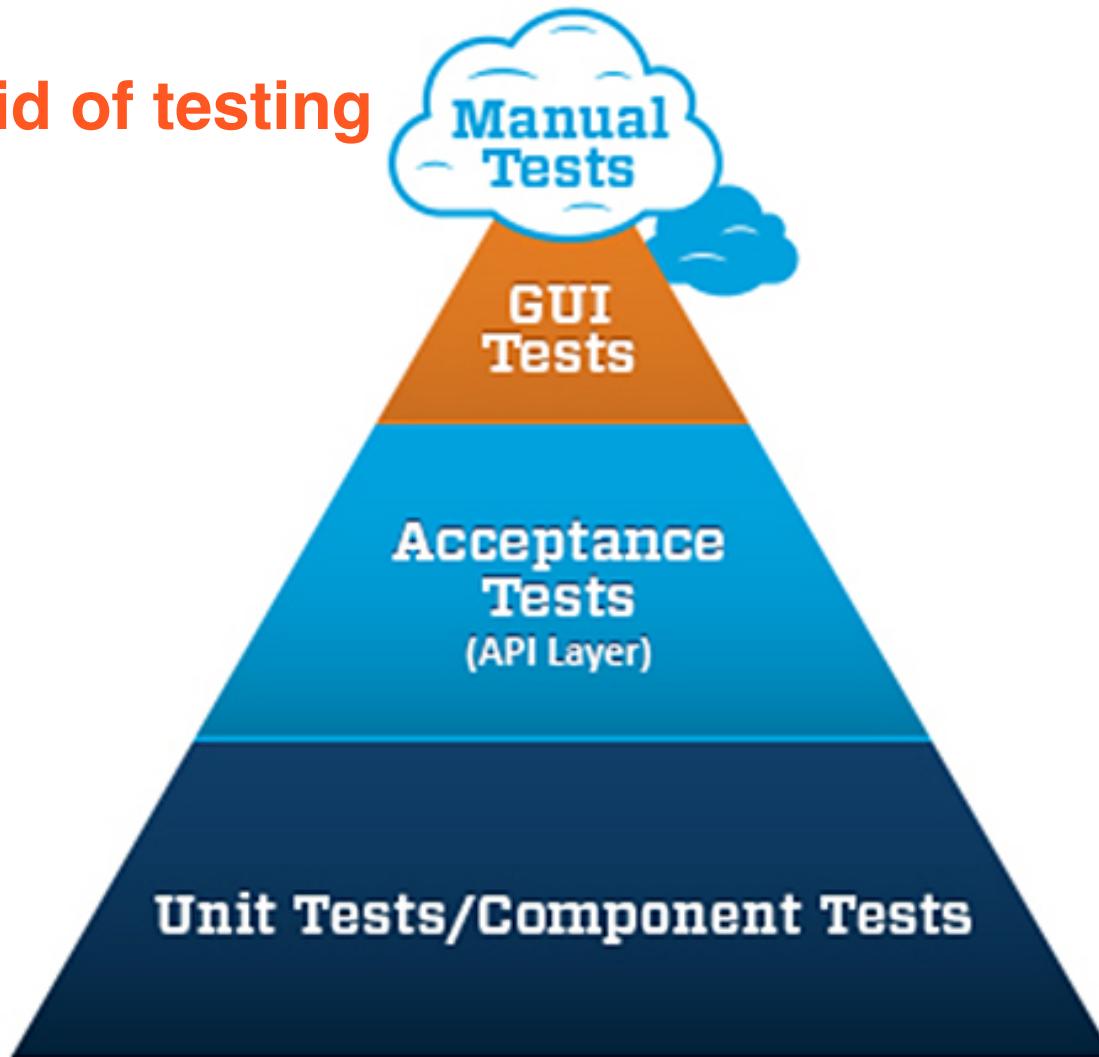


AUTOMATE ALL OF THE APPLICATION

What NOT to automate?

- Areas that will change
- Tests that are run only a few times
- Data validation

The pyramid of testing



How to automate?

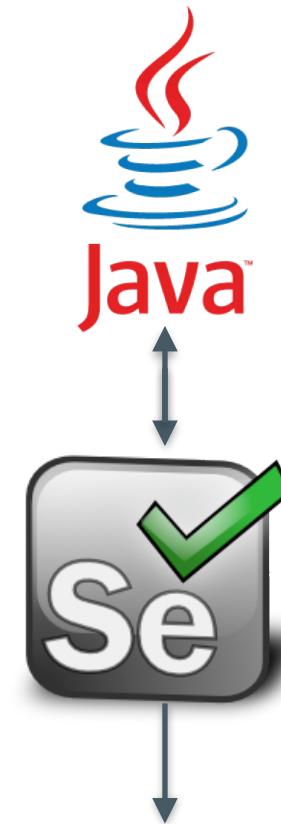
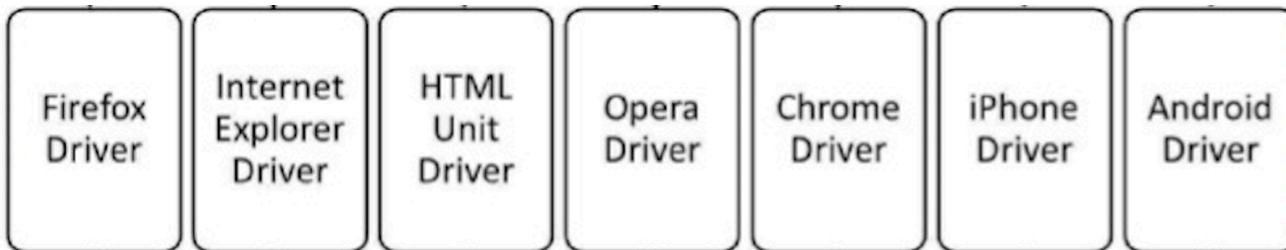
- Automation is more than a record and play tool
- Automation is more than test execution
- Automation needs a framework
- Good automation needs code review and refactoring
- Automation needs maintenance

What to use to automate?



What do we need

- Programming language
- Selenium WebDriver



- Browser



Selenium WebDriver

- **WebDriver** is a tool for automating web application testing, and in particular to verify that they work as expected.

JUnit

- Open source framework that can be used for writing and running automated tests that can be integrated with Selenium to produce reliable automation tests

The structure of an automated test

- Navigation
- Interactions (click, type, read text, select dropdown, etc)
- Verification (a test is not a test, unless it checks something)



Serenity BDD



What is



Serenity
BDD

- Test automation framework
- Helps you write better, more effective automated acceptance tests, and use these acceptance tests to produce world-class test reports and living documentation
- Previously known as



Why use



Serenity

BDD

- Already an established framework
- Easy to use
- Great reporting
- Is based on Selenium WebDriver
- A lot of already implemented methods (helper methods)

Automated test

```
public class NoPOMTest99GuruLogin {  
    /**  
     * This test case will login in http://demo.guru99.com  
     * Verify login page title as guru99  
     * Login to application  
     * Verify the home page using Dashboard  
     */  
    @Test(priority=0)  
    public void test_Home_Page_Appear_Correctly() {  
        WebDriver driver = new FirefoxDriver();  
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
        driver.get("http://demo.guru99.com/V4/");  
        //Find user name and fill user name  
        driver.findElement(By.name("uid")).sendKeys("demo");  
        //find password and fill it  
        driver.findElement(By.name("password")).sendKeys("password");  
        //click login button  
        driver.findElement(By.name("btnLogin")).click();  
        String homeText = driver.findElement(By.xpath("//table//tr[@class='heading3']")).getText();  
        //verify login success  
        Assert.assertTrue(homeText.toLowerCase().contains("guru99 bank"));  
    }  
}
```

This is a small script. Script maintenance looks easy. But with time test suite will grow. As you add more and more lines to your code, things become tough.

The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.

① Find user name and fill it

② Find password and fill it

③ Find Login button and click it

Find home page text and get it

④

⑤ Verify home page has text 'Guru99 Bank'

Automated test

- A better approach to script maintenance is to create a separate class file which would find web elements, fill them or verify them.
- This class can be reused in all the scripts using that element.
- In future, if there is a change in the web element, we need to make the change in just 1 class file and not 10 different scripts. Interactions (click, type, read text, select dropdown, etc)

What is POM?



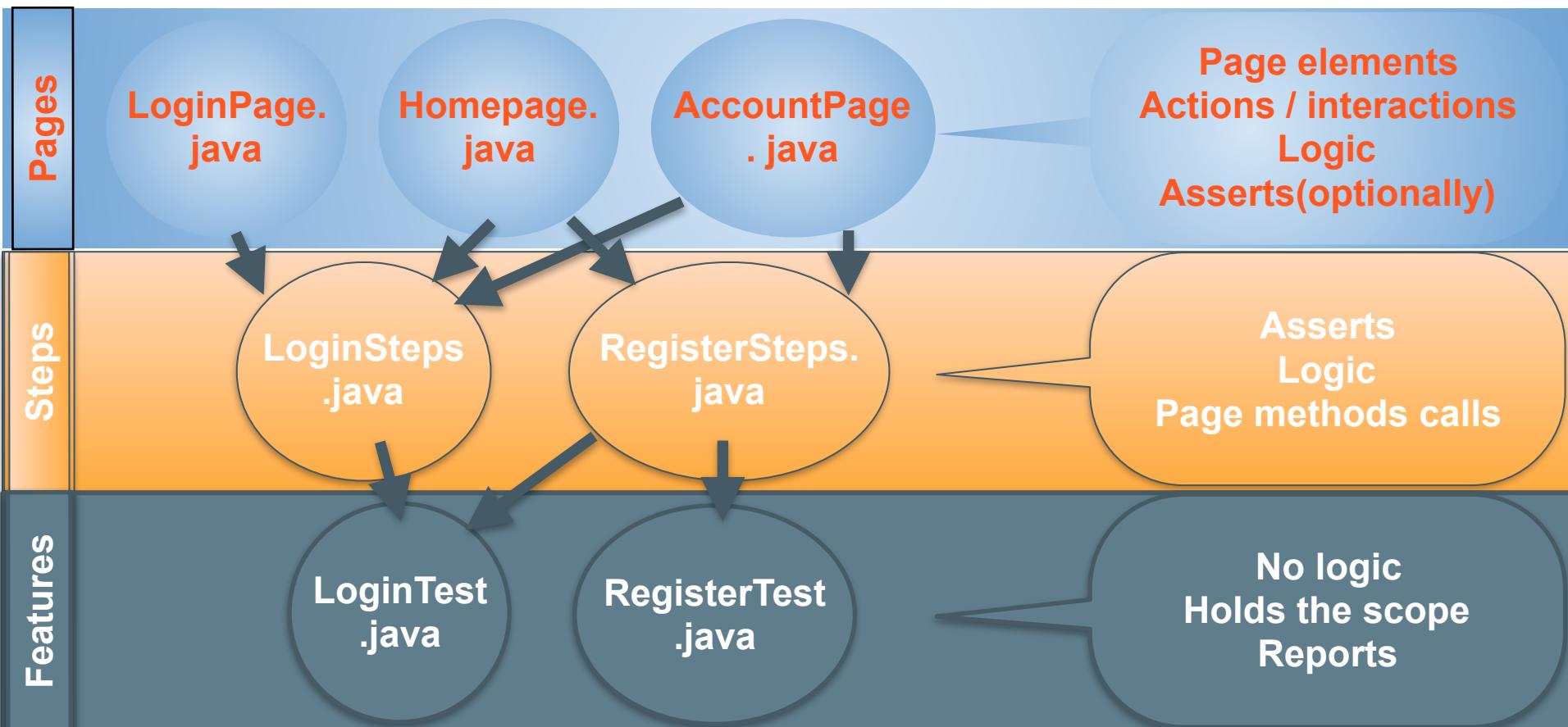
- **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
- Under this model, for each web page in the application, there should be corresponding page class.
- This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
- Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be `waitForPaymentScreenDisplay()`.

POM and Serenity?

- Serenity builds on the **POM** structure
- It adds a new layer between the classic **PAGE - TEST**
- Between these two, Serenity adds a new class, called **STEPS**
- The role of the **STEPS** class is to combine specific methods from the **PAGE** class, to create more complex functions



Pages - Steps - Tests



Demo project on GIT

<https://github.com/cosminevo/TrelloDemoApp>

- Read the [README.md](#) file
- In order to run the tests you need to have an account, and generate the KEY and TOKEN for that user

Thank you!



Performance Testing





WHY

WHERE

WHAT

WHEN

HOW

WHO

WHAT?



*Functional
Testing?*

WHAT the product
does?!

VS

*Non-functional
Testing?*



HOW WELL the
system behaves?!

Non-functional Testing Types



1. Performance
2. Usability
3. Security
4. Portability
5. Internationalization
6. Migration

WHAT is Performance Testing?

PERFORMANCE TESTING = how fast & stable is the system?

OBJECTIVE IS TO FIND THE **BOTTLENECKS** IN THE SYSTEM



WHY do we need performance testing?





evozon

TYPES...

LOAD TESTING

SOAK TESTING

STRESS TESTING

SPIKE TESTING

VOLUME TESTING

SCALABILITY
TESTING

TOOLS



POSTMAN

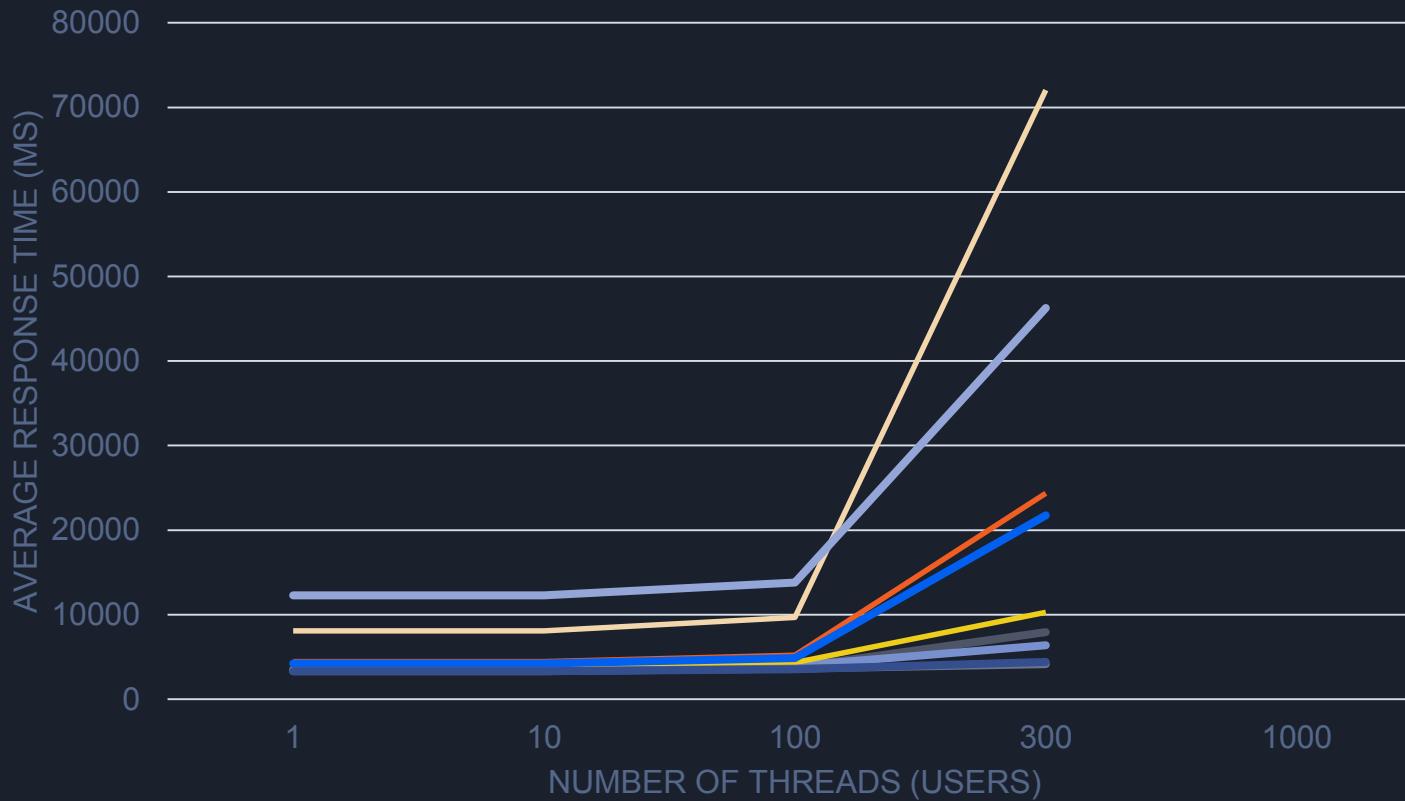


evozon

DEMO...



Analyse and Report



Prerequisites

Instal Java if not already installed

- [Download Java for Windows](#)

Get Jmeter

- [Download Apache JMeter](#)

Start Jmeter

- Unzip archive and save folder on your machine
- Start Jmeter by going to the “**bin**” folder inside the unzipped files and run
 - *jmeter.bat*



The script

- Access site (homepage)
- Search for phone
- Select Phone
- Add to cart
- View Cart

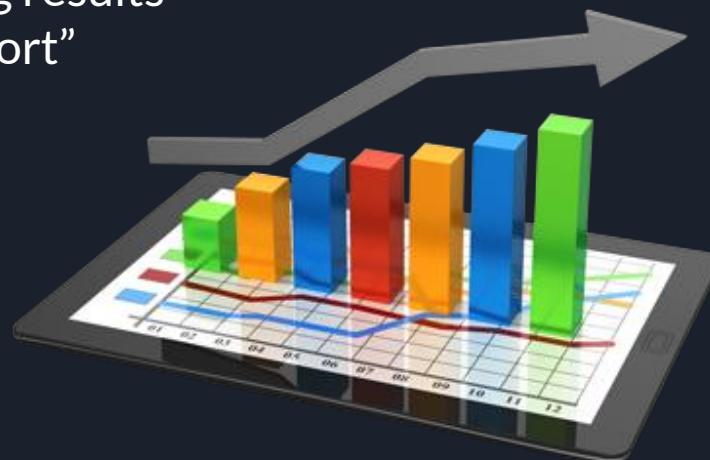


Running script

jmeter -n -t “name and location of test plan.jmx”

-I “name and location of jtl file to log results”

-e -o “folder where to generate report”





SUMMER INTERNSHIPS

Apply at internship@evozon.com until the 26th of April
mentioning the team you'd like to be part of.

evozon

Java Web

.Net

Javascript

PHP

Testing

Questions



Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan Lecture 8: Symbolic execution

Babeş-Bolyai University

Cluj-Napoca

2018-2019



Surprise!



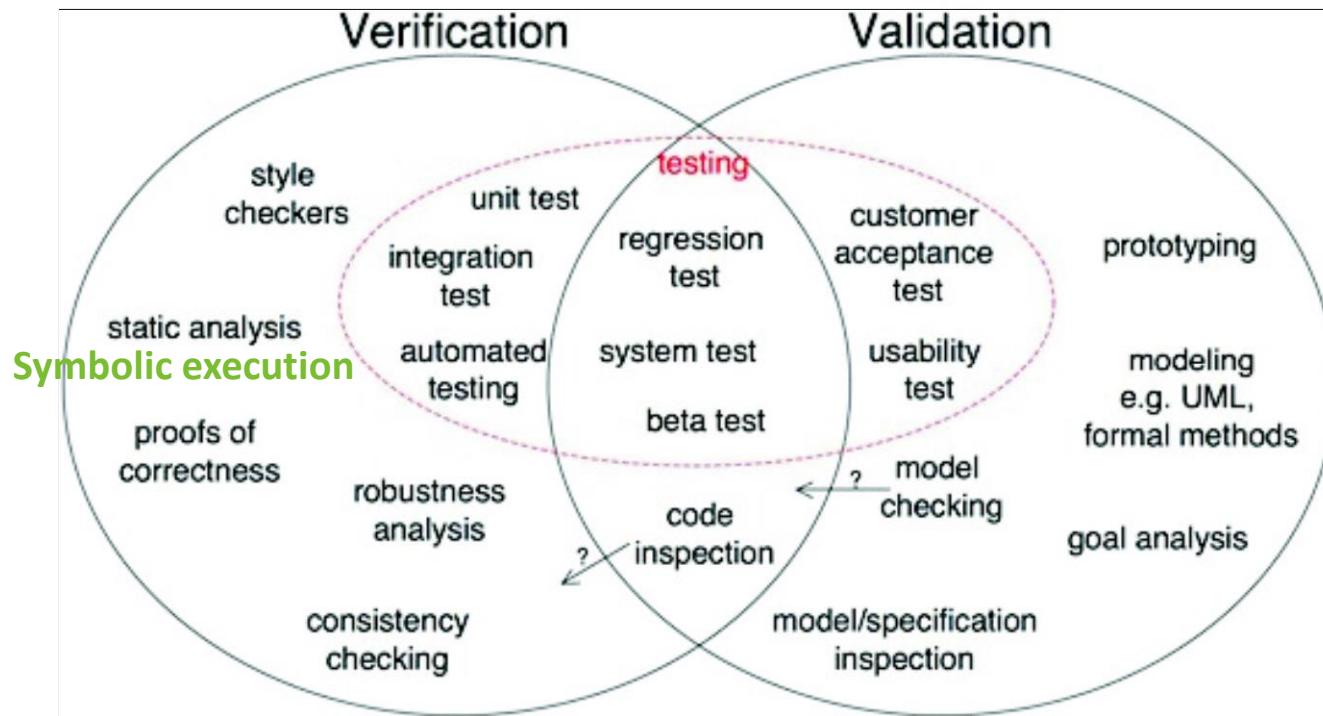


Floyd, Dijkstra, Hoare (25XP)

- **Robert Floyd OR Edsger Wybe Dijkstra OR Charles Antony Richard Hoare**
- 1 page A4 information (electronic format and printed format)
 - short bio
 - profession
 - Institution
 - known by...
 - awards
 - interesting facts
- Feel free to select a format: plain text, mindmap, other
- Delivery: Lecture 8

Sales paradigm - SSVV

- Motivate the STUDENT - what you will learn!



- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>

Outline

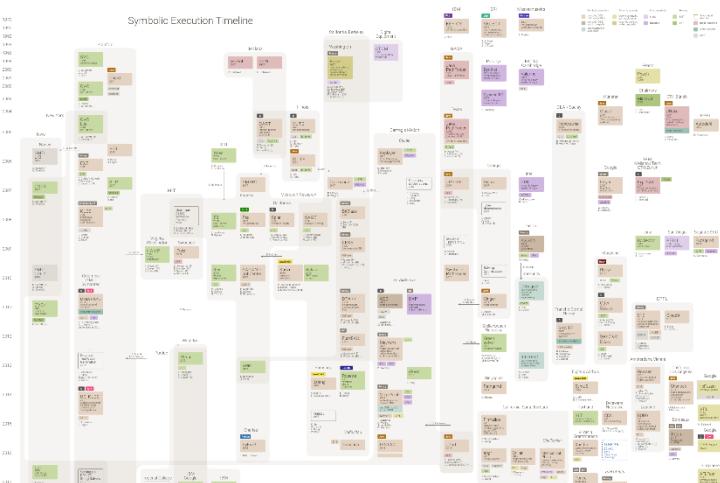
- Static analysis, Testing, Symbolic execution
- Conventional vs Symbolic execution
- Symbolic execution for sequential, alternative, repetitive structures
 - Sequential structure execution
 - Alternative structure execution
 - Repetitive structure execution
- Symbolic Execution Tree
 - Symbolic Execution Tree
 - Properties
- Questions
- Next lecture (in L09 (free on 26 April 2019) but still today)
 - Model checking
- Next next Lecture (L10)
 - ENDAVA Presentation, topic: **Verification and Validation during the Software Development Life Cycle**
 - **When:** Friday, May 10, 2019, hours 14:00-16:00;
 - **Where:** Room A2 (FSEGA Building)

Static analysis

Symbolic execution

- Bugs that are missed by testing: rare features, rare circumstances, nondeterminism.
 - ➔ Static analysis
 - Can analyze all possible runs of a program
 - But, can it finds deep, difficult bugs?
 - Abstraction let us model all possible runs
 - Static analysis abstraction <>> developer abstraction
- Testing works
 - reported bugs are real bugs, but each test only explores **one** possible execution.
`assert (f(5)==6)`
 - We **hope** test cases generalize, but no guarantees!
- ➔ Symbolic execution **generalizes** testing
 - ➔ $y=\alpha$, `assert(f(y)==2*y+1)`
- **Remarks:**
 - symbolic execution is not meant to inspect the quality of the code.
 - static analysis deals with issues of path feasibility,
 - dynamic analysis tends to deal with path coverage.
 - Symbolic analysis is sort of in between and deals with state space explosion by logically forking the analysis at branches and solving for a set of satisfiable constraints.

Symbolic execution - research

- 1976 - King [Kin76], Clarke [Cla76]
-
- 2005 - Microsoft: DART [God05]
- 2006 - Univ. Stanford: EXE, Univ. Illinois: CUTE and jCUTE [SA06]
-
- 2008 - KLEE (Stanford) [CDE08]
-
- 1999 - 2016 - NASA: Symbolic (Java) Path Finder [PV09], [CS13]
 - <http://javapathfinder.sourceforge.net/>
 - <http://babelfish.arc.nasa.gov/trac/jpf>
- Modern Symbolic Execution Techniques
 - mix concrete and symbolic execution
 - Concolic testing (DART – Directed Automated Random Testing)
 - EGT (Execution-Generated Testing)
- 2017 -Learn&Fuzz: Machine Learning for Input Fuzzing
 - <https://patricegodefroid.github.io/>
- 2018
 - Chopped Symbolic Execution (ICSE) (2006 -EXE)
 - Shadow Symbolic Execution for Testing Software Patches
 - <https://www.doc.ic.ac.uk/~cristic/>
- 2018 -Deep Reinforcement Fuzzing, Konstantin Böttinger, Patrice Godefroid, Rishabh Singh
- SAGE (2005 -DART)
 - <https://patricegodefroid.github.io/>
 - <https://channel9.msdn.com/blogs/peli/automated-whitebox-fuzz-testing-with-sage> - **video**
- <https://www.microsoft.com/en-us/security-risk-detection/>
- PEX
 - <https://www.microsoft.com/en-us/research/project/pex-and-moles-isolation-and-white-box-unit-testing-for-.net/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fprojects%2Fpex%2F>
- Symbolic execution timeline

Symbolic execution

What is symbolic execution ?

- Symbolic execution
 - Execution of program with symbols as argument.
 - Symbolic execution supplies symbols (as input to a program) representing arbitrary values.
 - $\text{int FunctionName}(1, 2) \rightarrow \text{int FunctionName}(a1, a2)$
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- Symbolic execution
 - Produces a concrete input (a test case) on which the program will fail to meet the specification.
 - But it cannot, in general, prove the absence of errors
- Key idea
 - Evaluate the program on symbolic input values
 - Use an automated theorem prover to check whether there are corresponding concrete input values that make the program fail.

Symbolic execution

Symbolic state

- **Symbolic state**
 - Set of (particular) concrete states, yet not instantiated.
 - Symbolic states represent sets of concrete states.
- A **symbolic state** is described by:
 - **Variables**, i.e. symbolic values/expressions for variables;
 - **Path condition** - a conjunct of constraints on the symbolic input values;
 - **Program counter** - the statement that is executed.
- All paths in the program form its execution tree, in which some paths are feasible and some are infeasible.
- Symbolic execution is a bug finding technique based on automated theorem proving:
 - Evaluates the program on symbolic inputs, and a solver finds concrete values for those inputs that lead to errors.

Conventional vs Symbolic execution

Conventional execution (CE)

- Function Sum
- Normal execution result of $\text{Sum}(1,3,5)$
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;
- 5: return z;
- 6:

	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9
5	1	3	5	4	8	9

Conventional vs Symbolic execution

Symbolic execution (SE)

- Function Sum
- Symbolic execution result of Sum
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;
- 5: return z;
- 6:

	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	$\alpha+\beta$	-	-
3	α	β	γ	$\alpha+\beta$	$\beta+\gamma$	-
4	α	β	γ	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$
5	α	β	γ	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$

Symbolic execution

Symbolic execution for **sequential, alternative, repetitive structures**

- **Sequential structure execution**
 - path condition
 - condition to execute a statement;
 - when the symbolic execution starts, the value(pc) = true
 - the condition is updated from one statement to other
 - If τ represents the condition to execute statement $< I >$ then
 $pc' = pc \wedge \tau(I)$

Symbolic execution

Symbolic execution for
sequential, alternative, repetitive structures

Conventional

	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9
5	1	3	5	4	8	9

- Sequential execution -

```
1 : int Sum(int a, int b, int c)
2 :   int x := a + b;
3:   int y := b + c;
4:   int z := x + y - b;
5: return z;
6:
```

Symbolic

	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	$\alpha+\beta$	-	-
3	α	β	γ	$\alpha+\beta$	$\beta+\gamma$	-
4	α	β	γ	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$
5	α	β	γ	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$

Symbolic execution

Symbolic execution for sequential, **alternative**, repetitive structures

- Alternative structure execution
- Symbolic execution of an IF statement
 - if (η) then
 - A
 - else
 - B.
- During symbolic execution → value(η) could be true, false, or some symbolic formula over the input symbols.
 - “unresolved” execution of a conditional statement
- Path Condition (Initial value of pc is true)
 - $pc \rightarrow \eta$
 - $pc \rightarrow \neg\eta$

Symbolic execution

Symbolic execution for
sequential, **alternative**, repetitive structures

Symbolic

Conventional

- Alternative execution -

	x	b	If condition
1	6	-	-
2	6	False	-
3	6	False	6 modulo 2=0
4	6	True	6 modulo 2=0
6	6	True	6 modulo 2=0

```
1 : boolean IsEven(int x)
2 : boolean b := False;
3: If (x modulo 2 ==0) then
4:     b:=true;
    else
5:     b:=false;
6: IsEven:=b;
```

	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	$\alpha \text{ modulo } 2=0$
Case ($\alpha \text{ modulo } 2=0$) is True			
3	α	False	$\alpha \text{ modulo } 2=0$
4	α	True	$\alpha \text{ modulo } 2=0$
6	α	True	$\alpha \text{ modulo } 2=0$
Case ($\text{not } (\alpha \text{ modulo } 2=0)$) is True			
5	α	False	$\text{not}(\alpha \text{ modulo } 2=0)$

Symbolic execution

Symbolic execution for sequential, alternative, **repetitive** structures

- Symbolic execution of an WHILE statement

while (η)

A

endWh;

B

- During symbolic execution → value(η) could be true, false, or some symbolic formula over the input symbols.
→ “unresolved” execution of a conditional statement
- Condition to execute A: pc for executing “while” and η .
- Condition to execute B: pc for executing “while” and $\neg \eta$.

Symbolic execution

Symbolic execution for sequential,
alternative, repetitive structures

Conventional

	x	y	z	u	While condition	
1	5	3	-	-		
2	5	3	1	-		
3	5	3	1	1		
4	5	3	1	1	1<=3	
5	5	3	5	1		
6	5	3	5	2		
4	5	3	5	2	2<=3	
5	5	3	25	2		
6	5	3	25	3		
4	5	3	5	3	3<=3	
5	5	3	75	3		
6	5	3	75	4		
4	5	3	75	4	not 4<=3	
7						
8	5	3	75	4		

- Repetitive execution -

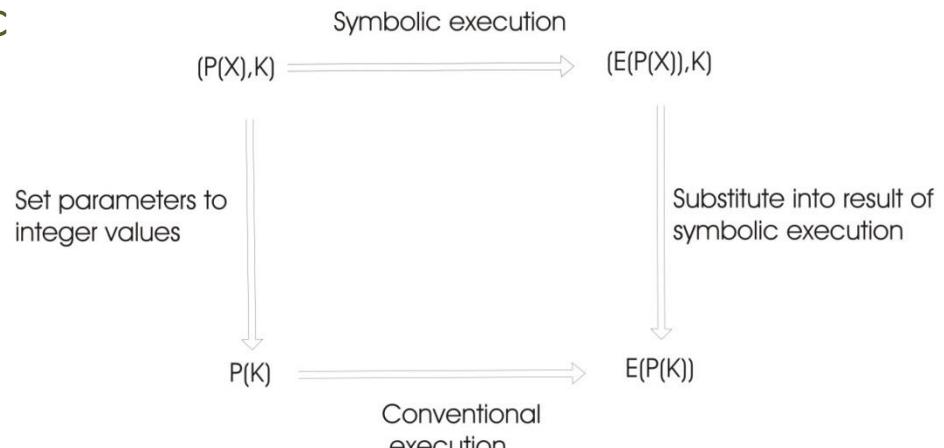
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while(u ≤ y)
 - 5: z:=z*x;
 - 6: u:=u+1
 - 7: endwh;
 - 8:

x	y	z	u	Path condition	Remarks
1	α	β	-	-	True
2	α	β	1	-	
3	α	β	1	1	
4	α	β	1	1	1≤=β
					Case not(1≤=β), → 1>β
4	α	β	1	1	1>β
8	α	β	1	1	β=0 and z=1
					Case (1≤=β)
4	α	β	1	1	1≤=β
5	α	β	α	1	1≤=β
6	α	β	α	2	1≤=β
7					
4	α	β	α	2	2≤=β and 1≤=β
					Case not(2≤=β) and 1≤=β, → 2>β and 1≤=β
4	α	β	α	2	2>β and 1≤=β
8	α	β	α	2	β=1 and z=α
					Case (2≤=β) and 1≤=β
4	α	β	α	2	2≤=β and 1≤=β
5	α	β	α ²	2	2≤=β and 1≤=β
6	α	β	α ²	3	2≤=β and 1≤=β
7					
4	α	β	α ²	3	3≤=β and 2≤=β and 1≤=β
					Case not(3≤=β) and 2≤=β and 1≤=β → 3>β and 2≤=β and 1≤=β
4	α	β	α ²	3	3>β and 2≤=β and 1≤=β
8	α	β	α ²	3	β=2 and z=α ²

Symbolic execution

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
 - $\text{Sum}(a, b, c) \rightarrow \text{Sum}(1, 3, 5)$
 - $\text{Sum}(1, 3, 5) = 9$
- Symbolic execution (SE)
 - $\text{Sum}(a, b, c) = \alpha + \beta + \gamma$
 - Instantiate the symbolic result
 - $\rightarrow \alpha = 1, \beta = 3, \gamma = 5$
 - $\rightarrow 1+3+5=9$



Symbolic execution

Symbolic Execution Tree

- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
 - Associate a node with each statement executed.
 - Associate a directed arc connecting the associated nodes with each transition between statements.
 - For IF statement execution, the associated node has two arcs leaving the node which are labeled “T” and “F” for the true and false part, respectively.
 - Associate the complete current execution state, i.e. variable values, statement counter, and pc with each node.

Symbolic execution

Symbolic Execution Tree

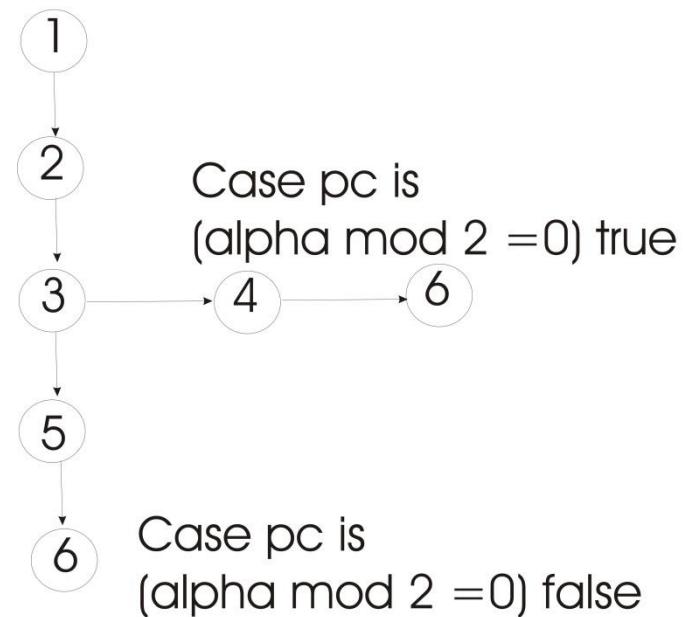
```
1 : int Sum(int a, int b, int c)
2 :   int x := a + b;
3:   int y := b + c;
4:   int z := x + y - b;
5: return z;
6:
```



Symbolic execution

Symbolic Execution Tree

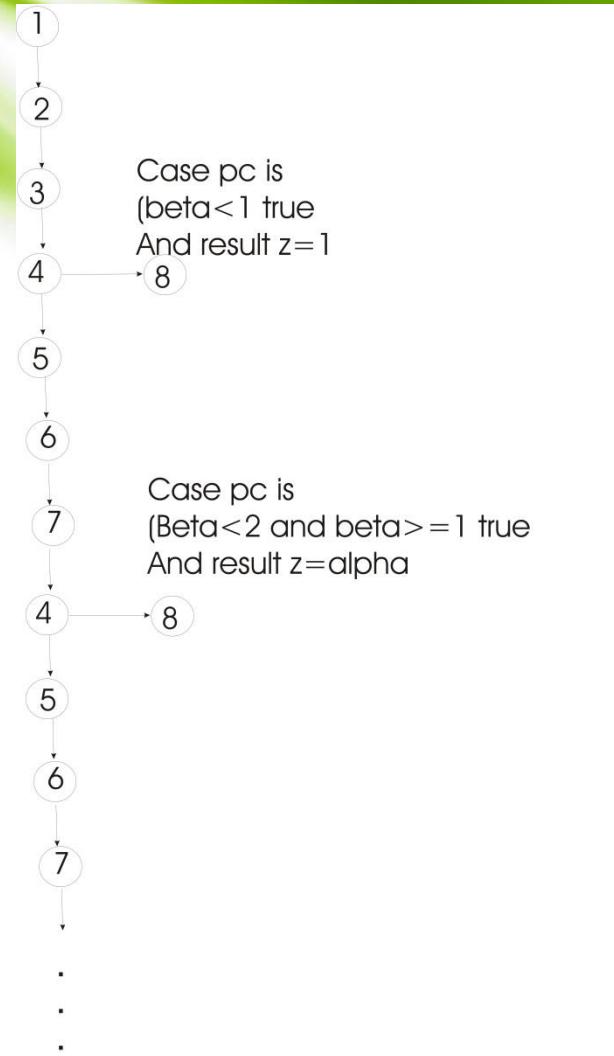
```
1 : boolean IsEven(int a)
2 : boolean b := False;
3: If (x modulo 2 ==0) then
4:     b:=true;
    else
5:     b:=false;
6: IsEven:=b;
```



Symbolic execution

Symbolic Execution Tree

```
1 : Power(int x, int y, int z)
2 :   z := 1;
3:   u:=1
4:   while(u ≤ y)
5:     z:=z*x;
6:     u:=u+1
7:   endwh;
8:
```



Symbolic execution

Properties of the Symbolic Execution Tree

- For each terminal leaf exists a particular non symbolic input.
- The pc associated with any two terminal leaves are distinct.
- Test case generation
 - to execute every statement at least once
 - to include execution of each branch both ways
 - finding input values to reach a particular point in a program
- Symbolic execution
 - Symbolic variables for input variables
 - Execute the program symbolically
 - Collect symbolic path constraints
 - Use constraint solver to generate test inputs for each execution path
- **Remaining problem** - to instantiate the pc with particular values.
- The pc specifies a class of equivalent tests, and any feasible solution to the constraints (represented by the pc) would be a representative member.
- The symbolic execution also provides expressions describing the program outputs for all inputs in this set.

Symbolic execution – research- revisited

- 1975 - First introduced
 - 1976 - King [Kin76], Clarke [Cla76]
 - 2005 - Microsoft: DART [God05]
 - 2006 - Univ. Stanford: EXE, Univ. Illinois: CUTE and jCUTE [SA06]
 - ...
 - 2008 - KLEE (Stanford) [CDE08]
 - ...
 - 1999 - 2016 - NASA: Symbolic (Java) Path Finder [PV09], [CS13]
 - <http://javapathfinder.sourceforge.net/>
 - <http://babelfish.arc.nasa.gov/trac/jpf>
 - Modern Symbolic Execution Techniques
 - mix concrete and symbolic execution
 - Concolic testing (DART – Directed Automated Random Testing)
 - EGT (Execution-Generated Testing)
 - 2017 -Learn&Fuzz: Machine Learning for Input Fuzzing
 - <https://patricegodefroid.github.io/>
 - 2018 -Chopped Symbolic Execution (ICSE) (2006 -EXE)
 - <https://www.doc.ic.ac.uk/~cristic/>
 - 2018 -Deep Reinforcement Fuzzing, Konstantin Böttinger, Patrice Godefroid, Rishabh Singh
- SAGE (2005 -DART)
 - <https://patricegodefroid.github.io/>
 - <https://channel9.msdn.com/blogs/peli/automated-whitebox-fuzz-testing-with-sage> - [video](#)
 - PEX
 - <https://www.microsoft.com/en-us/research/project/pex-and-moles-isolation-and-white-box-unit-testing-for-.net/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fprojects%2Fpex%2F>
 - Symbolic execution timeline

Surprise!

Symbolic execution

3-5 minutes

Formative Assessment

Anonymous voting

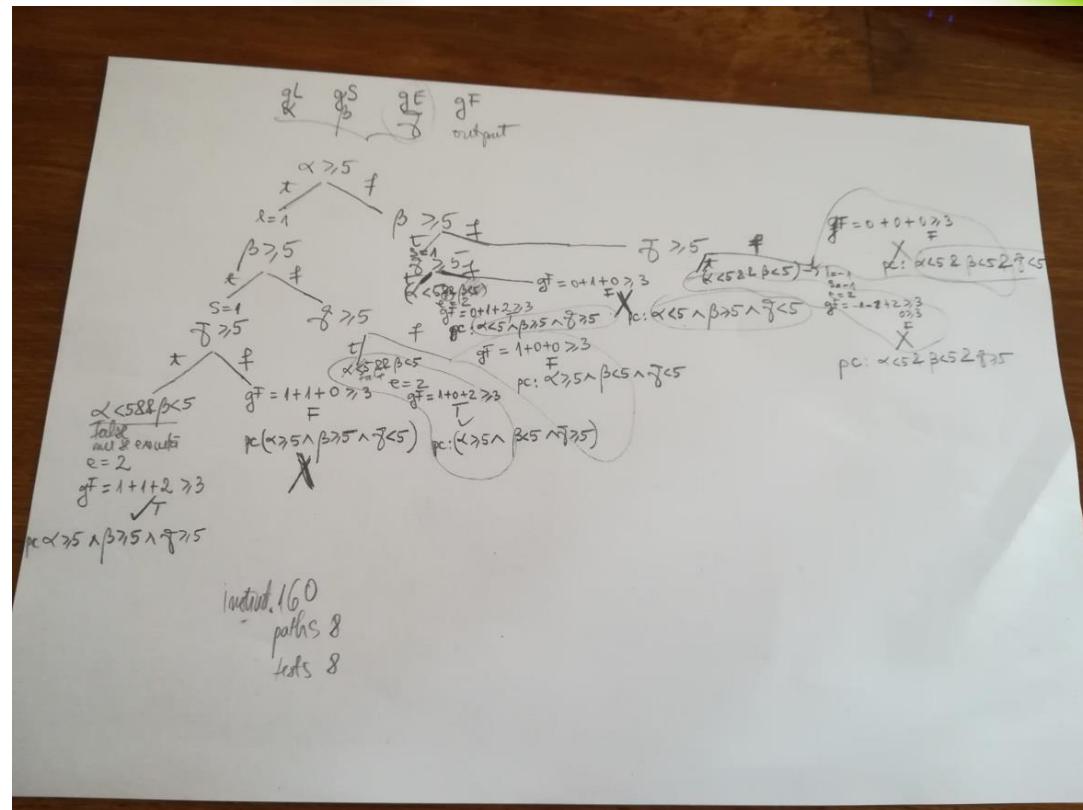
Symbolic execution

Symbolic Execution – example - <http://klee.github.io/getting-started/>

- Subalgorithm FinalGrade (gL , gS , gE , gF) is

```
1     int l=0, s=0, e=0;
2     if (gL>=5)
3         l=1
4     if (gS>=5)
5         s=1
6     if (gE>=5){
7         if(gL<5 && gS<5)
8             l=-1, s=-1
9         e = 2
10    }
11    gF= l+s +e
12    assert (gF>=3)
```

EndSubalgorithm



Surprise!





Surprise!

Symbolic execution tree

5 minutes

25 XP

In class lecture assessments
(bonus points)
Take Home – Lecture 11

```
int get_sum(int a, int b, int c) {  
    int x=0, y=0, z=0,d=0;  
    if (a >=3 )  
        x=-2;  
    if (b<5){  
        if (a<3 && c>0){  
            y=1;  
        }  
        z=2;  
    }  
    d=x+y+z;  
    // assert(d!=3);  
    return d;}
```

Questions

- Thank You For Your Attention!

References

- [Kin76] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.
- [Cla76] L. A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, SE-2(3):215–222, 1976.
- [God05] P. Godefroid. Dart: directed automated random testing. pages 213–223, 2005.
- [SA06] Koushik Sen and Gul Agha. Cute and jcute: Concolic unit testing and explicit path model-checking tools. In *Proceedings of the 18th International Conference on Computer Aided Verification*, pages 419–423, 2006.
- [CDE08] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, pages 209–224, 2008.
- [PV09] Corina S. Pasareanu and Willem Visser. A survey of new trends in symbolic execution for software testing and analysis. *Int. J. Softw. Tools Technol. Transf.*, 11(4):339–353, 2009.
- [CS13] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 56(2):82–90, 2013.

Software Systems Verification and Validation

Assoc. Prof. Andreea Vescan

Babeş-Bolyai University

Cluj-Napoca

2018-2019

Lecture 09: Model checking

Spin Model Checker



Outline

- Spin
- Promela Model
 - Statements
 - Examples
- Concurrency and Interleaving Semantics
 - Examples
- Linear Temporal Logic
 - Examples
- JSpin
- Questions

Model checking

Spin

- Developed at Bell Labs.
- In 2002, recognized by the ACM with Software System Award.
- SPIN (= Simple Promela Interpreter)
- is a tool for analyzing the logical consistency of concurrent systems
- Concurrent systems are described in the modelling language called Promela (= Protocol/Process Meta Language)

Promela

- Promela (= Protocol/Process Meta Language)
- allows for the dynamic creation of concurrent processes.
- communication via message channels can be defined to be
 - synchronous (i.e. rendezvous),
 - asynchronous (i.e. buffered).

Promela Model

- Promela model consist of:
 - type declarations
 - channel declarations
 - variable declarations
 - process declarations
 - [init process]
- A process type (**proctype**) consist of
 - a name
 - a list of formal parameters
 - local variable declarations
 - Body
- A process
 - is defined by a **proctype definition**
 - executes concurrently with all other processes, independent of speed of behaviour
 - communicate with other processes
 - using global (shared) variables
 - using channels
 - There may be several processes of the same type.
 - Each process has its own local state:
 - process counter (location within the **proctype**)
 - contents of the local variables

Statements

- The body of a process consists of a sequence of statements.
- A statement is either
 - executable: the statement can be executed immediately.
 - blocked: the statement cannot be executed.
- An assignment is always executable.
- An expression is also a statement; it is executable if it evaluates to non-zero
- The **skip statement is always executable.**
 - “does nothing”, only changes process’ process counter
- A **printf statement is always executable (but is not evaluated during verification, of course).**
- **assert(<expr>);**
 - The **assert-statement is always executable.**
 - If <expr> evaluates to zero, SPIN will exit with an error, as
- the <expr> “has been violated”.
 - The **assert-statement is often used within Promela models,**
- to check whether certain properties are valid in a state.

Examples (01 Simple Examples)

- ReversingDigits.pml
 - Check
 - Random
- DiscriminantOfQuadraticEquation.pml
 - Check
 - Random
- NumberDaysInMonth.pml
 - Check
 - Random
- MaximumNondeterminism.pml
 - Check
 - Random
 - “Branch 1” and “Branch 2”
- Maximum –second example-MaximumIfElse.pml
 - Check
 - Random
- GCD.pml
 - Check
 - Random
- IntegerDivison01.pml
 - Check
 - Random

Concurrency and Interleaving Semantics

02 Concurrency and interleaving semantics

- Promela processes execute concurrently.
 - Non-deterministic scheduling of the processes.
 - Processes are interleaved (statements of different processes do not occur at the same time).
 - exception: rendez-vous communication.
- All statements are atomic; each statement is executed without interleaving with other processes.
- Each process may have several different possible actions enabled at each point of execution - only one choice is made, non-deterministically.
- InterleavingStatements.pml
 - Check
 - Random
 - 6 possibilities of the execution
 - n1,p,n2,q;
 - n1,n2,p,q;
 - n1,n2,q,p;
 - n2,q,n1,p;
 - n2,n1,q,p;
 - n2,n1,p,q.
 - Interactive simulation – Interactive button
- InterferenceBetweenProcesses.pml
- InterferenceBetweenProcessesDeterministic.pml

Examples

03 Critical section

- CriticalSection_Incorrect.pml
 - both processes – in the critical section
- CriticalSection_MutualExclusion.pml – not satisfied
 - Mutual exclusion – at most one process is executing its critical section at any time.
- CriticalSection_With_Deadlock.pml
 - Blocking on an expression – user Interactive simulation
 - Absence of deadlock – it is impossible to reach a state in which some processes are trying to enter their critical sections, but no process is successful.
- CriticalSection_SolutionAtomic.pml
 - The atomic sequence may be blocked from executing, but once it starts executing, both statements are executed without interference from the other process.

Linear Temporal Logic

- Temporal logic formulae can specify both safety and liveness properties.
- LTL \equiv propositional logic + temporal operators

$[] P$ always P

$<>P$ eventually P

$P \ U \ Q$ P is true until Q becomes true

Examples 04 LTL examples

- CriticalSection_MutualExclusionLTL.pml
 - LTL formula:
 - []mutex
 - Translate
 - Verify
- CriticalSection_MutualExclusionLTL02.pml
 - LTL formula:
 - []mutex
 - Translate
 - Verify
- CriticalSection_With_Starvation.pml
 - LTL formula:
 - <>csp
 - Translate
 - Acceptance
 - Verify

JSpin

- <http://spinroot.com/>
- Installation JSpin

<http://jspin.software.informer.com/5.0/>

Surprise!





JSpin

Take Home - Bonus

- PromelaMerryMe question
- Model a system with 2 actor: He/She
- He asks "Merry me" and She "finally" answers "YES".
- LTL property: question is addressed, then eventually YES will be answered
- 2 versions:
 - Use an intermediary state: notDecidedYet
 - Model that: She/He takes 3 seconds to answer.

- **25 XP**
 - Problem + LTL property checked

- Work in teams (2 students)
- Delivery: Lecture 11
- Deliverables
 - Promela program
 - Version 1 – with notDecidedYet
 - Version 2 - 3 seconds to answer
 - LTL properties

- **Study**
 - Lecture09-Demo
 - Lecture09-JSpin-install

Questions

- Thank You For Your Attention!

References Sources

[1] Baier Christel, Katoen Joost-Pieter, Principles of Model Checking , ISBN 9780262026499, The MIT Press, 2008

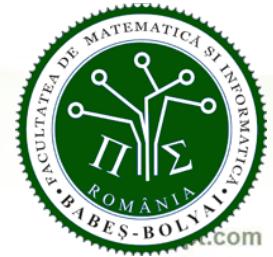
- Chapter 1 - System verification, Chapter 2 – Modelling Concurrent systems (pag. 19-20), Chapter 3 (pag. 89, 107, 120-121), Chapter 5 – Linear Temporal Logic (pag. 229-233), Chapter 6 – Computation Tree Logic (pag. 313-323)

[2] Ben-Ari, Mordechai, Principles of the Spin Model Checker, ISBN 978-1-84628-770-1, Springer-Verlag London, 2008

Software Systems Verification and Validation

Lecture 09: Model checking

Assoc. Prof. Andreea Vesca
Babeş-Bolyai University
Cluj-Napoca
2018-2019

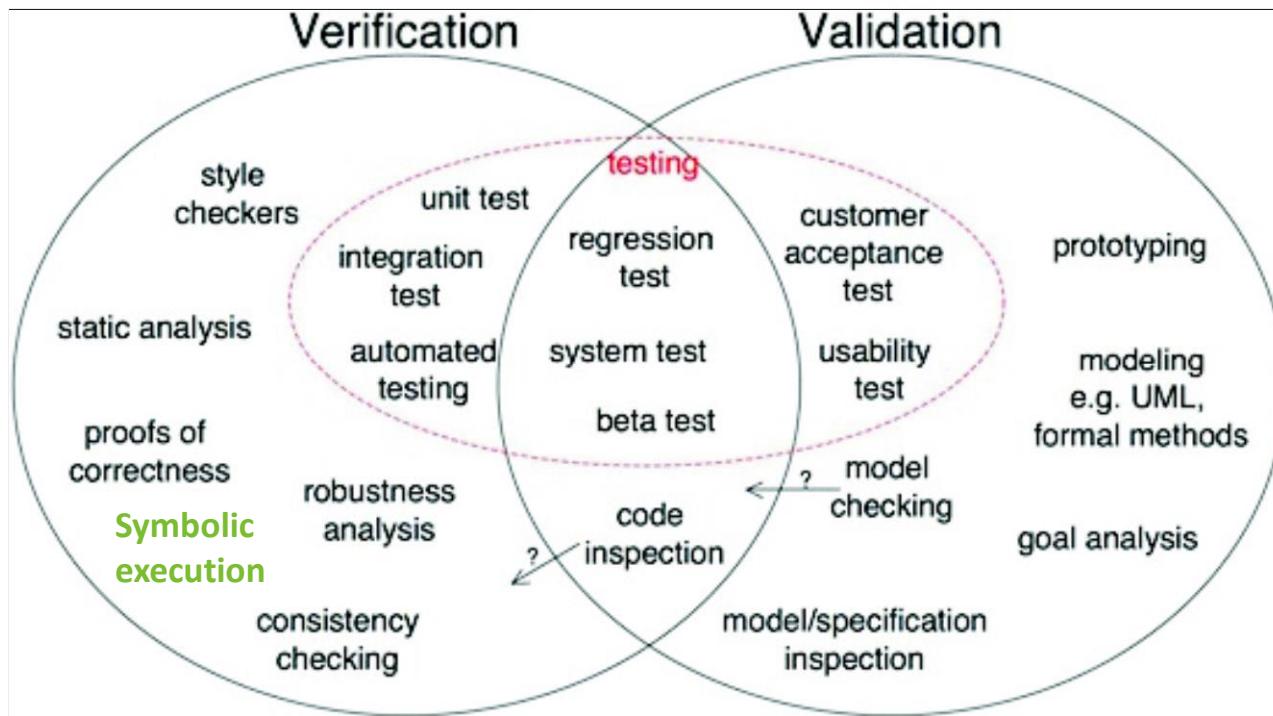


Outline

- System verification
- Model checking
- Transition system
- Linear-Time Properties
- Linear-Time Logic
- Computation Tree Logic
- Next lecture:
 - Spin Model Checker (still today!)
- Questions

Sales paradigm - SSVV

- Motivate the STUDENT - what you will learn!

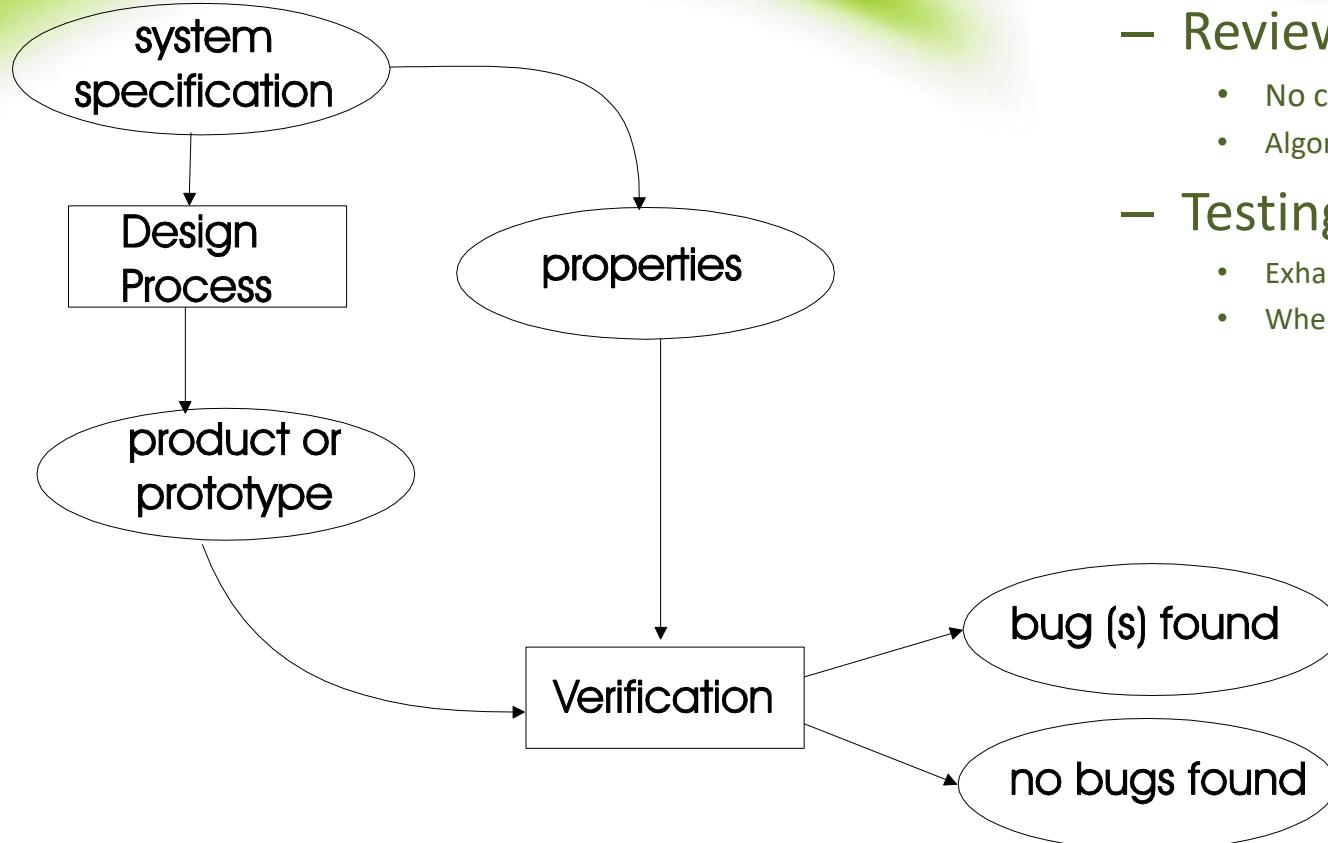


- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>

System verification (1)

- Information and Communication Technology (ICT)
- Correct ICT systems
 - It is all about money.
 - It is all about safety.
- Reliability of the ICT systems
 - Interactive systems - concurrency & nondeterminism
 - Pressure - to reduce system development time
- System verification techniques

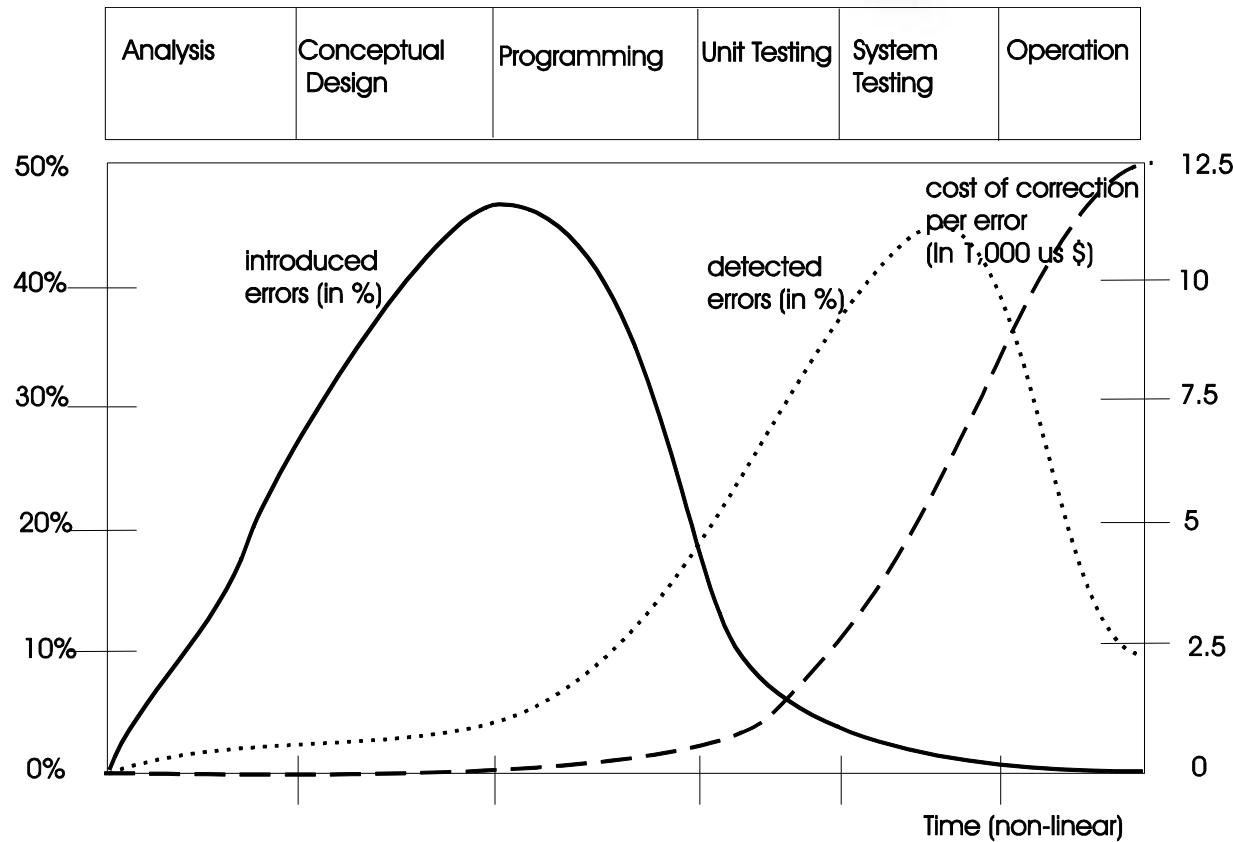
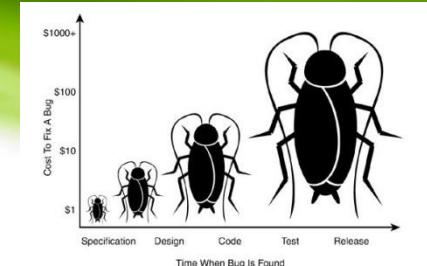
System verification (2)



- Software verification
 - Review
 - No concurrency defects
 - Algorithm defects
 - Testing
 - Exhaustive testing?
 - When to stop?

System verification (3)

- Catching software errors: the sooner the better



Model checking (1)

Formal methods

- More time and effort spend on verification than on construction
 - in software/hardware design of complex systems.
- The role of formal methods:
 - To establish system correctness with mathematical rigor.
 - To facilitate the early detection of defects.
- Verification techniques
 - Testing – small subset of paths is treated
 - Simulation - restrictive set of scenarios in the model
 - Model checking - exhaustive exploration
- **Remark.** Any verification using **model-based techniques** is only as good as the model of the system.

Model checking (1)

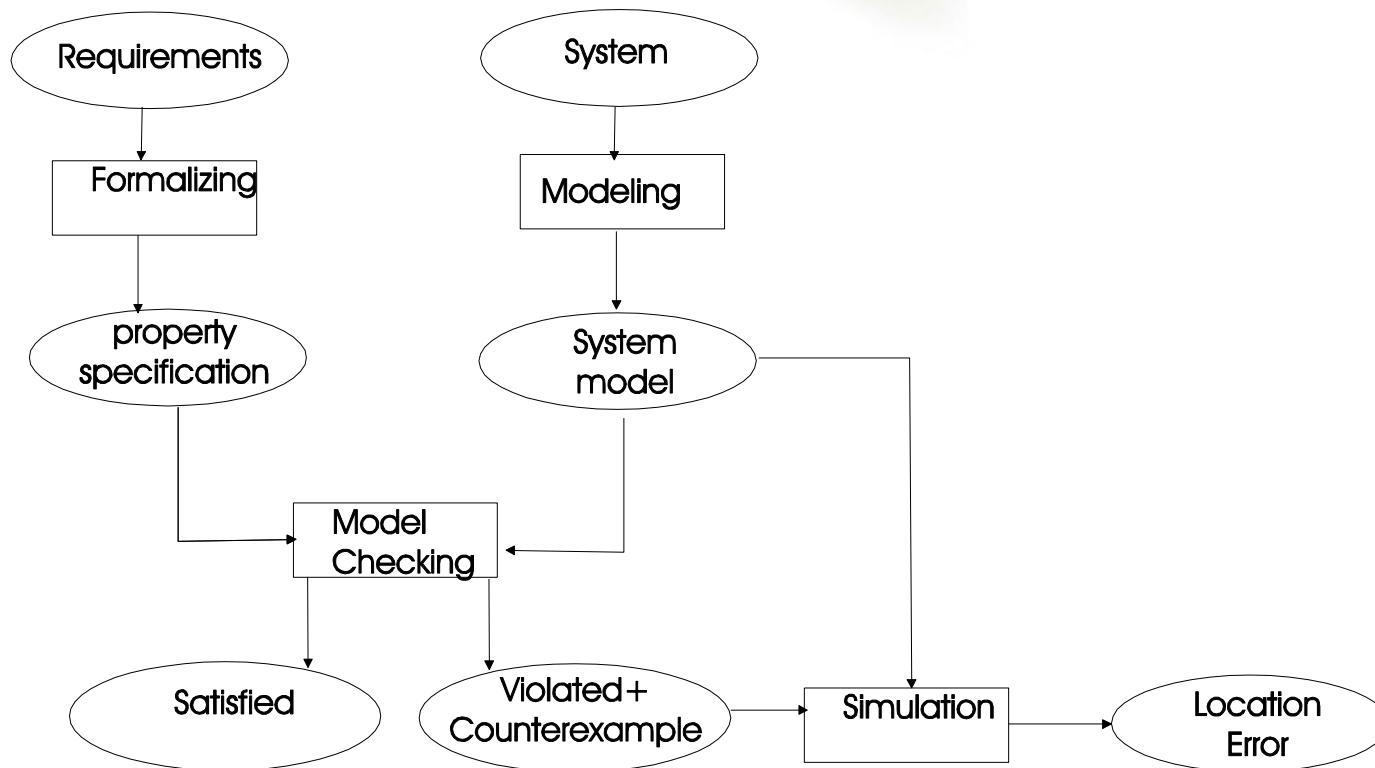
Formal methods



- Mechanical Engineering is like looking for a black cat in a lighted room.
- Chemical Engineering is like looking for a black cat in a dark room.
- Software Engineering is like looking for a black cat in a dark room in which there is no cat.
- Systems Engineering is like looking for a black cat in a dark room in which there is no cat and someone yells, “I got it!”

Model checking (2)

Approach



Model checking (3)

Characteristics

- Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.
- The model checking process
 - Modeling phase
 - model the system under consideration
 - formalize the property to be checked.
 - Running phase
 - Analysis phase
 - property satisfied?
 - property violated?

Model checking (4)

Strengths and Weaknesses

Strengths

- General verification approach
- Supports partial verification
- Provides diagnostic information
- Potential “push-button” technology
- Increasing interest by industry
- Easily integrated in existing development cycles

Weaknesses

- Appropriate to control-intensive applications
- Its applicability is subject to decidability issues
- It verifies a system model
- Checks only stated requirements
- Suffers from the state-space explosion problem
- Requires some expertise

Transition system (1)

Definition

- Transition systems - used in computer science as models to describe the behavior of the systems.
- Transition systems - directed graphs:
 - Nodes - represent states;
 - Edges - model transitions, i. e. state changes.
- A Transition System (TS) is tuple $(S, \text{Act}, \rightarrow, I, \text{Ap}, L)$, where
 - S is a set of states,
 - Act is a set of actions,
 - $\rightarrow \subseteq S \times \text{Act} \times S$ is a transition relation,
 - $I \subseteq S$ is a set of initial states,
 - AP is a set of atomic propositions, and
 - $L : S \rightarrow 2^{\text{AP}}$ is a labeling function.
- TS is called finite if S , Act and AP are finite.

Transition system (2)

Remarks

- Intuitive behavior of a transition system
 - Initial state $s_0 \in I$
 - Using the transition relation \rightarrow the system evolves
 - Current state s , a transition $s \xrightarrow{\alpha} s'$ is selected *nondeterministically*
 - The selection procedure is repeated and finishes once a state is encountered that has no outgoing transitions.
- The labeling function L relates a set $L(s) \subseteq 2^{AP}$ at atomic propositions to any state s . $L(s)$ intuitively stands for exactly those atomic propositions $a \in AP$ which are satisfied by state s .
- Given that ϕ is a propositional logic formula, then s satisfies the formula ϕ if the evaluation induced by $L(s)$ makes the formula ϕ true,

$$s \models \phi \text{ iff } L(s) \models \phi.$$

Transition system (3)

Example

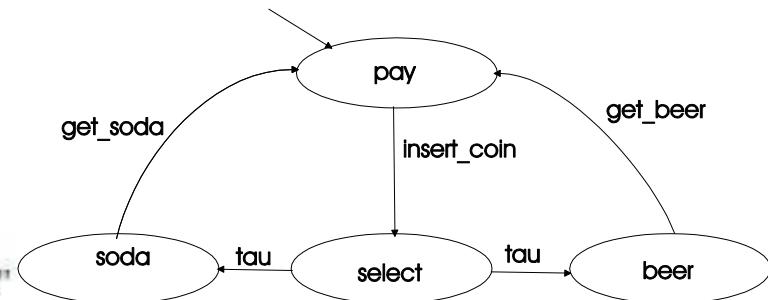
Beverage Vending Machine

- $S = \{pay, select, soda, beer\}$, $I = \{pay\}$
- $Act = \{insert_coin, get_soda, get_beer, \tau\}$
- Example transitions: $pay \xrightarrow{insert_coin} select$, $beer \xrightarrow{get_beer} pay$
- Atomic propositions depends on the properties under consideration.

A simple choice - to let the state names act as atomic propositions, i. e. $L(s) = \{s\}$.

"The vending machine only delivers a drink after providing a coin."

$AP = \{paid, drink\}$, $L(pay) = \emptyset$, $L(soda) = L(beer) = \{paid, drink\}$, $L(select) = \{paid\}$.



Linear-Time Properties

- **Deadlock** – if the complete system is in a terminal state, although at least one component is in a (local) nonterminal state.
 - A typical deadlock scenario occurs when components mutually wait for each other to progress.
- **Safety properties** = “nothing bad should happen”.
 - The number of inserted coins is always at least the number of dispensed drinks.
 - A typical safety property is deadlock freedom
 - Mutual exclusion problem – “bad” = more than one process is in the critical section
- **Liveness properties** = “something good will happen in the future”.
 - Mutual exclusion problem – typical liveness properties assert that:
 - (eventually) – each process will eventually enter its critical section
 - (repeated eventually) = each process will enter its critical section infinitely often
 - (starvation freedom) – each waiting process will eventually enter its critical section
- **Remark**
 - **Safety properties** - are violated in finite time (a finite system run)
 - **Liveness properties** – are violated in infinite time (by infinite system runs)

Temporal Logic

- **Propositional temporal logics** - extensions of propositional logic by temporal modalities.
- The elementary temporal modalities that are present in most temporal logics include the operators
 - “**eventually**” (eventually in the future) - \diamond
 - “**always**” (now and forever in the future – \square
- The nature of time in temporal logics can be either **linear or branching**.
- The adjective “temporal”
 - specification of the relative order of events
 - does not support any means to refer to the precise timing of events

Linear-Time Logic (1)

Syntax of LTL

- Construction of LTL formulae in LTL - ingredients:
 - atomic propositions $a \in AP$, (stands for the state label a in a transition system)
 - boolean connectors like conjunction \wedge and negation \neg ,
 - basic temporal modalities "next" \bigcirc and "until" \bigcup .
- LTL formulae over the set AP of atomic proposition are formed according to the following grammar:
 $\varphi ::= true | a | \varphi_1 \wedge \varphi_2 | \neg \varphi | \bigcirc \varphi | \varphi_1 \bigcup \varphi_2$, where $a \in AP$.

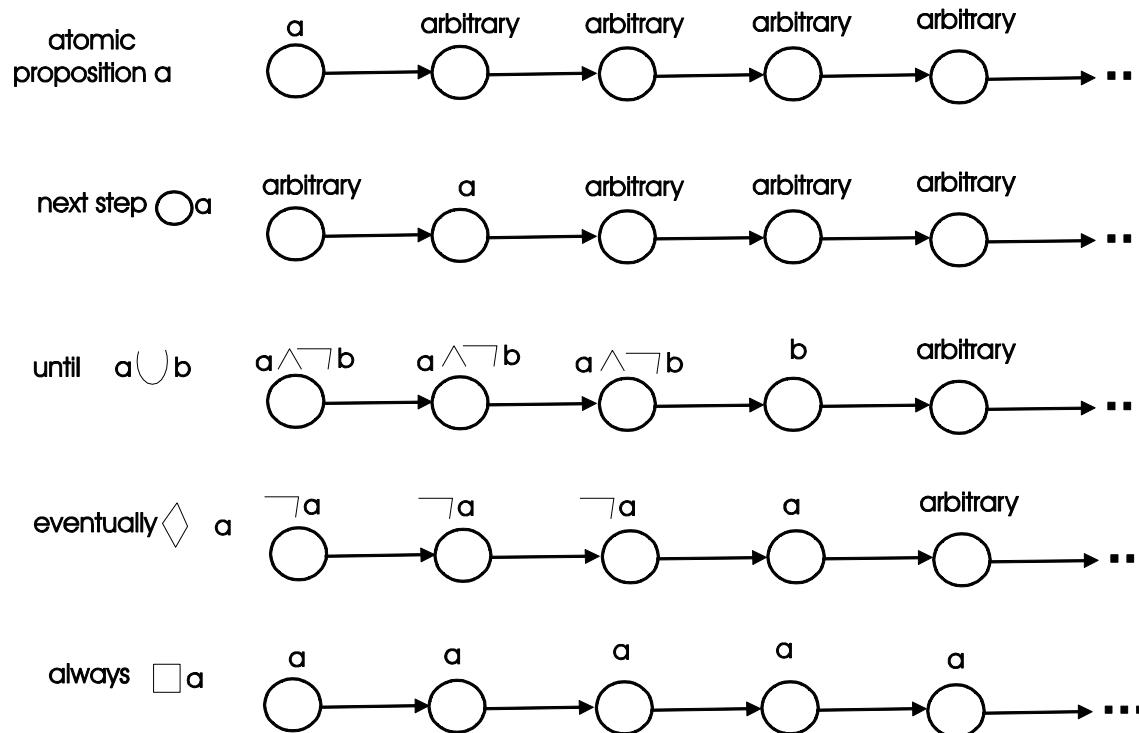
Linear-Time Logic (2)

LTL temporal modalities

- The until operator allows to derive the temporal modalities \diamond ("eventually", sometimes in the future) and \square ("always", from now on forever) as follows:
 - $\diamond\varphi = \text{true} \cup \varphi$.
 - $\square\varphi = \neg\diamond\neg\varphi$.
- By combining the temporal modalities \diamond and \square , new temporal modalities are obtained:
 - $\square\diamond\varphi$ - "infinitely often φ ."
at any moment j there is a moment i $i \geq j$ at which an a state is visited
 - $\diamond\square\varphi$ - "eventually forever φ ."
from some moment j on, only a -states are visited.

Linear-Time Logic (3)

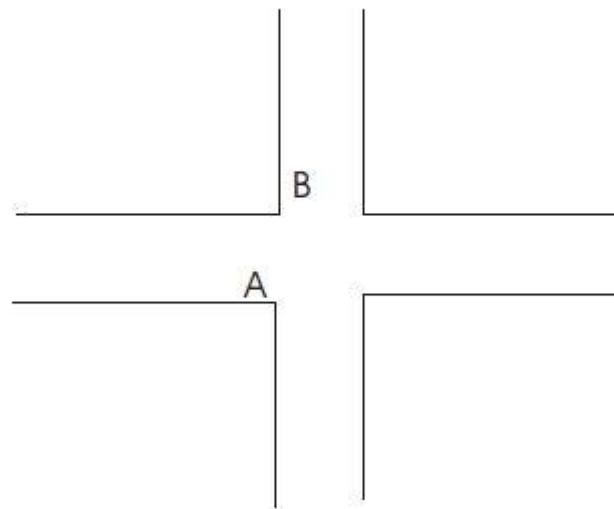
Intuitive meaning of temporal modalities



Linear-Time Logic (4)

LTL semaphore example

- $\square(\neg(A = \text{green} \wedge B = \text{green}))$
 - A and B can not be simultaneously green.
- $\square(A = \text{yellow} \rightarrow A = \text{red})$
 - If A is yellow eventually will become red.
- $\square(A = \text{yellow} \rightarrow \bigcirc(A = \text{red}))$
 - If A is yellow then it will be red into the next state.
- $\square(\neg(B = \text{green}) \bigcup(A = \text{red}))$
 - B will not be green until A changes in red.



Computation Tree Logic (1)

Syntax of CTL

- Construction of CTL formulae:
 - as in LTL by the next-step and until operators,
 - must be not combined with boolean connectives
 - no nesting of temporal modalities is allowed.
- CTL formulae over the set AP of atomic proposition are formed according to the following grammar:
$$\phi ::= \text{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \exists \phi \mid \forall \phi,$$
where $a \in AP$ and φ is a path formula.
- CTL path formulae are formed according to the following grammar:
$$\varphi ::= \bigcirc \phi \mid \phi_1 \bigcup \phi_2,$$
where ϕ, ϕ_1 and ϕ_2 are state formulae.

Computation Tree Logic (2)

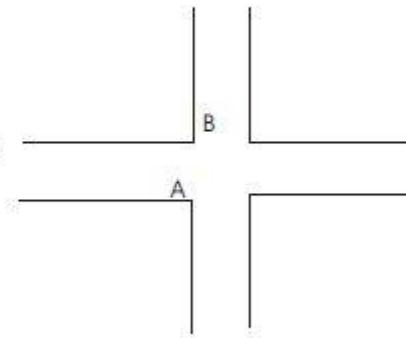
CTL - state and path formulae

- CTL distinguishes between state formulae and path formulae:
 - State formulae express a property of a state.
 - Path formulae express a property of a path, i.e. an infinite sequence of states.
- Temporal PATH operators \bigcirc and \bigcup
 - $\bigcirc\phi$ holds for a path if ϕ holds in the next state of the path;
 - $\phi \bigcup \psi$ holds for a path if there is some state along the path for which ψ holds, and ϕ holds in all states prior to that state.
- Path formulae \Rightarrow state formulae by prefixing them with
 - path quantifier \exists (pronounced "for some path");
 $\exists\phi$ - holds in a state if there exists some path satisfying ϕ that starts in that state.
 - path quantifier \forall (pronounced "for all paths".)
 $\forall\phi$ -holds in a state if all paths that start in that state satisfy ϕ .

Computation Tree Logic (3)

CTL semaphore example

- $\forall \square(B = \text{yellow} \rightarrow \forall \bigcirc(B = \text{red}))$.
 - If B is yellow, it will become (sometime in the future) red.



Surprise!

Model checking

3-5 minutes

Formative Assessment

Anonymous voting

Next Lecture (Still today!)

- JSpin

Questions

- Thank You For Your Attention!

References

Sources

[1] Baier Christel, Katoen Joost-Pieter, Principles of Model Checking , ISBN 9780262026499, The MIT Press, 2008

- Chapter 1 - System verification, Chapter 2 – Modelling Concurrent systems (pag. 19-20), Chapter 3 (pag. 89, 107, 120-121), Chapter 5 – Linear Temporal Logic (pag. 229-233), Chapter 6 – Computation Tree Logic (pag. 313-323)



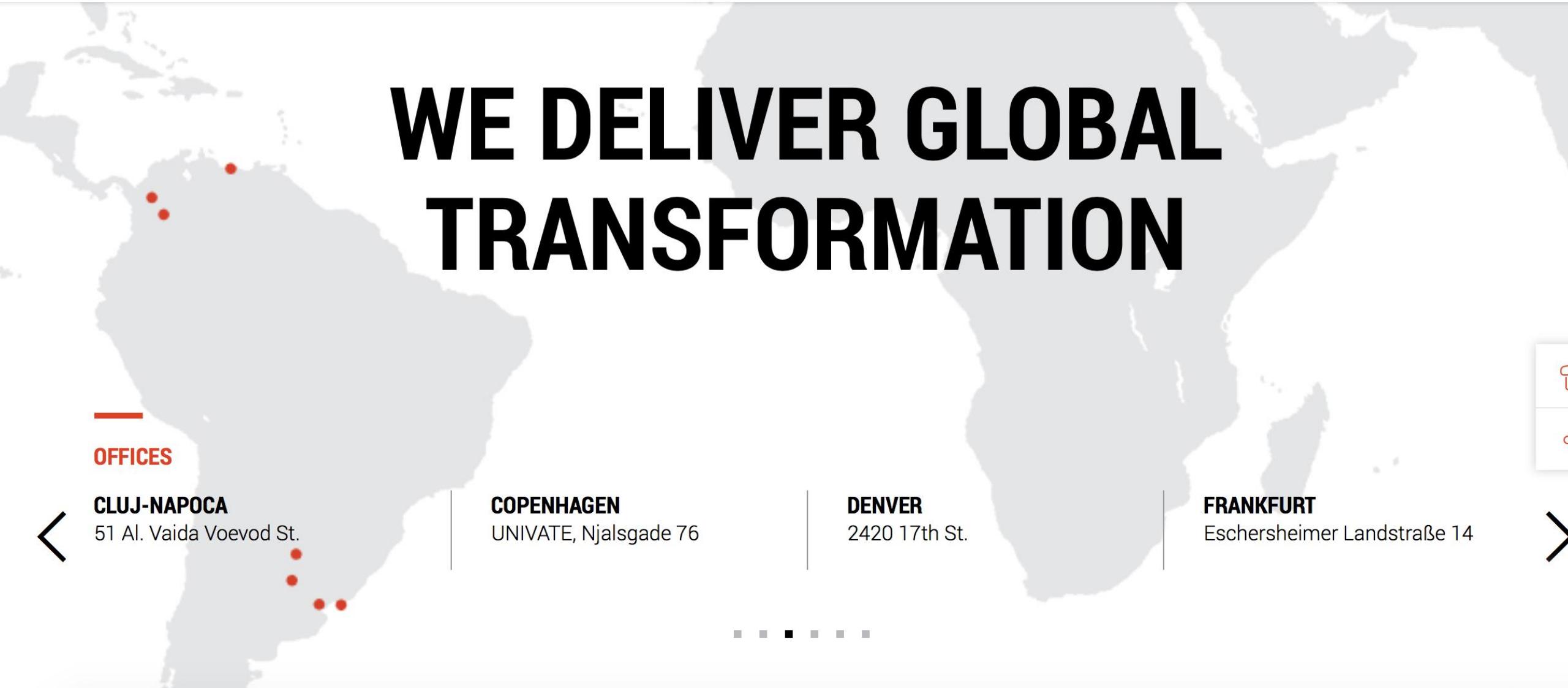
VERIFICATION & VALIDATION DURING THE SOFTWARE DEVELOPMENT LIFE CYCLE

PRESENTERS:
ROXANA ONAC

AGENDA

- **INTRODUCTION**
- **QUALITY ASSURANCE & QUALITY CONTROL (QA & QC)**
- **QUALITY MANAGEMENT SYSTEMS (QMS)**
- **CONTINUOUS IMPROVEMENT (CI)**
- **REQUIREMENTS/ DESIGN**
- **TEST PLANNING**
- **TEST DESIGN**
- **TEST EXECUTION**
- **TEST REPORTING**

WHAT DO YOU KNOW ABOUT ENDAVA?



WE DELIVER GLOBAL TRANSFORMATION

OFFICES**CLUJ-NAPOCA**

51 Al. Vaida Voevod St.

COPENHAGEN

UNIVATE, Njalsgade 76

DENVER

2420 17th St.

FRANKFURT

Eschersheimer Landstraße 14



—

PHILOSOPHY

OUR PHILOSOPHY IS SIMPLE

"We focus on helping people succeed. The people who work for us, the people who engage with us, and the people who use the systems and applications we design, build, and operate."

JOHN COTTERELL, CHIEF EXECUTIVE, ENDAVA



BE MORE

we're prodigies and hard workers, peace lovers and fighters,
we're designers and project managers, automation testers and architects,
we're leaders and rebels, skaters and bikers,
but above all, we are people who believe they can be more

[See Videos >](#)



OUR AREA OF EXPERTISE



QA & QC

QUALITY ASSURANCE AND QUALITY CONTROL

WHAT IS THE DIFFERENCE?

QA & QC



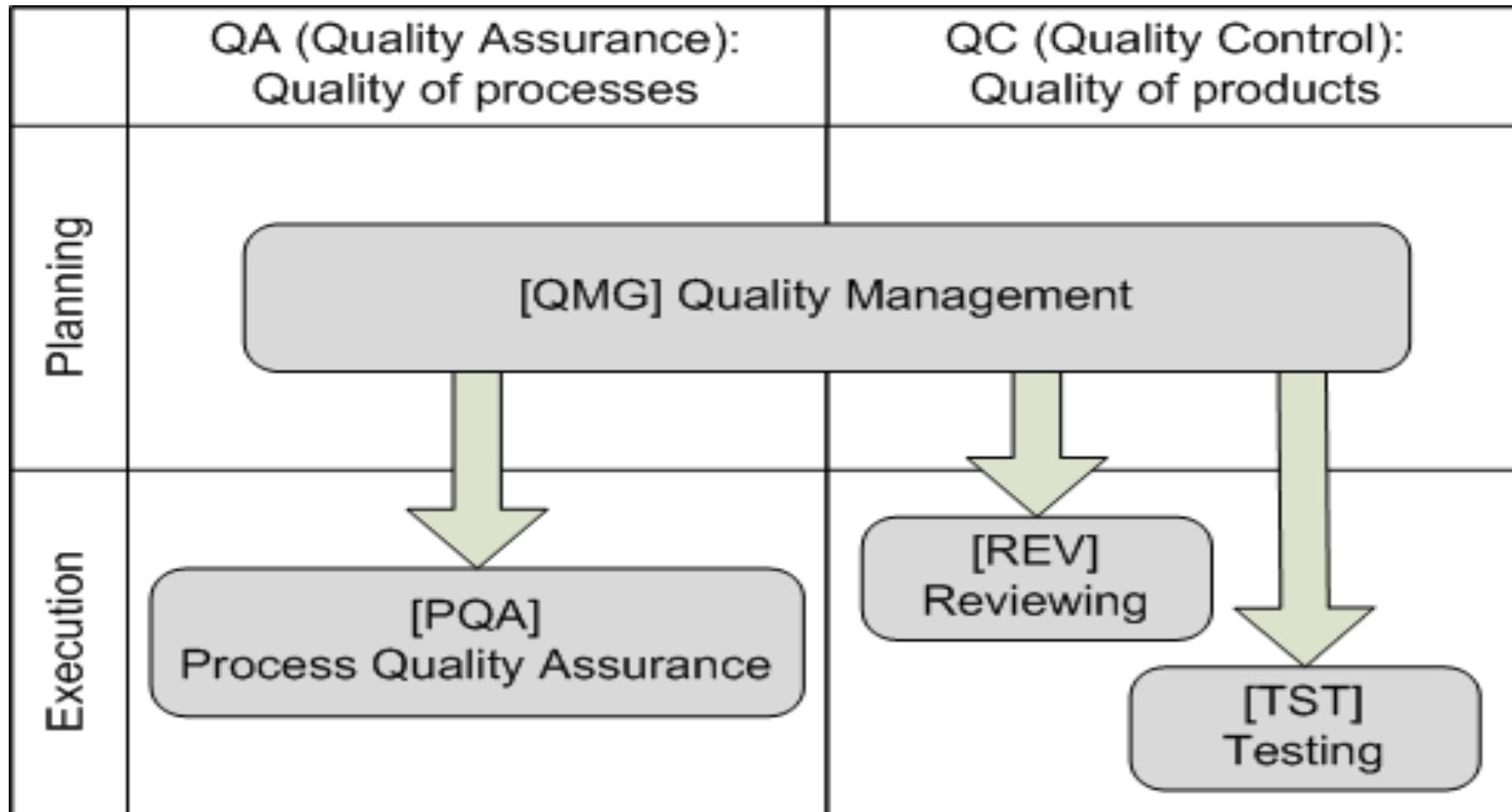
QMS

A QUALITY MANAGEMENT SYSTEM (QMS)
IS A SET OF POLICIES, PROCESSES AND PROCEDURES
REQUIRED FOR PLANNING AND EXECUTION
(PRODUCTION/DEVELOPMENT/SERVICE) IN THE CORE
BUSINESS AREA OF AN ORGANIZATION

QMS



QMS



MEASURING QUALITY

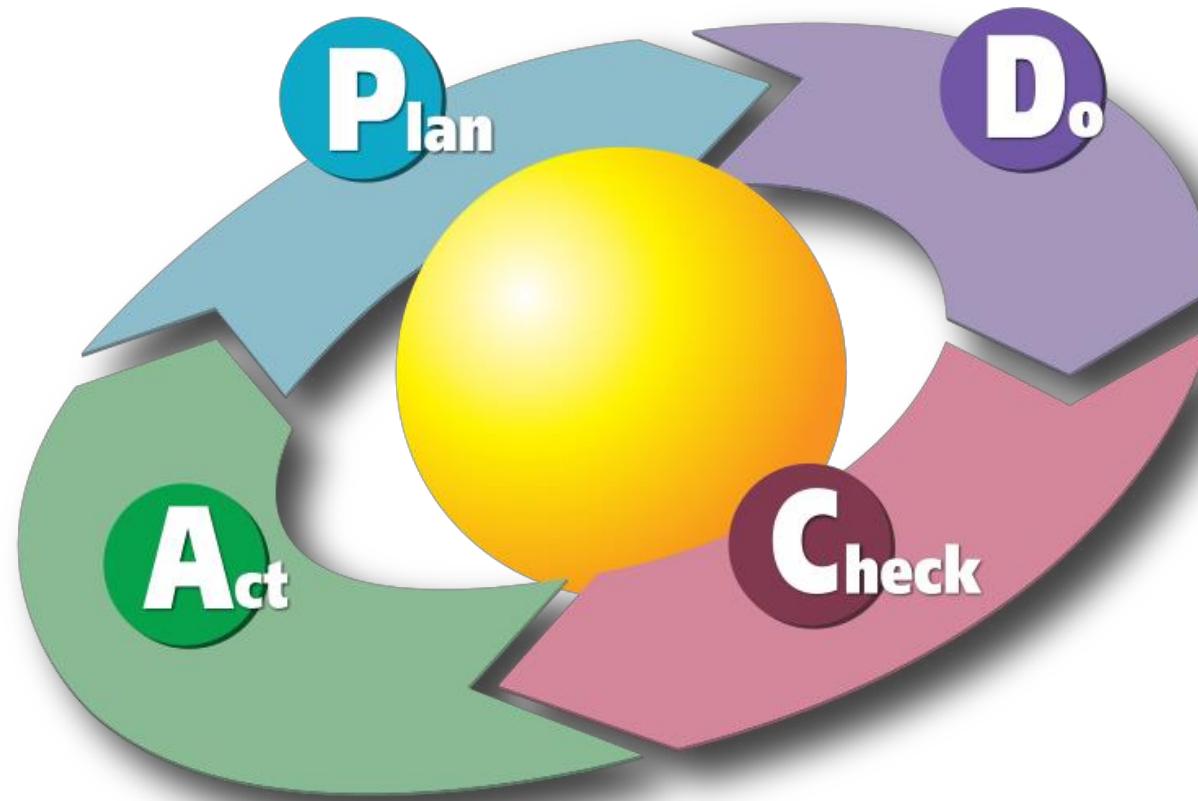
- **IF YOU CANNOT DEFINE IT, YOU CANNOT ACHIEVE IT**
- **IF YOU CANNOT MEASURE IT**

You do not know how you are progressing

You do not know when you have arrived

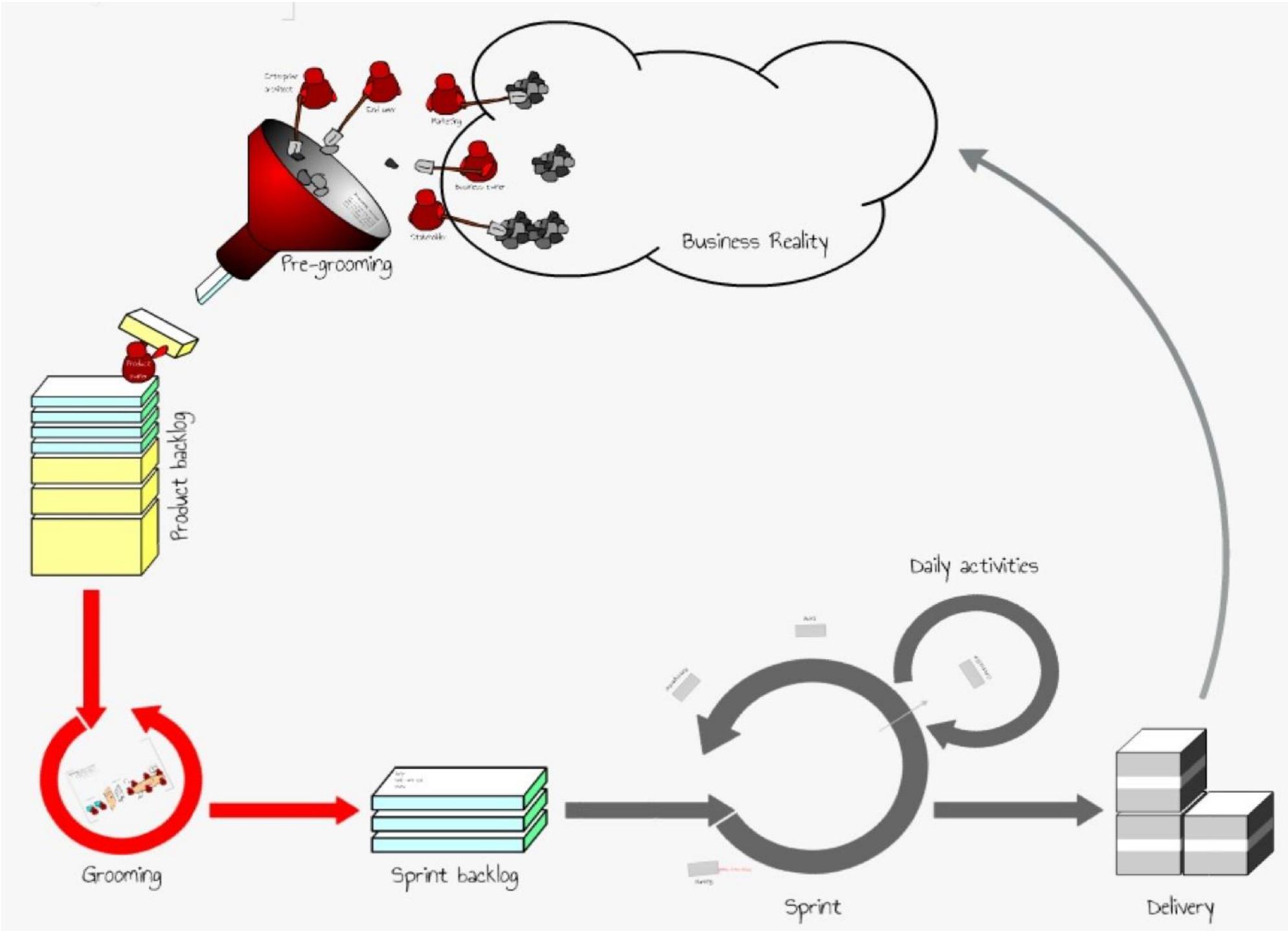
You cannot demonstrate it

CI



- **MAIN DRIVER WAS THE VERY DIVERSIFIED WAY OF WORKING:**
 - Multiple technologies
 - Multiple methodologies
 - Various level of client control
 - Locations
 - Company growth, etc

- **MAIN GOAL IS TO ENSURE COMPANY-WIDE SPREAD OF**
 - Lessons learned
 - Best practices
 - Ensure a uniformed way-of-working between projects



SUMMARY

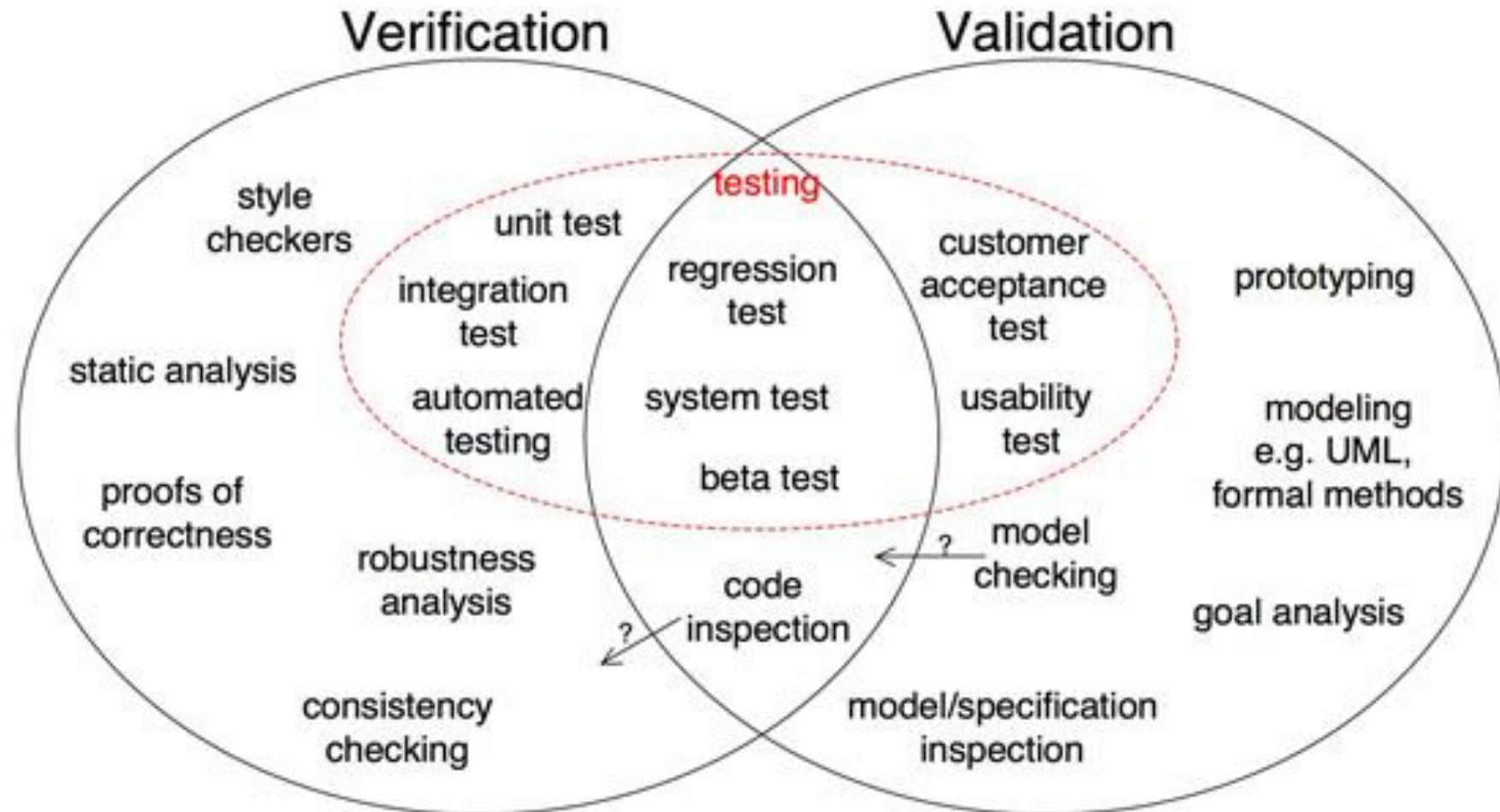
- QA & QC GIVE THE **BEST RESULTS** TAKEN TOGETHER
- **CUTTING CORNERS IS NOT AN OPTION**
- **DELIVERING QUALITY IS EVERYBODY'S JOB**
- YOU HAVE TO DEFINE "**WHAT IS QUALITY**" IN ORDER TO ACHIEVE IT
- **CONTINUOUS IMPROVEMENT**
- QA PROCESSES MUST BE PRESENT IN EACH STAGE OF THE **DEVELOPMENT LIFECYCLE**

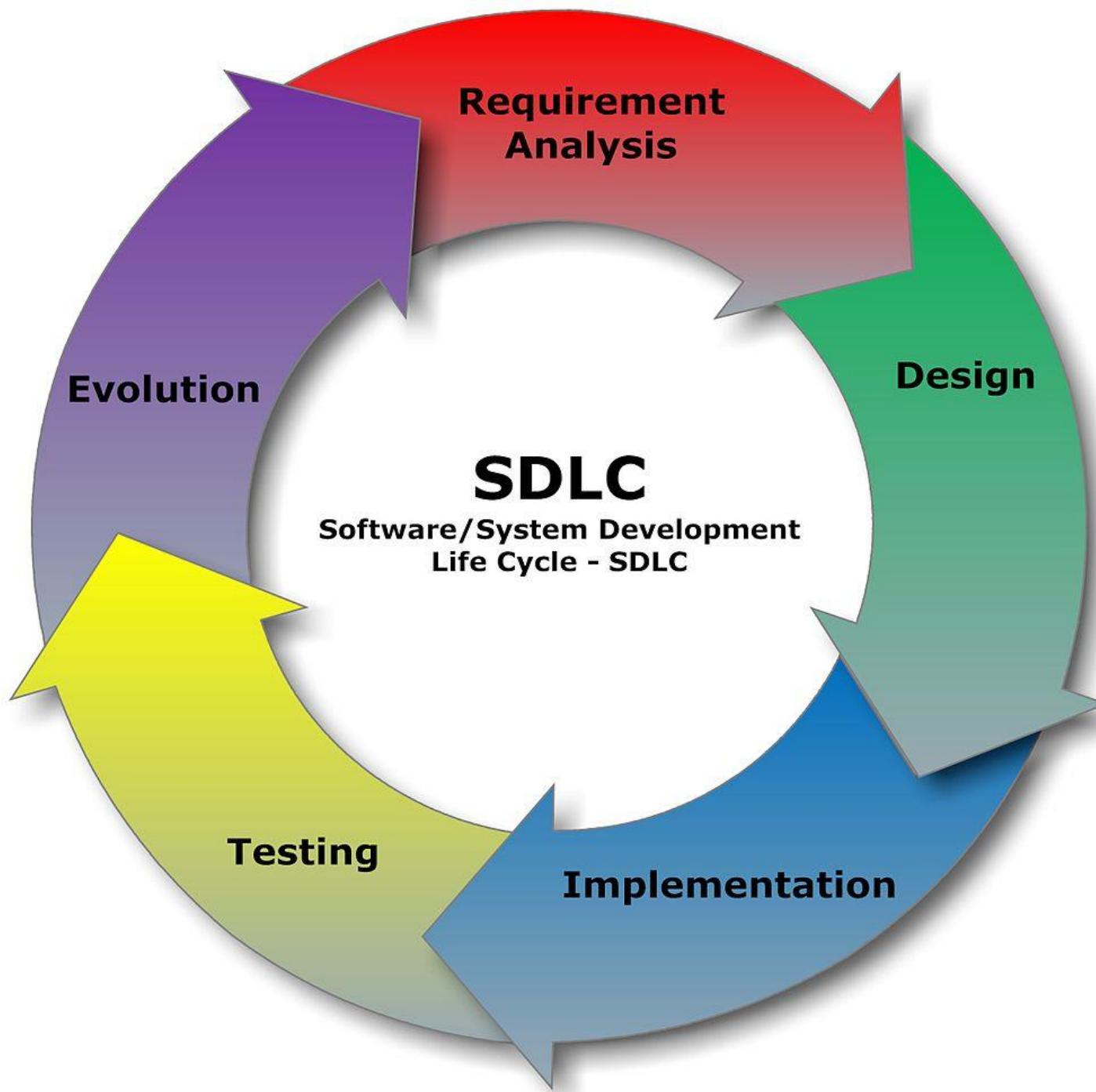
ASK QUESTIONS

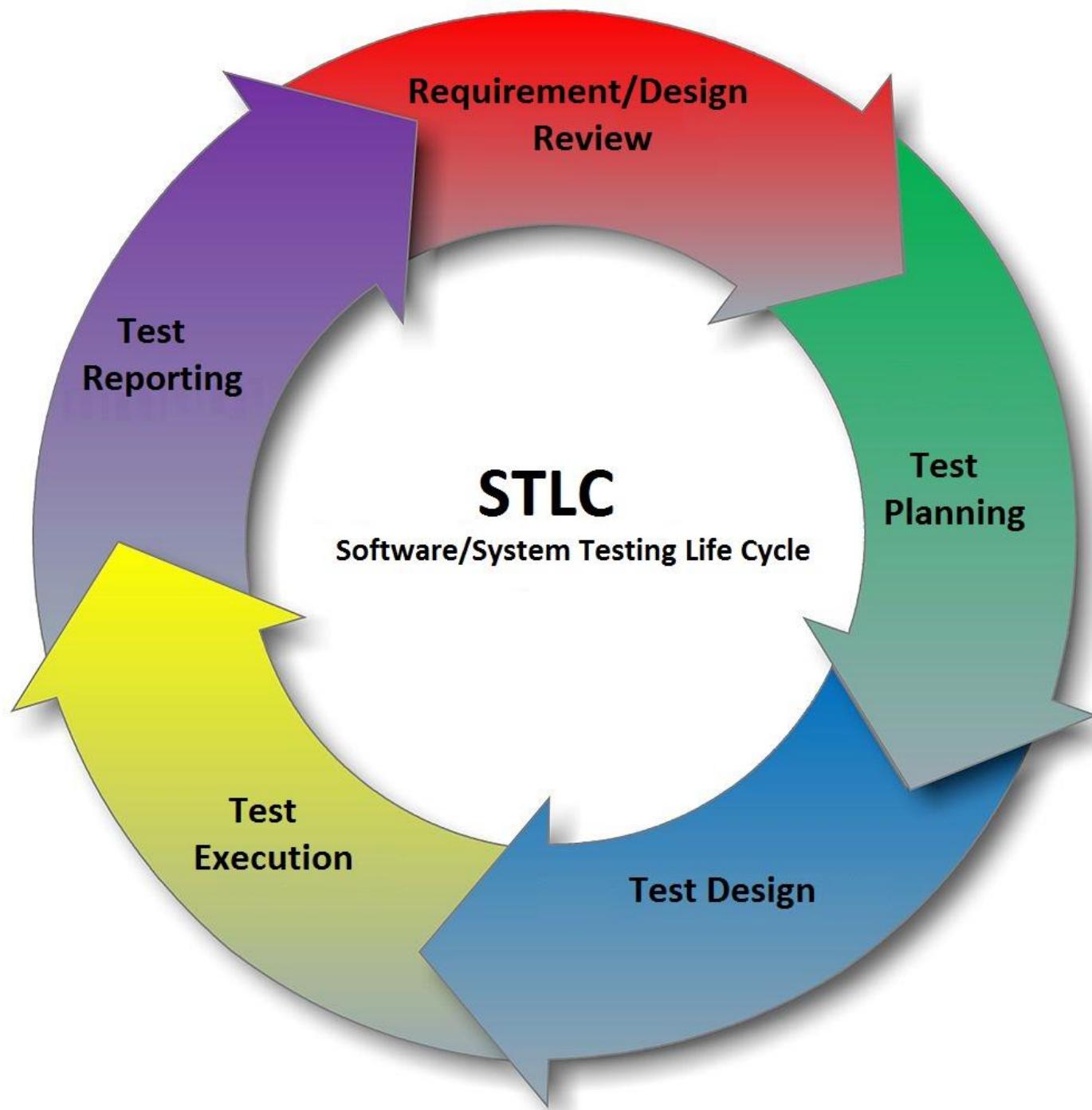


WHY IS SOFTWARE TESTING NECESSARY?

VERIFICATION & VALIDATION



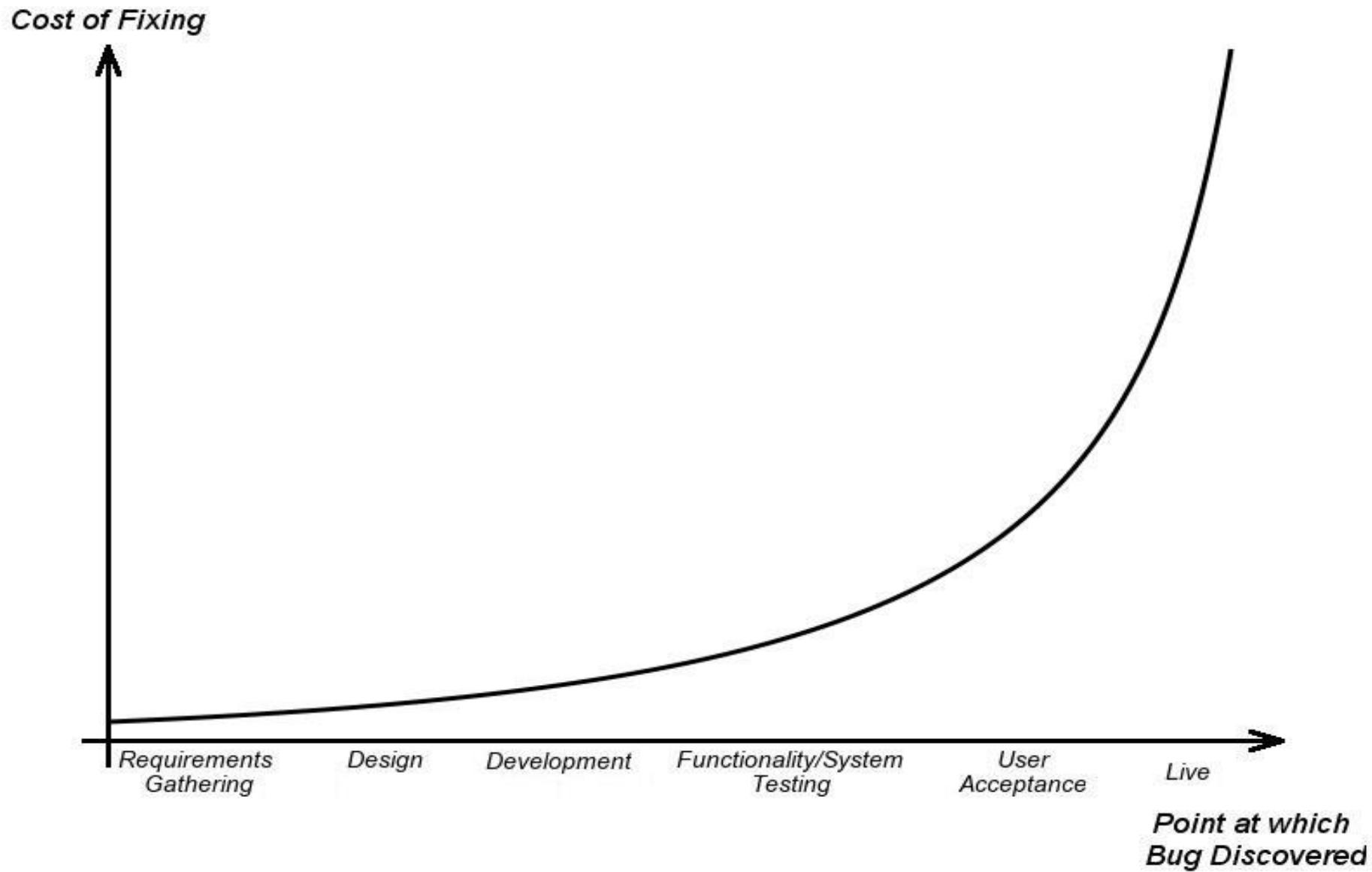


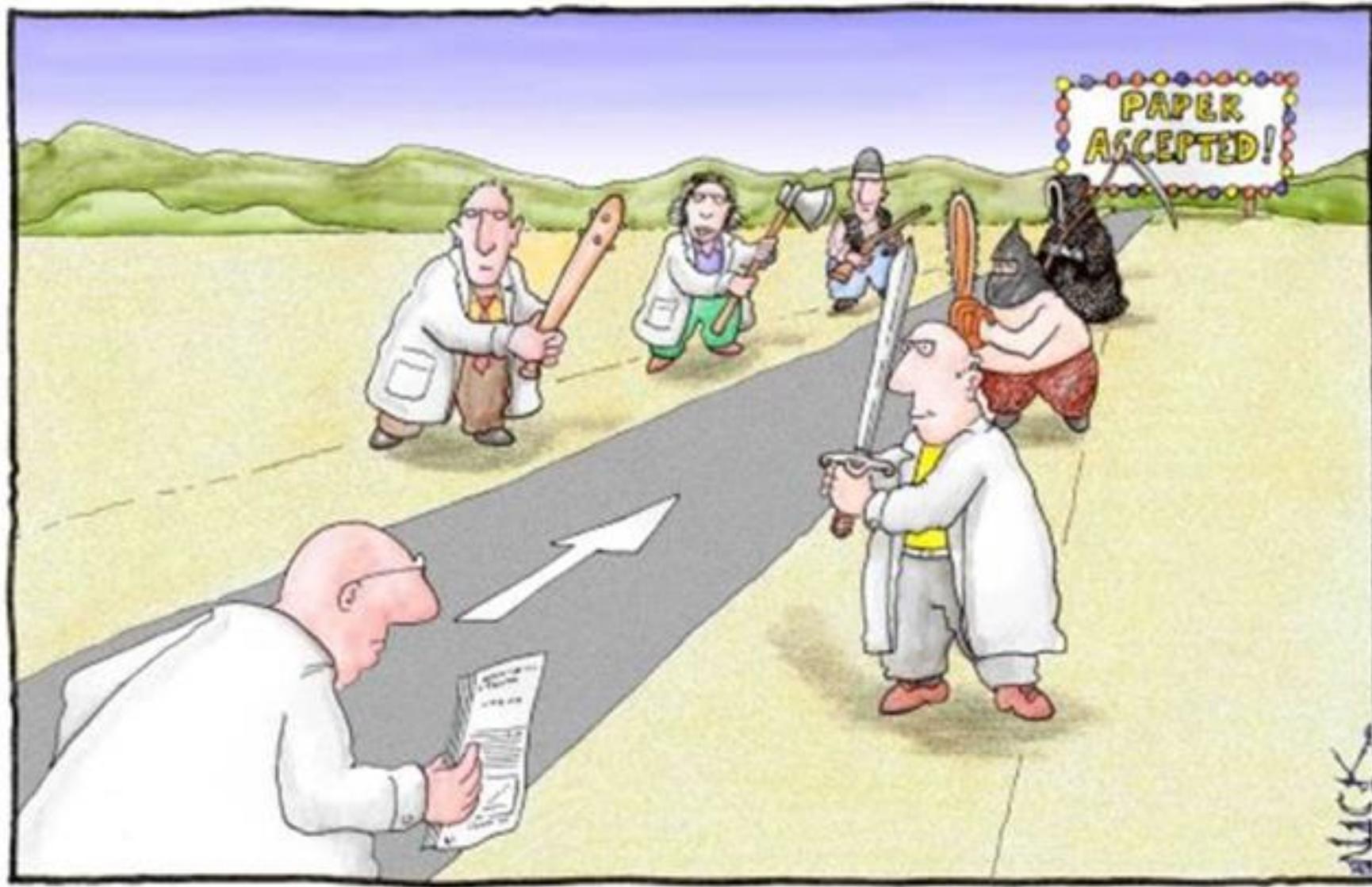


REQUIREMENTS/ DESIGN/ REVIEW

WE ARE FINDING A DEFECT IN REVIEW 9 TIMES FASTER THAN IN TESTING.

**WE ARE SOLVING A DEFECT FOUND IN REVIEW 5 TIMES FASTER THAN A
DEFECT FOUND IN TESTING.**





TEST PLANNING

TESTING TYPES

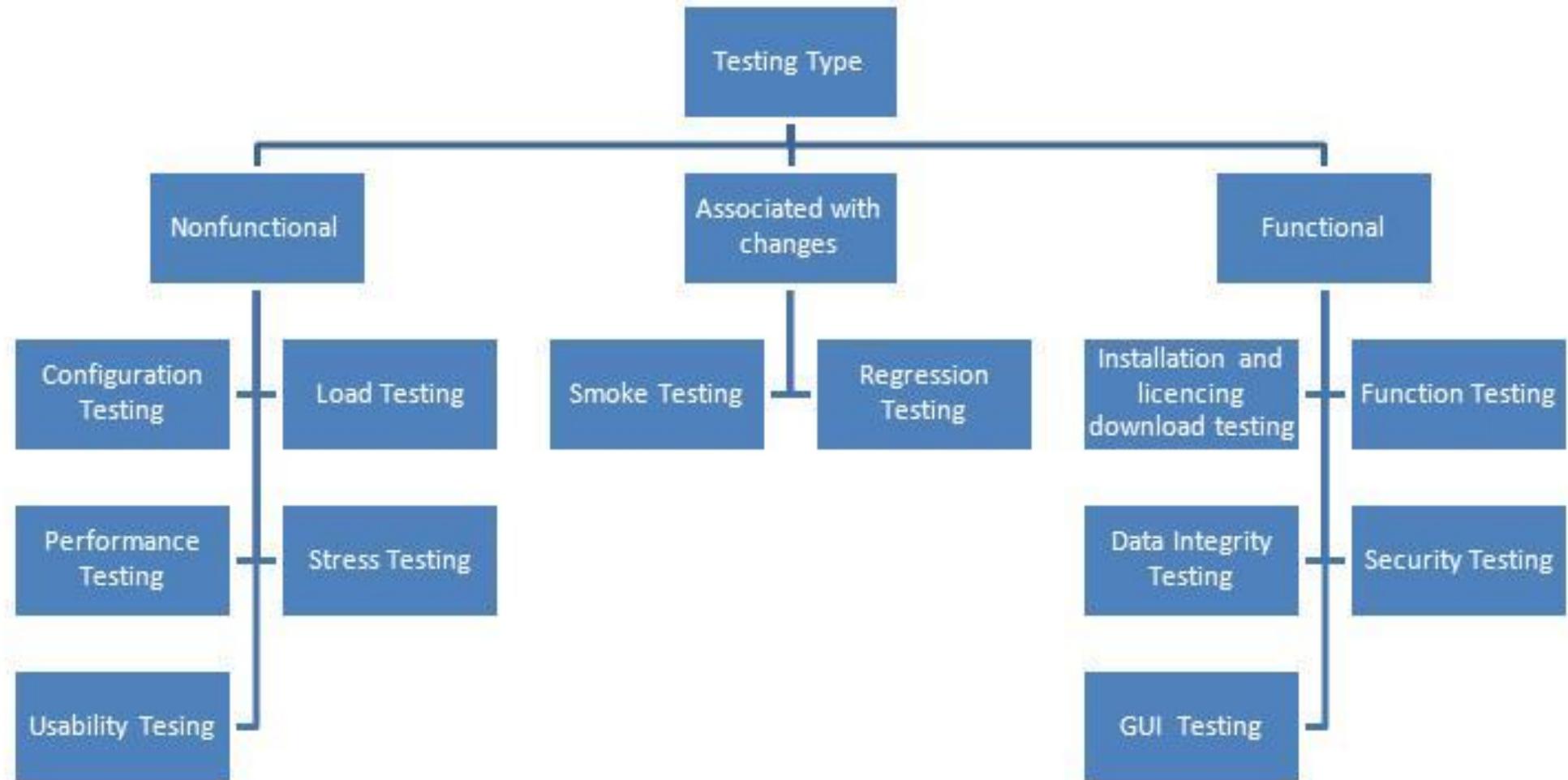
TESTING LEVELS

SCOPE

ACTIVITIES

DELIVERABLES

TESTING TYPES



TESTING LEVELS

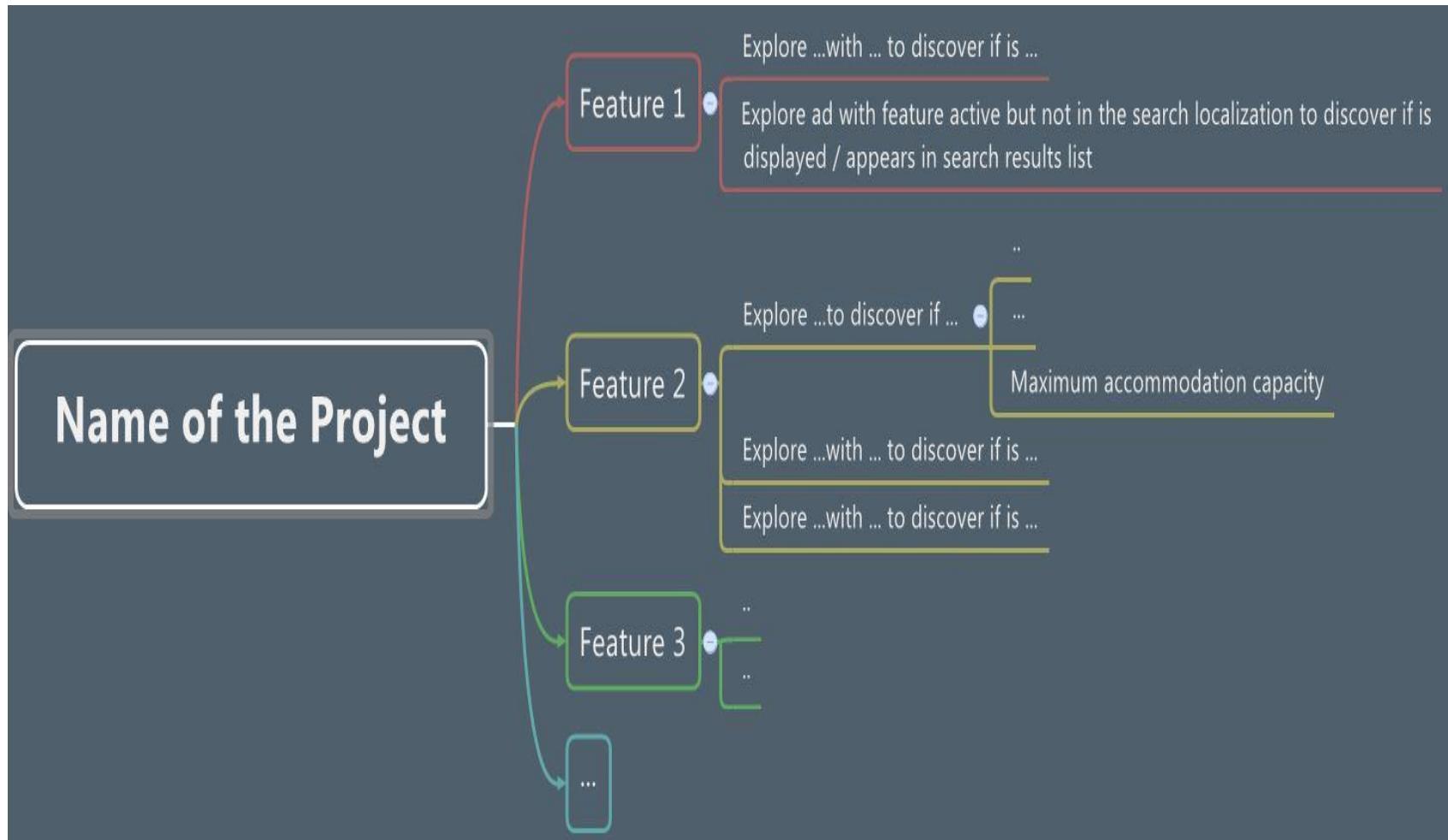
		Unit tests	System tests	Integration tests	Acceptance tests
Responsible:		Endava	Endava	Endava	Client
Test types	Functional tests	•	•	•	•
	Functional: Security		•		
	Non-Funct: Usability		•		•
	Non-Funct: Load, stress & performance		•		
	Regression tests		•	•	

TEST DESIGN

TOOLS

SCOPE
ACTIVITIES
DELIVERABLES

TEST SCENARIO



Version 1

Created on 13/09/2012 09:01:58 by florin.sibieanu

Last modified on 10/01/2013 14:34:14 by florin.sibieanu

Summary

Verify that user can't create a new organization setting the flags Trust Center false; Scoring center nothing checked ; Health Self Management false ; Initial Call false ; DefCoach=0 but i can save if it creates and then adds a new default coach (DefCoach=1).

This are the settings for Moove organization

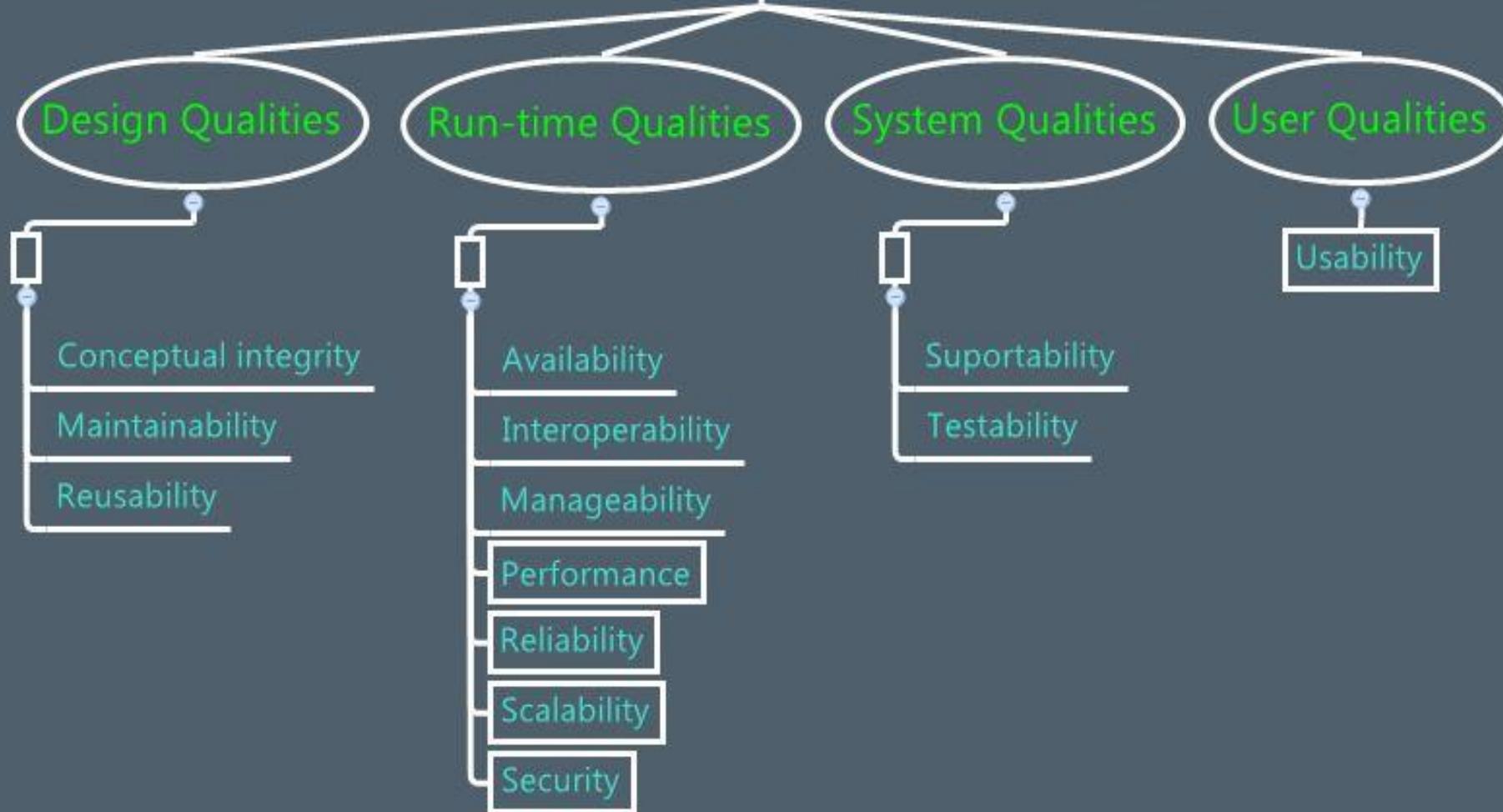
Preconditions

Admin credential should be known before starting the test

#	Step actions	Expected Results	Execution
1	Login using the admin credential Click on Control Panel Under "Portal" section click on "User and Organizations" link Click on Add button and select "Regular organizations" (from the dropdown) Fill in Name and press Save	Admin home page should load Liferay control panel page should load "User and Organizations" page should load Add "New Organization" page should load New organization should be created (a message should be displayed: "Your request completed successfully "); custom field should appear under "Organization Information" page	Manual  
2	Click on "Custom fields: under "Organization Information" section	Custom field page should load	Manual  
3	Set the fields Trust Center to false; Scoring center nothing checked; Health Self Management false ; Initial Call false and then press Save (at this moment there is no default coach present on that organization)	An error should be shown a this configuration can't be saved	Manual  
4	Under "Portal" section click on "User and Organizations" Click on the organization created before Add a new default coach for that organization	"User and Organization" page should load Organization page should load New default coach for that organization should be created	Manual  
5	Click on Edit organization button and then on "Custom Fields"	Organization's custom field page should load	Manual  
6	Set the fields Trust Center to false; Scoring center nothing checked; Health Self Management false ; Initial Call false and then press Save (at this moment there is a default coach present on that organization)	This configuration should be saved	Manual  

Create step**Execution type :** Manual**Test importance :** High**Review Remarks:**

Common Quality Attributes



TEST EXECUTION

TYPES AND LEVELS – START DOING

REGRESSION TESTING

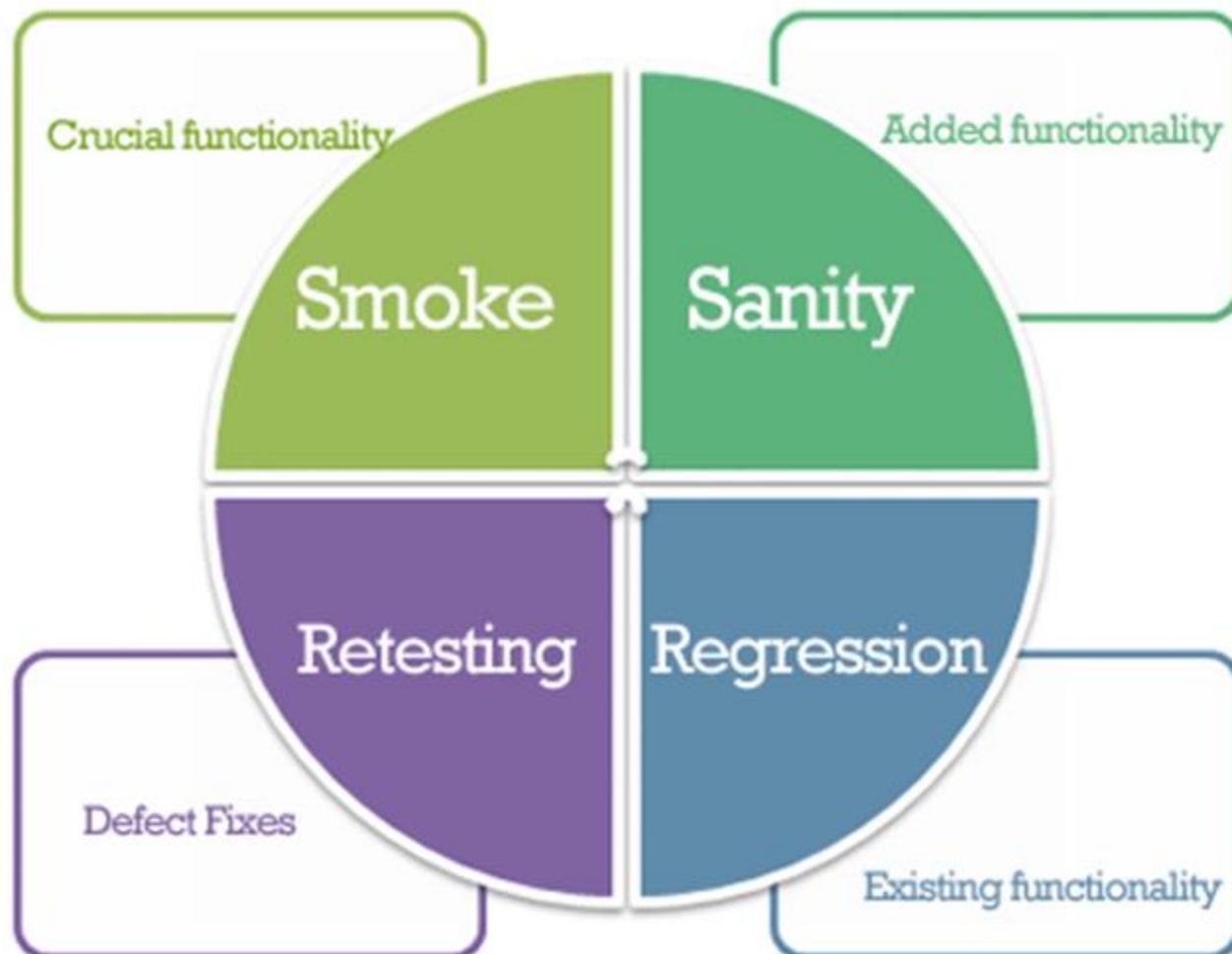
SMOKE TESTING

SANITY TESTING

SCOPE

ACTIVITIES

DELIVERABLES



Regression:
"when you fix one bug, you
introduce several newer bugs."



TEST REPORTING

ISSUE REPORTING – WHERE, HOW ?

TESTING ARTIFACTS / DELIVERABLES

SCOPE

ACTIVITIES

DELIVERABLES

ACTIVITIES

**DEFECT REPORTING
MEASUREMENTS
TEST REPORTS**

**DEMONSTRATE THE ADDED VALUE AND
THE QUALITY WE ARE DELIVERING
TO OUR CUSTOMERS,
BY MEASURING KEY INDICATORS
THAT DRIVE IMPROVEMENT AND INCREASE
CLIENT'S SATISFACTION**

EXAMPLES OF PROJECT'S METRICS

Selected metrics should be **meaningful, useful** at different levels (team, management, client), **easy to collect**.

PRODUCTIVITY METRICS

- Estimation deviation
- Sprint predictability
- Velocity
- Backlog health
- Waste
- Effort / time analysis

QUALITY METRICS

- Defects Detection Rate
- Defects per product
- CNC (% rework)
- Cost of Quality
- Unit Test Coverage
- Code Complexity
- Test automation

PRODUCT METRICS

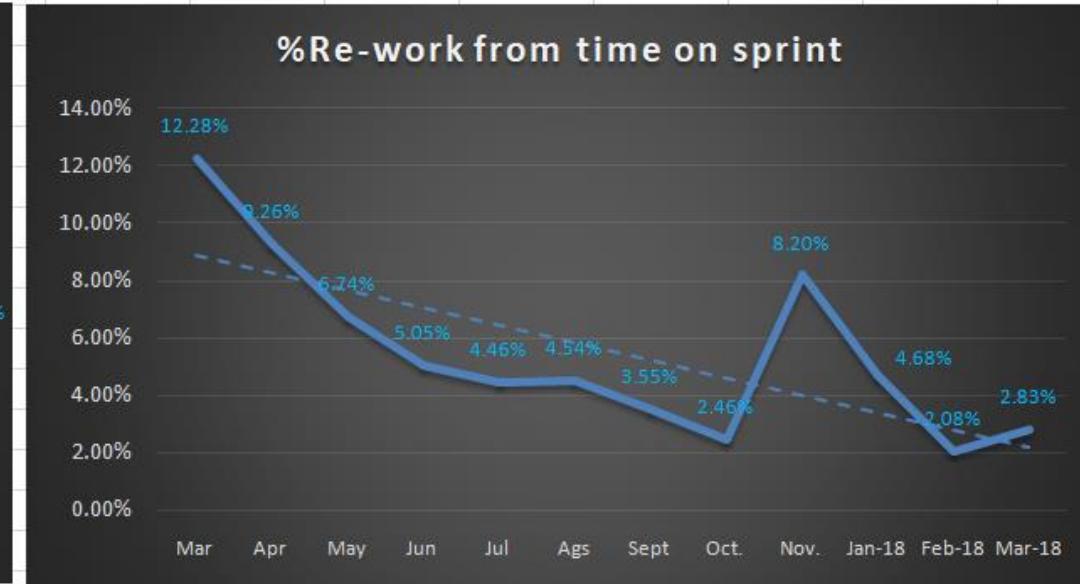
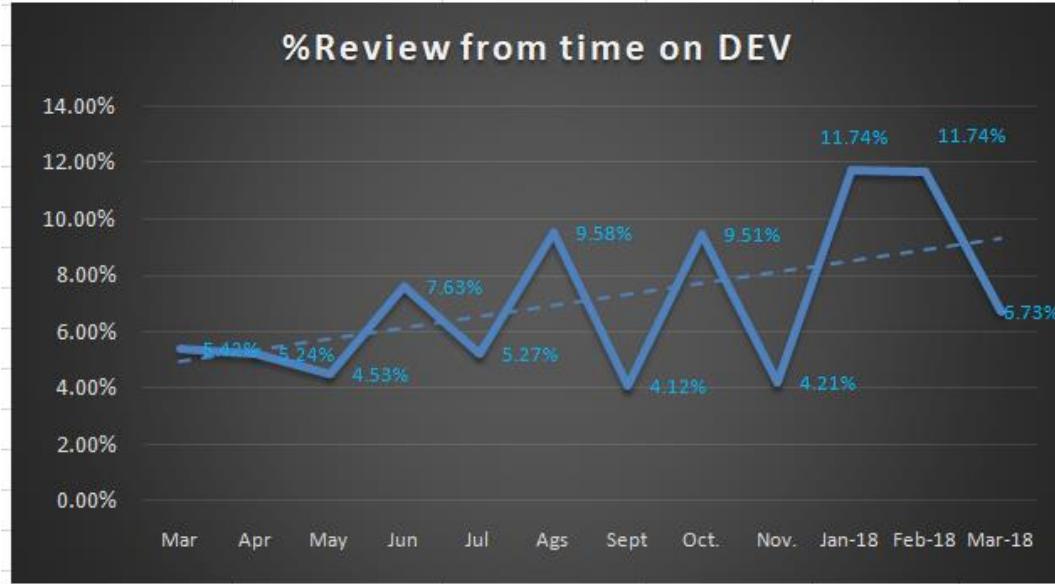
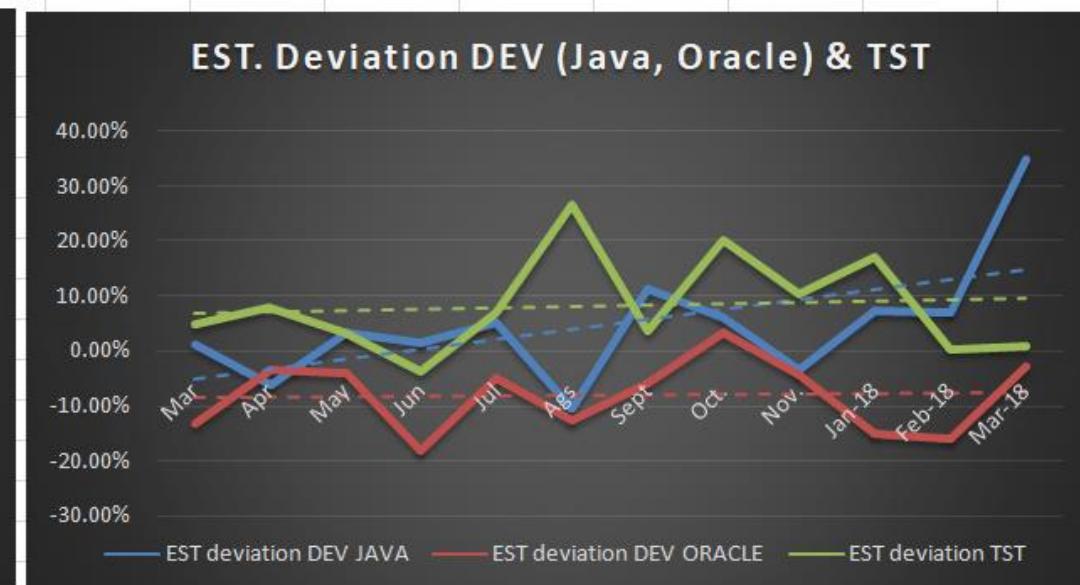
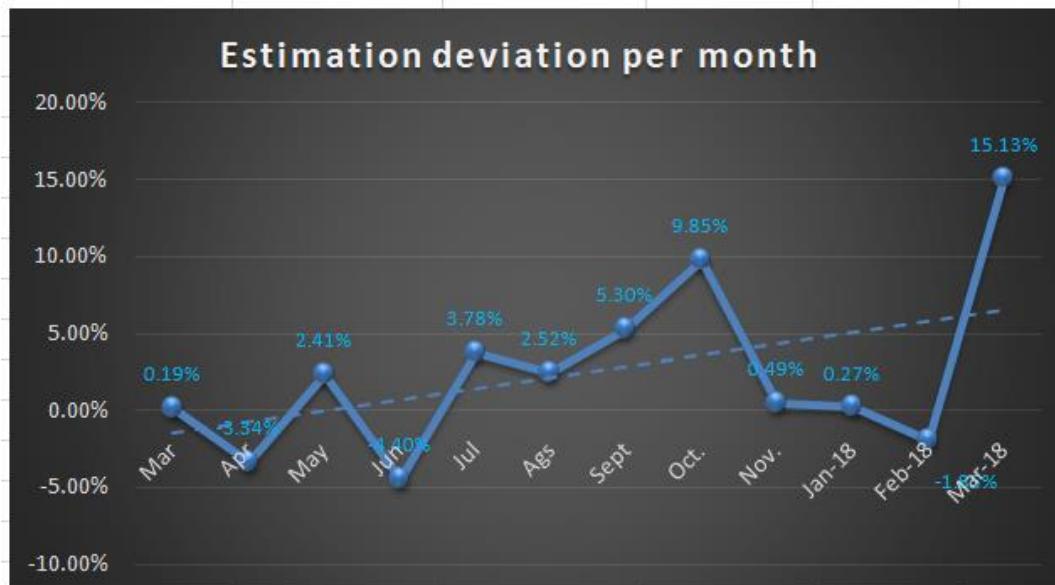
- Security Performance

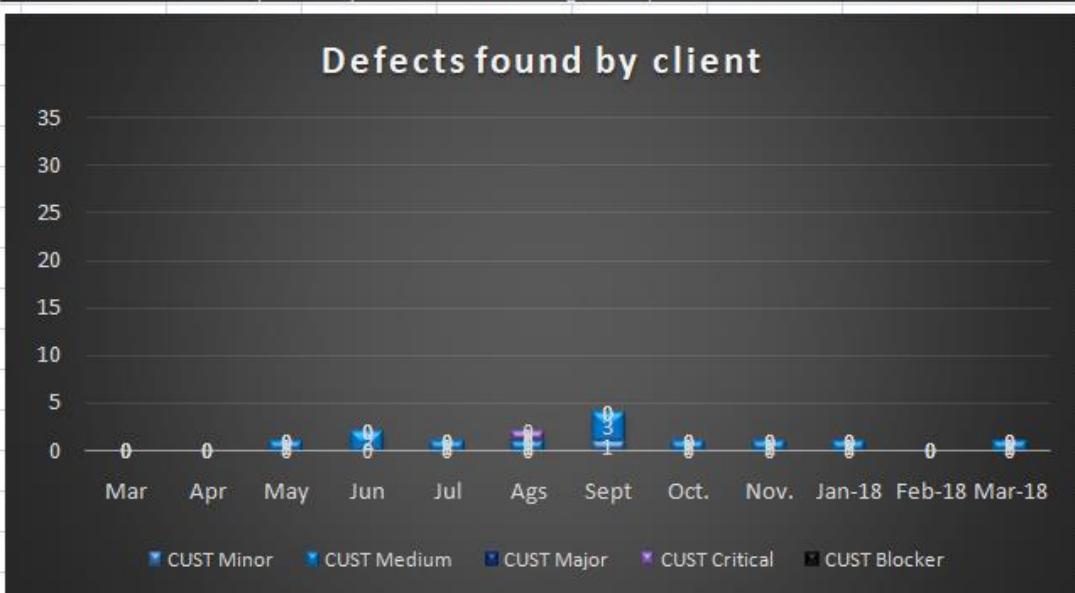
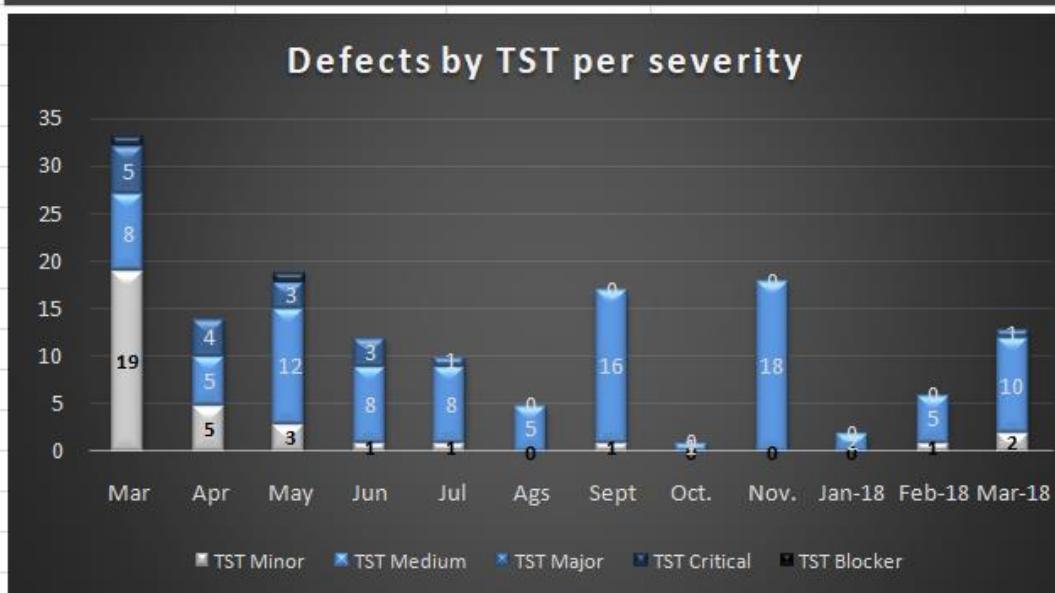
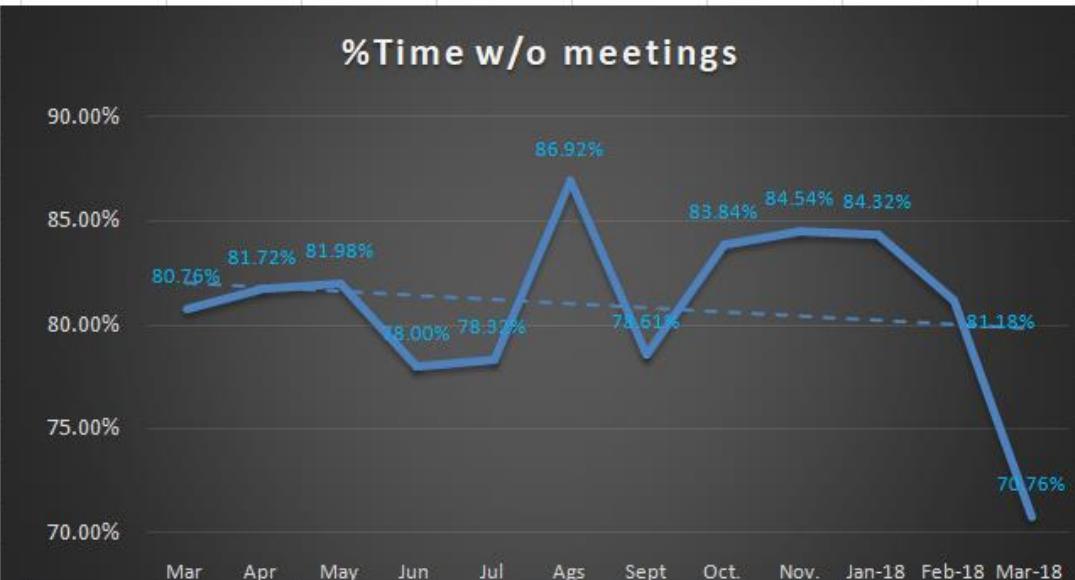
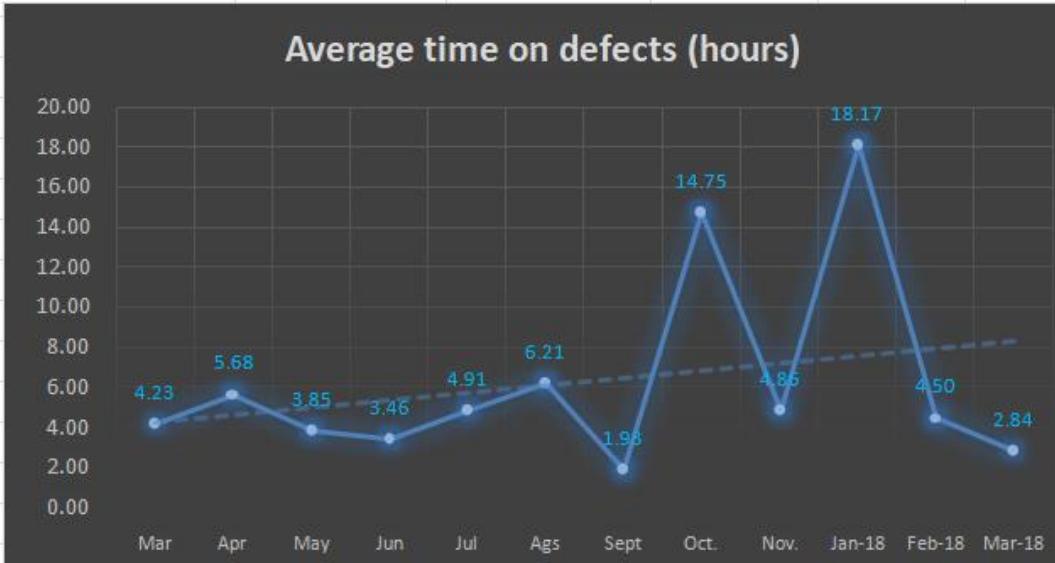
SATISFACTION METRICS

- Team morale
- PO satisfaction
- Client satisfaction

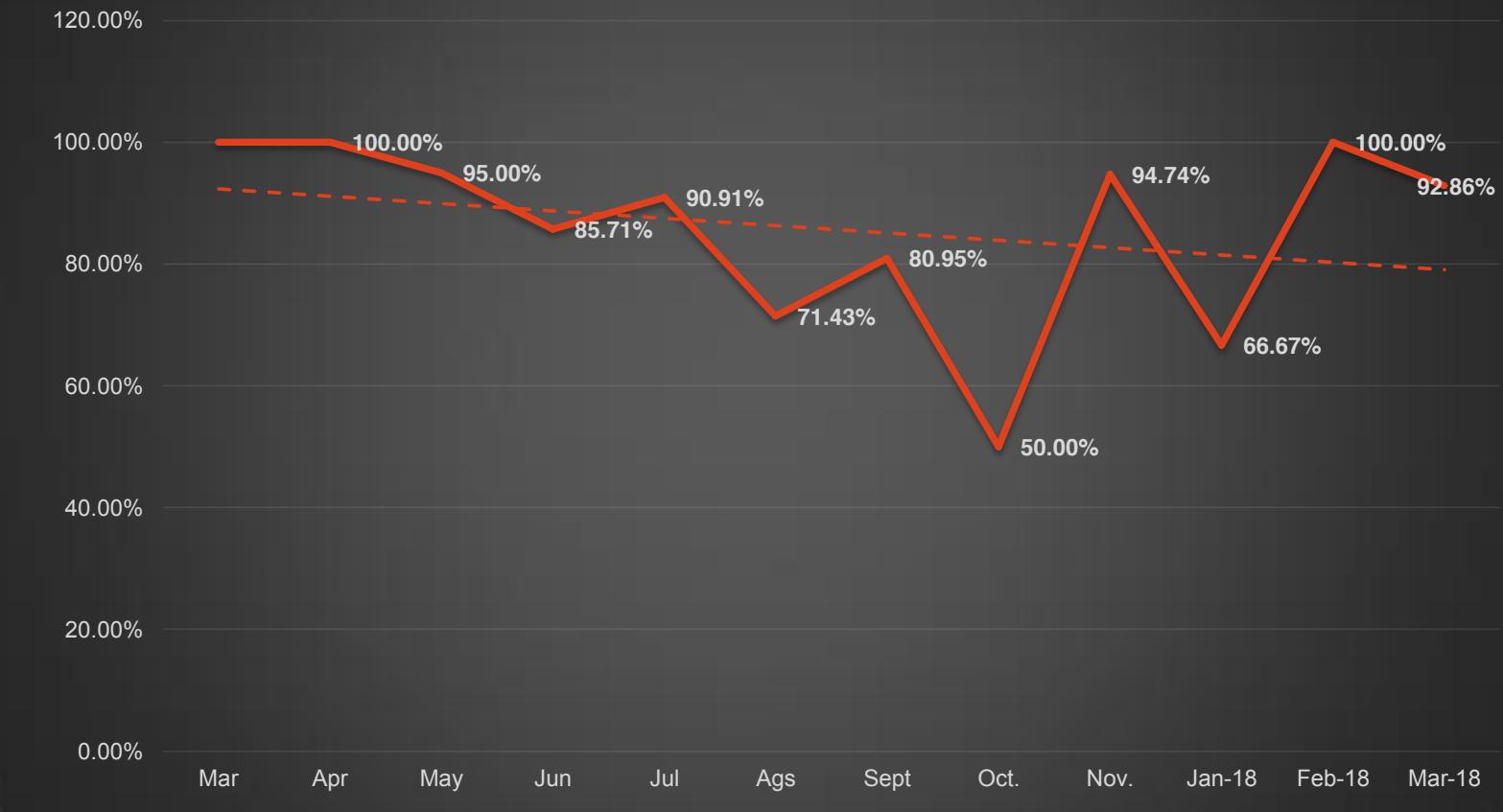
RESPONSIVENESS METRICS

- Cycle time
- Lead time

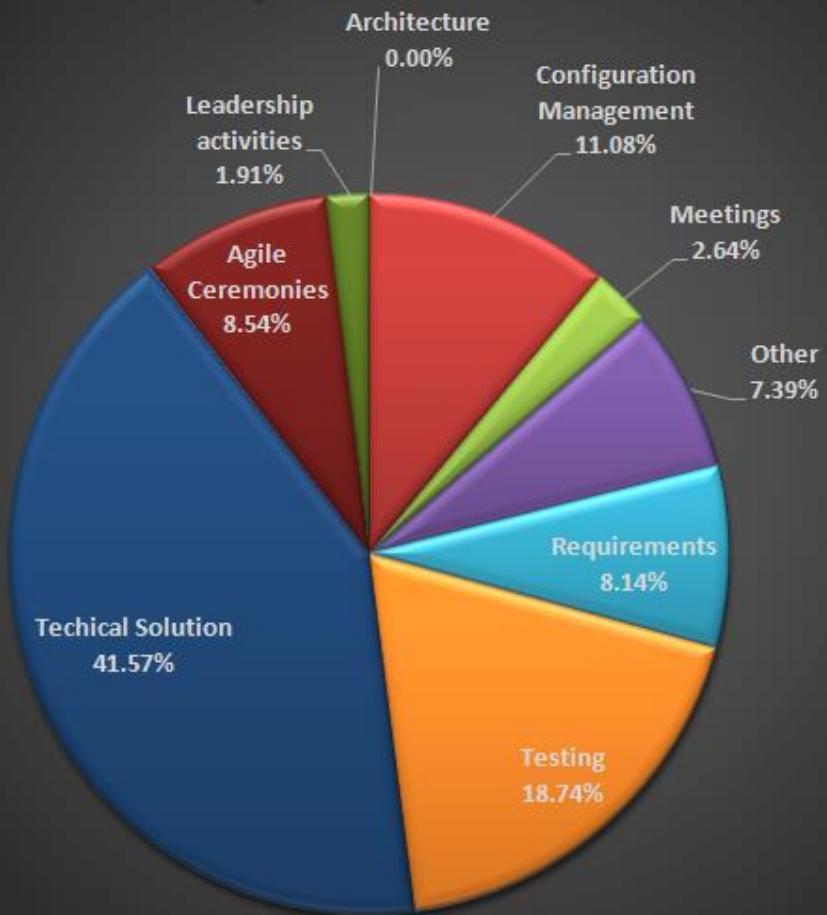




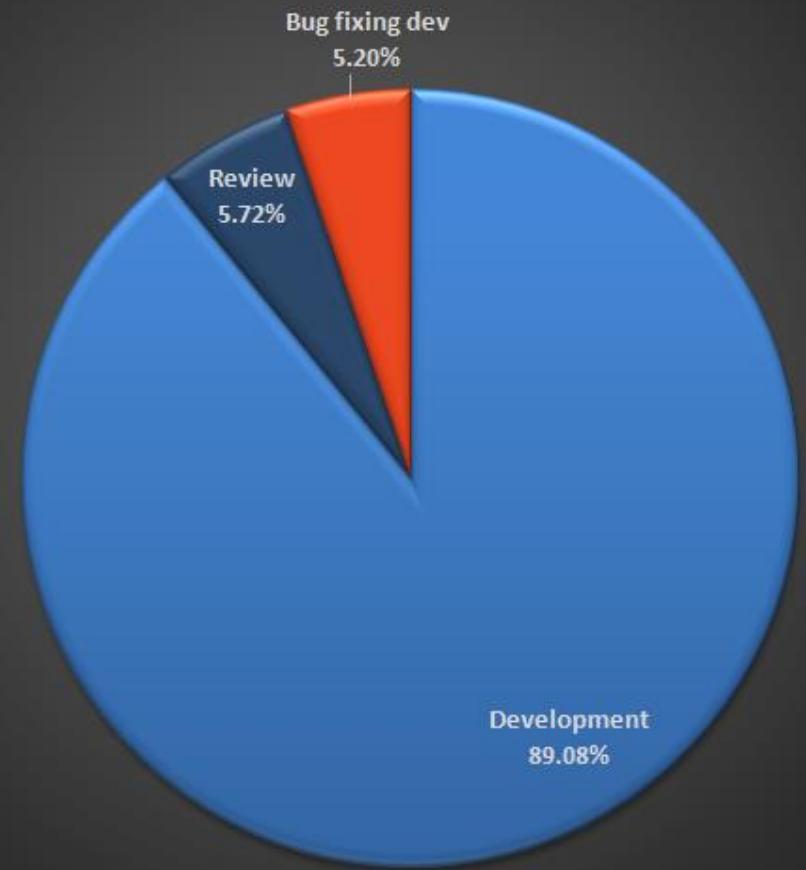
DDR TST vs. CUST



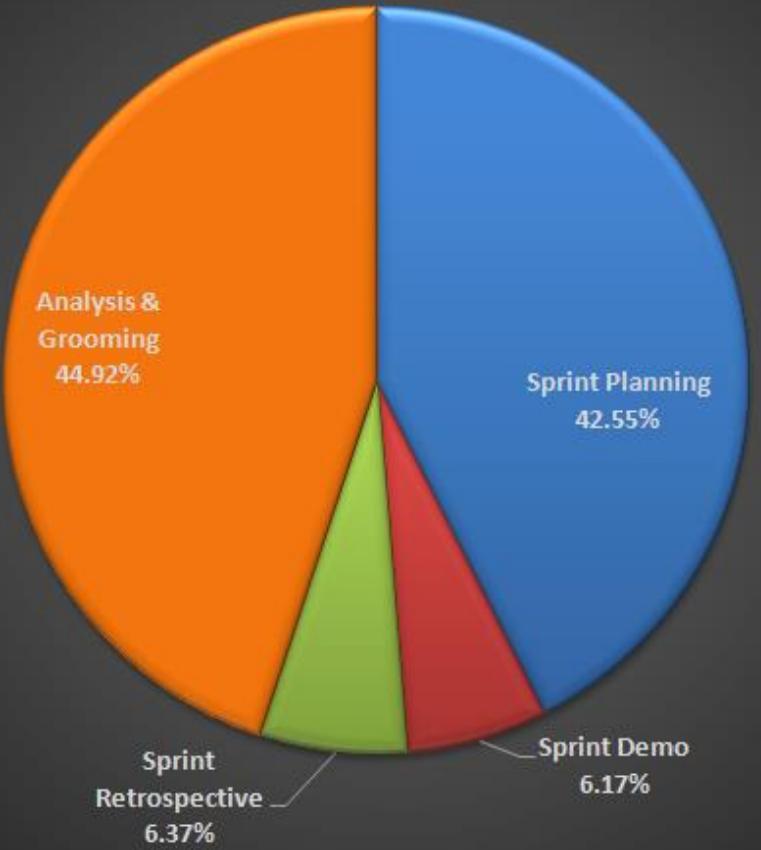
Project hours 2018



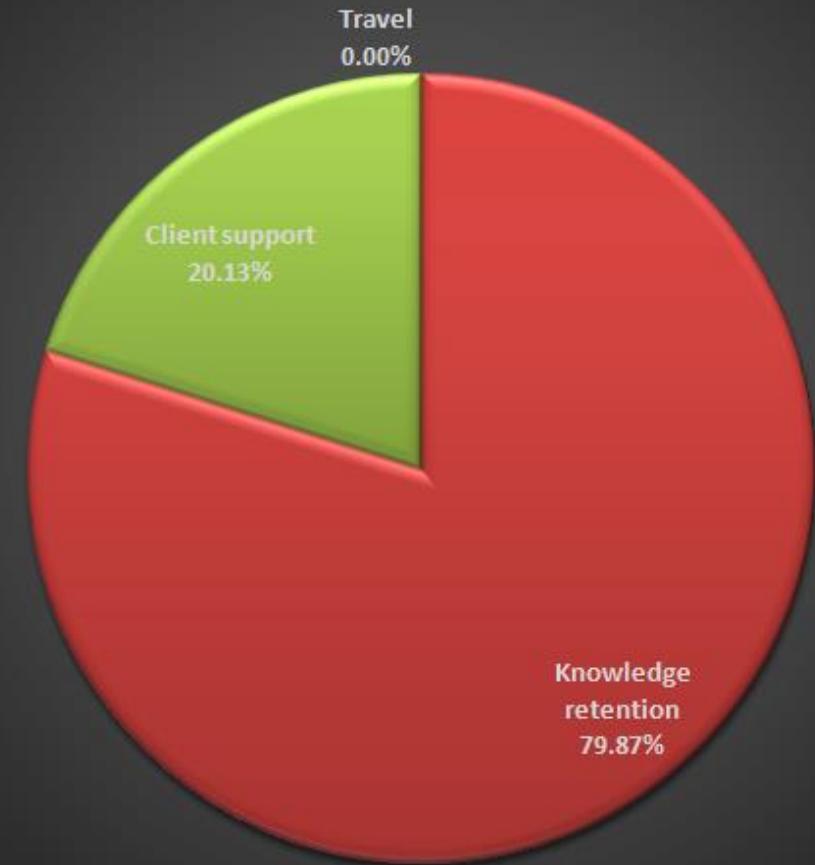
Technical Solution hours



Agile Ceremonies



Other activities



DELIVERABLES

TEST PLANS

TEST CASES / SCENARIOS

TEST SCRIPTS (AUTOMATION)

DEFECTS LOGS

TEST REPORTS

EXIT REPORT

RBTASLA-104

When the base currency is other than US or Euros(example Japan Yen), the values does not change after changing the base currency to Euros

[Edit](#) [Comment](#) [Assign](#) [More](#) [Closed > Open](#)

Details

Type:	<input checked="" type="checkbox"/> Bug	Status:	CLOSED
Priority:	<input checked="" type="checkbox"/> Blocker	Resolution:	Cannot Reproduce
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Source System	NA		
Incident ID:			
Steps to Reproduce:	Find out a client whose base currency is other than USD or Euros. Now change the base currency of this client through MDM to USD and observe the value on the CR UI		
Actual Result:	The value remained same but the text changed for JPY to USD(see the attached screenshot)		
Expected Result:	Would expect the value to change depending on the currency selected		

People

Assignee:	 Christophe de Breuck (Inactive)
	Assign to me
Reporter:	 Poonam Patil (Inactive)
Votes:	 Vote for this issue
Watchers:	 Start watching this issue

Dates

Created:	07-Oct-14Y 12:27 PM
Updated:	23-Feb-15Y 12:30 PM
Resolved:	18-Feb-15Y 3:08 PM

Time Tracking

Estimated:	<div style="width: 0%;"><div style="width: 100%;"></div></div> 0m
Remaining:	<div style="width: 0%;"><div style="width: 100%;"></div></div> 0m
Logged:	<div style="width: 100%;"><div style="width: 100%;"></div></div> 6h 15m
<input checked="" type="checkbox"/> Include sub-tasks	

Development

[Create branch](#)

Agile

Completed Sprint: **Sprint 10** ended 02-Mar-2015
[View on Board](#)

HipChat discussions

 Drop files to attach, or [browse](#).



2 SUBJECT OF THIS TEST REPORT

1. Test Scope

- The present test report concerns Client Reporting release 5.7.0, delivered by Endava on 02.03.2017. The release's main scope is to deliver the changes regarding Tridion migration phase two, the email templates and the Configuration page in MDM. As some impact was on all applications, they will all be delivered: MDM, HPO CR, HPO widget, Loans ACBS widget, OG, EOY OG, OAG and mail jobs.
- Development and testing for this release was done with widgets and HPO CR installed with the theme version BlueLight theme 3.76.0.

2. Test Strategy

- The following tables include the user stories delivered, as well as the test approach and conditions (scenarios) covered within those stories.
- All testing was done manually except where otherwise specified.
- The main focus was on testing the integration of the new ContentManagement service with all the applications, fail and error handling were also in scope.
- Informations about how to run the jobs can be found in the documentation from the References table, section #2.

3 PROGRESS AND QUALITY REPORT

This part serves as the quality report of the released packages. Also, it is used for alignment on the areas covered by our testing team and the testing team.

3.1 PROGRESS OF THE TEST EXECUTION

HPO Client Reporting			
User Story/Bug	Test approach/conditions	Browser	Defects & comments
3244 MAINTAIN AND USE EMAIL SUBJECT AND CONTENT IN CLIENT REPORTING INSTEAD OF TRIDION	Tested the email sent from HPO CR: Disagree mail, approval mail sent at Reopen and approval mail sent at Close and Save new. Regression done on the Disclaimers	FF 51.0.1 Chrome	RABOCR-3959 RABOCR-3966 RABOCR-3969
3244	sendConfirmationMetadata service there was impact also on this service for the Confirmations. The email sent was the main focus of testing	NA	-

```

1 Feature: HPO Details - Check the upcoming maturities tables from MyOverview page
2
3 As an authorized Client Reporting client user, I want to be able to select a certain client I am authorized for and:
4   - see the when client's open transactions will mature
5   - check the upcoming maturities per product group
6   - check the upcoming settlements of the open trades, based on the period chosen (TBD)
7
8
9 @DONE
0 @HighPriority
1 Scenario Outline: The one where I have some upcoming ID transactions.
2 Then I should see the upcoming events for <product type>
3 |Client Id      |Product Type    |Expiration date      |Transaction Summary    |Rate    |
4 |<client id>  |<product type> |<expiration date>  |<transaction summary> |<rate> |
5
6 Examples:
7 |client id     |product type      |expiration date      |transaction summary      |rate    |
8 |300203933    |Cross Currency Swap |14-jun-2018          |EUR 124.500.000,00      |35,00%  |
9 |300203933    |Forward Rate Agreement |23-jun-2018          |EUR 124.500.000,00      |35,00%  |
0
1
2
3 @DONE
4 @HighPriority
5 Scenario Outline: The one where I have some upcoming FX transactions.
6 Then I should see the upcoming events for <product type>
7 |Client Id      |Product Type    |Expiration date      |Transaction Summary    |Rate    |
8 |<client id>  |<product type> |<expiration date>  |<transaction summary> |<rate> |
9
0 Examples:
1 |client id     |product type      |expiration date      |transaction summary      |rate    |
2 |300203933    |FX Option        |29-okt-2017          |U koopt Put USD 10,55 / Call BRL 500,00 | -30,500000 |
3 |300203933    |FX Option        |29-okt-2017          |U koopt Call BRL 500,00 / Put USD 20,77 | -30,500000 |
4
5
6
7 @DONE
8 @HighPriority
9 Scenario Outline: The one where I have some upcoming Com transactions.
0 Then I should see the upcoming events for <product type>
1 |Client Id      |Product Type    |Expiration date      |Transaction Summary    |Rate    |
2 |<client id>  |<product type> |<expiration date>  |<transaction summary> |<rate> |
3
4 Examples:
5 |client id     |product type      |expiration date      |transaction summary      |rate    |
6 |300203933    |Commodity Option |26-okt-2017          |USD 125,45               |          |

```

ASK QUESTIONS



KAHOOT.IT

.THX

EMAIL US AT:

ROXANA.SOPORAN@ENDAVA.COM

Software Systems Verification and Validation

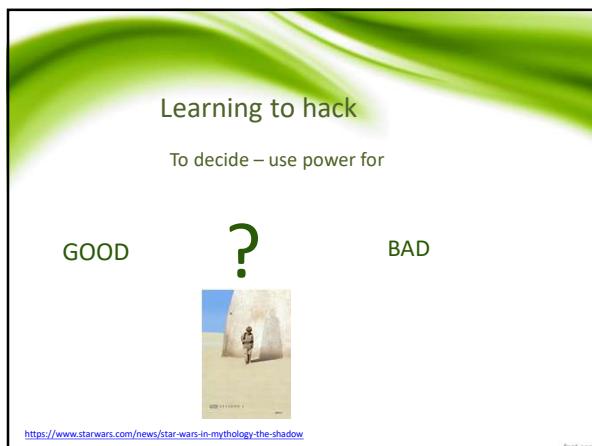


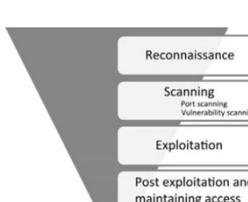
Outline

- Security testing
 - Definition
 - Terms
- Security testing types
- Security testing - More information
- Penetration testing
 - Definition
 - Demo
- Security testing
 - Video presentations by students

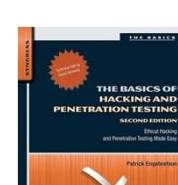
Security testing

- **SECURITY TESTING** is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.
<http://softwaretestingfundamentals.com/security-testing/>
- Security terms (<https://www.qualitestgroup.com/white-papers/what-is-security-testing/>)
 - **Asset** – Anything that has value to an organization, subject to many kinds of threats. [ISO/IEC 13335-1:2004]
 - **Threat** – A potential cause of an unwanted incident, which may result in harm to a system or organization. [ISO/IEC 27001:2005]
 - **Vulnerability** – Defined as a weakness of an asset or group of assets that can be exploited by one or more threats. [After ISO/IEC 27001:2005]. Vulnerabilities can be found in software, information systems, network protocols and devices, etc. If vulnerability is not managed, it will allow a threat to materialize. Examples of vulnerability include unpatched software, weak passwords, lack of access control, no firewall installed, etc.
 - **Risk** – The potential that a given threat will exploit vulnerabilities to cause loss or damage to an asset or group of information assets and thereby cause harm to the organization. It is measured in terms of a combination of the probability of an event and the severity of its consequences.
 - **Information security**– the preservation of confidentiality, integrity and availability of information; in addition, other properties, such as authenticity, accountability, non-repudiation, and reliability can also be involved. [ISO27002:2005] Industrial espionage is unauthorized collection of confidential, classified or proprietary documents.





- A penetration test - attempts to exploit the vulnerabilities to determine whether unauthorized access or other malicious activity is possible.
 - <https://www.veracode.com/security/penetration-testing>



The Basics of Hacking and Penetration Testing, Second Edition, Effect Hacking and Penetration Testing White Hat, Patrick Brumley

Penetration testing

Demo

Security testing

Next Lecture

- Agile testing

Questions

- Thank You For Your Attention!

Software Systems Verification and Validation

Assoc. prof. dr. Andreea Vescan Lecture 12: Structure of
the final exam + Agile Testing

Babeş-Bolyai University
Cluj-Napoca
2018-2019



Outline

- Surprises!
- Structure of the final exam
- Agile Testing
- Surprises!

Surprises!

- Experiment : Mindmaping

How to Mind Map®



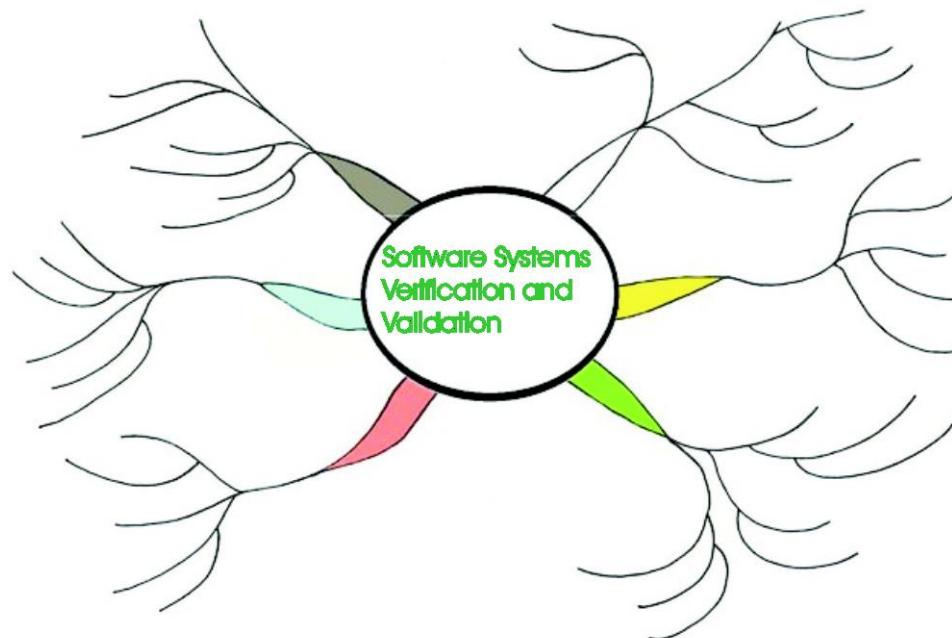
Quick Start Guide

7 Steps to Better Thinking

1. Set your purpose/goal.
2. Start in the CENTRE of blank paper turned sideways.
3. Quickly sketch an IMAGE of your focus in the centre.
4. Use at least 3 COLOURS, for emphasis, structure, texture, creativity.
5. Draw curved lines, radiating from centre (thick to thin) CONNECTING main branches to central image & at each level.
6. Use 1 key word or image per line for more power and flexibility in thinking.
7. Use images throughout as a picture paints a 1,000 words.

Surprises!

- Experiment: Mindmaping
“Software System Verification and Validation” in the next 5 minutes!



© Paul Foreman <http://www.mindmapinspiration.com>

Earn: 25 XP in Lecture 12
(if the second mindmap is created)

Outline

- Surprises!
- Structure of the final exam
- Agile Testing
- Surprises!

Examination dates

31.05.2018 – groups 932, 935, 936

7.06.2018 - groups 931, 933, 934

Please come to the scheduled examination date with your own group! You will not be evaluated otherwise!

Grading

Grading (Tentative) – will be modified – Lecture 01

- $F = 20\% L + 20\% S + 10\% Q + 50\% E$
 - L=lab; S=Seminar; Q=Quizzes; E=Written;
 - Bonus points! See the homepage of the course!
- Conditions to participate at the final exam
 - There is no restriction regarding the participation at the written examination regarding obtained marks at L, S, Q.
 - **Laboratory activity: 3 out of 6 laboratories must be delivered.**
 - **Attendance lab (5 out of 6) -90%**
 - **Attendance sem (4 out of 6) – 75%**
 - Motivations
 - <http://www.cs.ubbcluj.ro/hotarare-privind-motivarea-absentelor-studentilor-nivel-licenta/>
 - Students will present the documents to motivate absences from the seminar/laboratory within one week from the date of absence.
- L/S/Q work may not be redone in the retake session.
- Students from Previous Years to 2018-2019 - All the above rules apply to students from previous years.
- Conditions to pass/complete the SSVV discipline:
 - $F \geq 5$ final grade.

Seminar

- Attendance: 4 out of 6 required
 - 20% of the final grade
- **Required readings before seminars!**
- Seminar structure
 - **Assignment 1** - 10-minutes – discussion on a given topic (the teacher is an observer!)
 - **Assignment 2** - 60-80 minutes – assignments on a given topic
 - **Assignment 3** – 10 minutes – quiz about required reading and seminar discussions.

Laboratory

- Attendance: 5 out of 6 required
- 20% of the final grade
- Lab structure
 - **First hour of each lab** - current lab discussion, problem assignment and **in-class** problem solving.
 - **The second our of each lab** - delivery of previous lab (exception first laboratory).
- Work in teams – exactly 2 member per team.
- Lab grading for each laboratory (except the last one)
 - **In class** assignments! (3 points)
 - **Take-home** (Homework) assignments! (7 points)
- No more than two lab problems will be delivered in one lab meeting. An extra lab problem is delivered **only if time allowed**.
- Delay in lab submissions –2p from that lab grade.
- Maximum 2 weeks delay in submission of the homework assignment.
- **Remark:**
 - At the end of the semester all laboratory homework assignments must be saved on CD and given to the teacher.
 - Each time you deliver a laboratory - the Deliverables of the in-class and take-home assignments must be uploaded in canvas.

Grading Gamifying Education

	Heroic Quests (quizzes)	Side Quests (Lab projects)	Social Quests (Seminars +Video Presentation)	Epic Quests (Final exam)	XP intervals	Grade
Normal session	300 XP	600XP Each Lab 100 XP (in-class 25XP+ take-home 75XP)	600 XP Each Sem 100XP (in-class 25XP + Quiz 75 XP)	Up to 1500 XP	[1400,1500]	5
					[1501,1800]	6
					[1801,2100]	7
					[2101,2400]	8
					[2401,2700]	9
Retake session	0 XP	0 XP	0 XP	Up to 1500XP	Over 2700	10

- **Final exam – you must come (be present) to final exam in order to compute the grade!**
- Bonus points = 300 XP (1p)
 - 200 XP – activities during lectures
 - 100 XP – Lecture 11 and Lecture 12 – Presentations
- Bonus points = 300 XP (1p)
 - Research paper only if you (will) have 300 XP for Labs
 - Topic by teacher + 2 members/team + deliverables
 - Paper submitted to journal for review

Structure of the final exam

Take Home EXAM – 500 XP

- **Solve at home and bring it at exam!**
- Take Home EXAM - for each student – canvas2 under Exam module.
- 75 XP- Mindmap a SSVV notion (each student will receive a notion)
- 25 XP short question (each student will receive a question before exam)
- 150 XP long question (each student will receive a question before exam)
 - The answers to **ShortQuestion** and **LongQuestion** must be **written by hand!**
- 50 XP Study – Assessment Methods at the “SSVV” discipline – survey – online – one day before Final exam
- 200 XP – Study - Impact of Software Testing knowledge on performance in programming
 - 3 requirements to be implemented (no test cases, test cases by students, test cases by teacher provided)
 - Survey – on line

In Class EXAM – 1000 XP

- Structure of exam

Topics:

- 1) Inspection
- 2) Testing (BBT, WBT, Levels of testing)
- 3) Correctness (Floyd/Hoare/Dijkstra, Refinement)
- 4) Symbolic execution
- 5) Model checking

5*150 XP = 750 XP

Topic: IT firm presentations: Altom presentation, Evoxon presentation, Endava presentation

3*50 = 150 XP

Topic: 1 General Questions

Sample type of questions:

- True/False proposition. Justify your answer.
- Fill in the missing information.....
- Multiple answer question.

- Time: 45 minutes for In Class Exam

Outline

- Surprises!
- Structure of the final exam
- Agile Testing
- Surprises!

Agile Testing

- Video presentations
 - Created by a team
 - Each video ~ 5 minutes
 - Audio + written + visual
 - Send by email the link for your video – 1 day before the presentation
- Team 1
 - Ratiu Cosmina
 - Corina Todoran
- Team 2
 - Dragodan Alexandra
 - Fratila Nicolae
 - Nacu Cristian
- Team 3
 - Buhai Alexandru
 - Csoka Ervin
 - Cuibus Ciprian
- Team 4
 - Grigore Dragoş-Alexandru
 - Todoran Ana-Corina
 - Vele Radu-George

SBTM

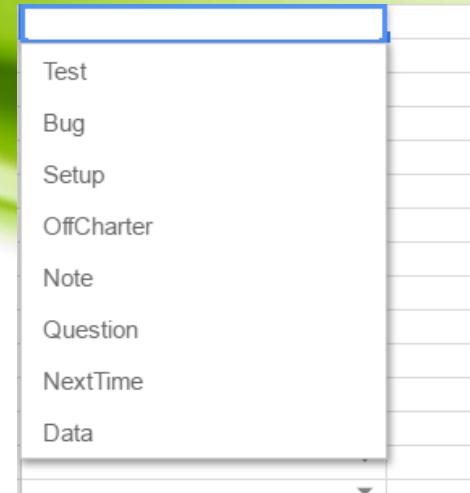
- A method [SBTM] for measuring and managing exploratory testing.
 - An approach to testing with:
 - No test cases
 - No expected results
 - No pass/fails metrics
 - Testing on a project is divided up into focused “sessions” of exploratory testing.
 - A test session
 - A dedicated and uninterrupted period of exploratory testing.
 - Focused on a set objectives defined in a “charter”
 - Generates notes and lightweight reporting on the testing carried out.
 - The testing task is expected to change and evolve.
- Exploratory testing**
- **Is not random**
 - **Is creative and requires intelligent thoughts.**

Session Based Test Management [SBTM]

- **Planning**
 - Charter: a single sentence that sets the scope of the testing
 - What you will test
 - How you plan to test it
 - The information you hope to provide.
 - That scope should be achieved in 90 minutes.
 - Charter are basis of estimation and coverage.
- **Testing**
 - A dedicated and uninterrupted period of testing (no meetings, no email)
 - Use testing experience or instinct; use your domain knowledge
 - Test scenarios could be ok, but not pre-scripted steps
 - Use the software, observe it and be alert of new questions or test ideas
 - Encourage thinking beyond specific features or tasks in isolation.
 - Fully imagine and describe users of the software
- **Note taking/ Reporting**
 - Not just pass/fail count: what you did. Why you did it. What you saw. What questions and ideas you had. Any frustrations/blocks.

SBTM [SBTM]

- Jonathan Bach – course on Session Bases Test Management
- Example:
 - Applied: Sanity and Survey Testing Focus Writer.
 - Software: Focus Writer.
- **Computed metrics**

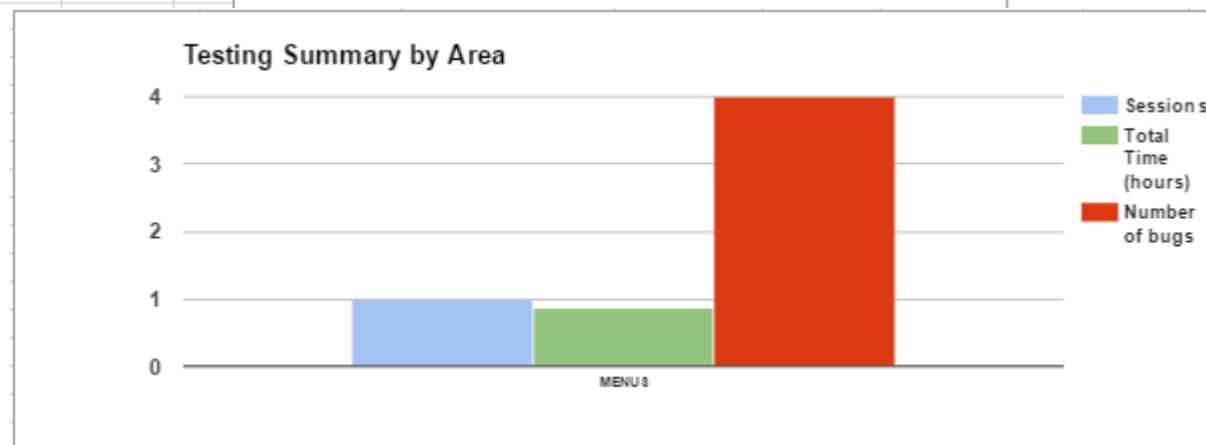
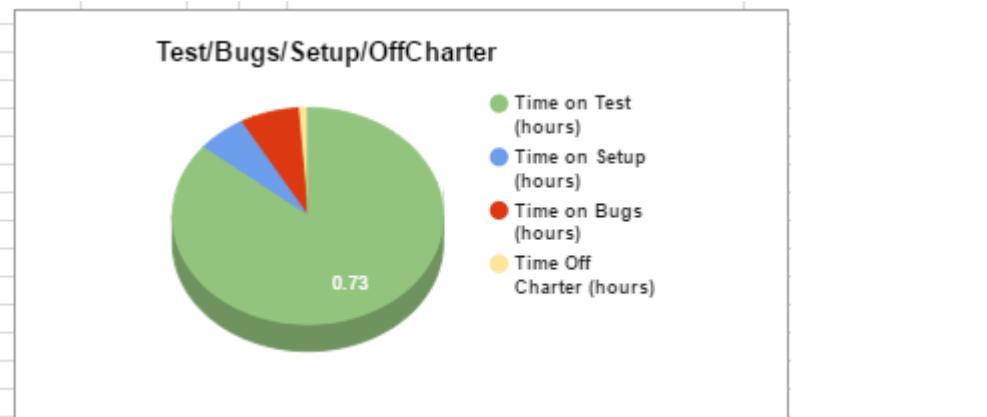


Session end.Duration		52
Number of bugs		4
Time on Test (mins)		44
Time on Setup (mins)		3
Time on Bugs (mins)		3
Time Off Charter (mins)		1
Number of questions		5
Number of ideas for next time		2
Session Tester	alexandra.casapu@altom.ro	
Session Charter	Perform Sanity Testing and Survey Testing for FocusWriter	
Planned session time (mins)		60
Environment Info	Windows, Product Version: FocusWriter 1.6.2, Mac OSX version 1.6.2, on MacOSX Sierra	
Area	Menus	
Current Active Tag	Test	
Time remaining (mins):		9
Session start	SESSION FNDFDI	

SBTM [SBTM]

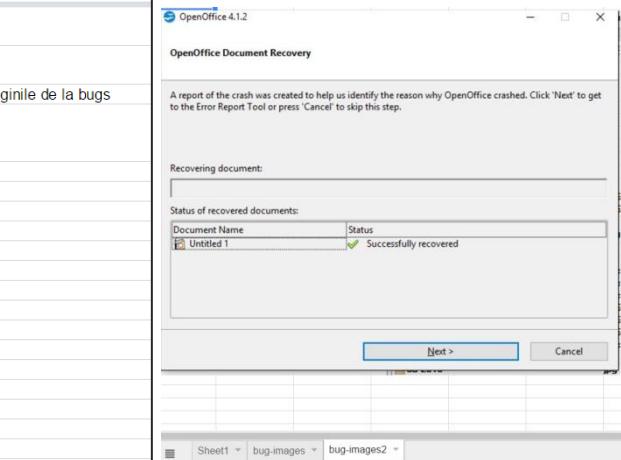
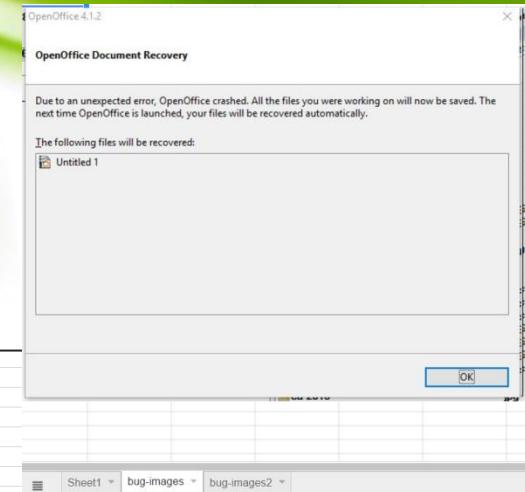
- Jonathan Bach – course on Session Based Test Management [RapidSoftwareTestingCourse_Altom]
- Example:
 - Applied: Sanity and Survey Testing Focus Writer.
 - Software: Focus Writer.

Overall Summary	
Total Sessions	1
Total Time (hours)	0.87
Total Bugs	4
Time on Test (hours)	0.73
Time on Setup (hours)	0.05
Time on Bugs (hours)	0.06
Time Off Charter (hours)	0.01
Total Questions	5
Total Next Time Ideas	2



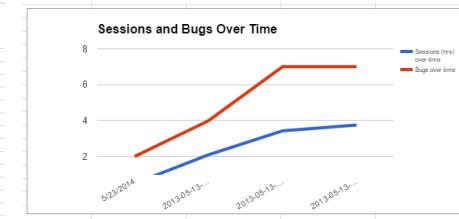
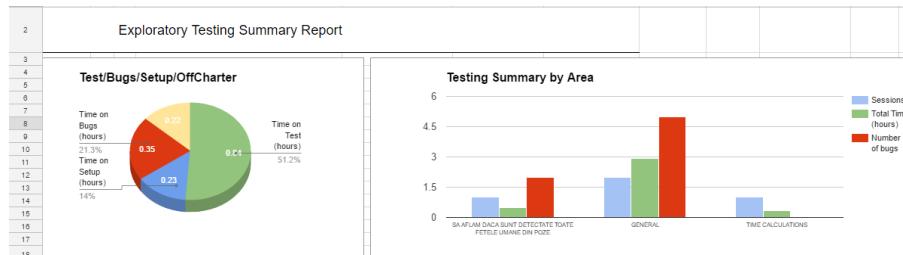
SBTM [SBTM]

- Tabara de testare 2016
 - Example:
 - Software: OpenOfficeImpress.
 - Computed metrics



SBTM

- Lecture 12
- Use SBTM for an open source application
- SBTM template available here:
 - <https://drive.google.com/a/altom.ro/previewtemplate?id=0Aqk-dNwZsfwrdHlfYTRSN3FKQzlVenBYWUhZUk44REE&mode=public&ddrp=1#>
 - Instructions how to use the template
 - <http://altom.ro/blog/version-2-1-of-the-sbtm-session-template-was-released>
 - <http://altom.ro/itester#SBTM>
- <https://docs.google.com/spreadsheets/d/1iyuDZyINzaNOf6fimpVUqDC7RVehrq6-k7u8sMC36ks/edit#gid=0>



Session Based Test Management [SBTM]

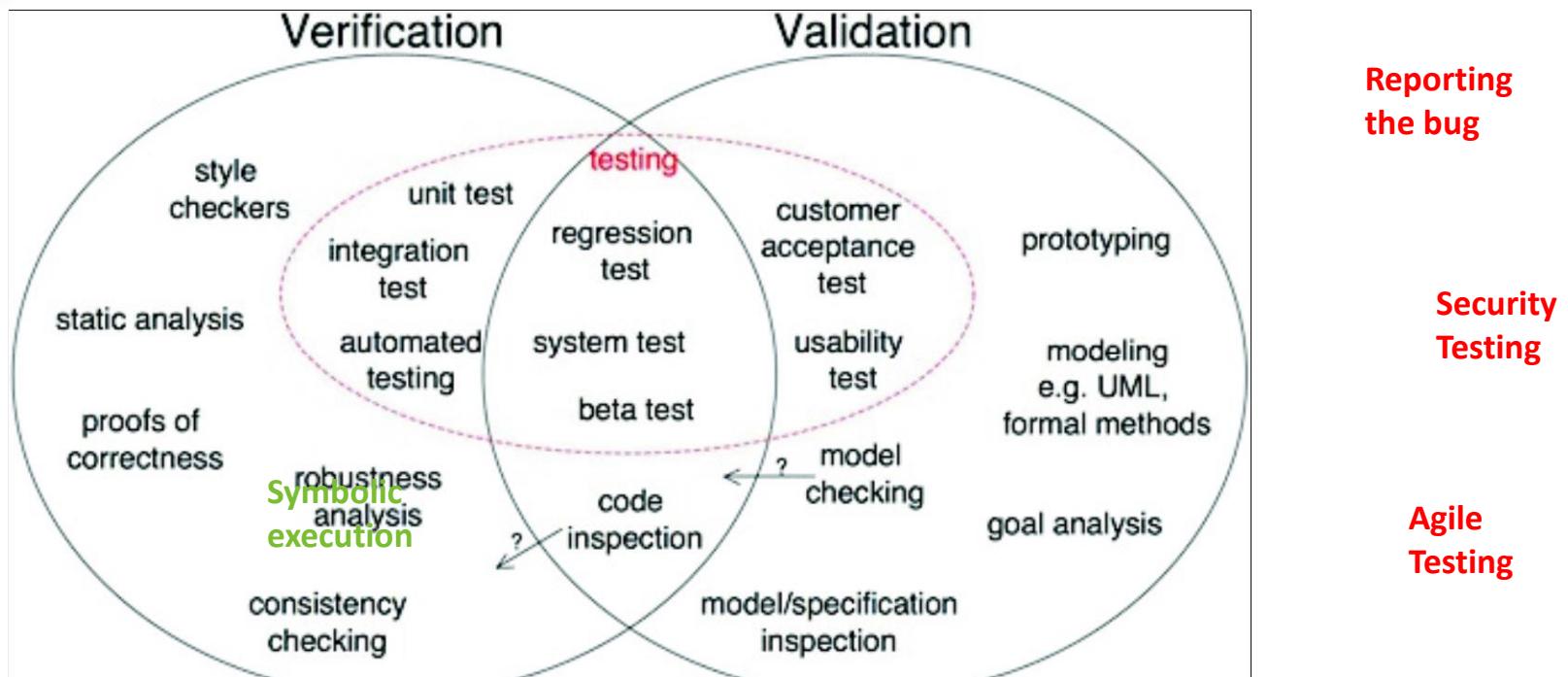
- **Demo: Session Testing**
- **Mission: Secret mission**
- **Time: 20 minute**
- **Debriefing: PROOF**
 - Past + Results + Outlook + Obstacles + Feeling

Outline

- Surprises!
- Structure of the final exam
- Agile Testing
- **Sales paradigm – SSVV**
- Surprises!

Sales paradigm - SSVV

- Motivate the STUDENT - what you will learn!



- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>

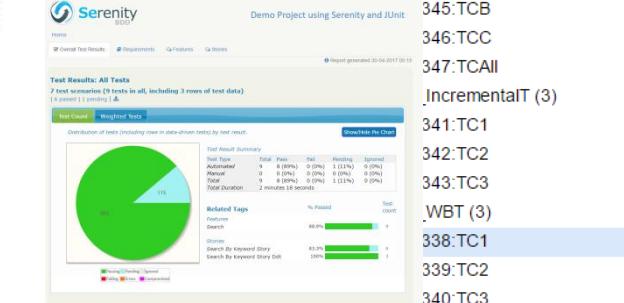
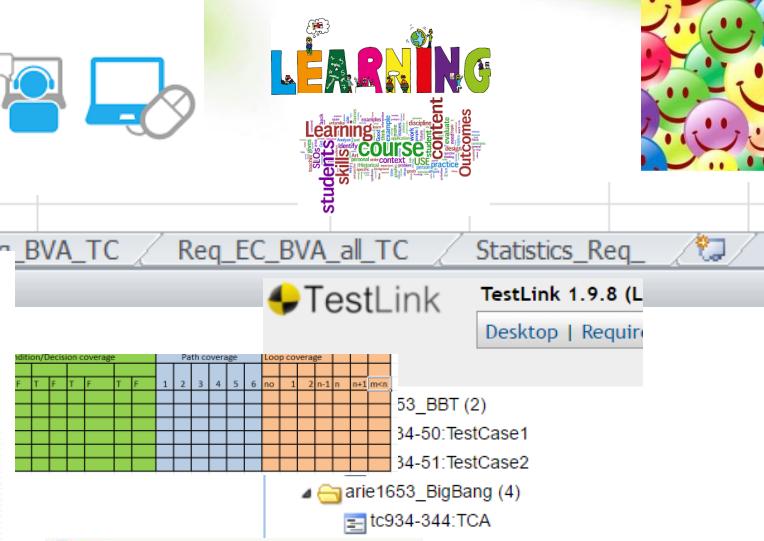
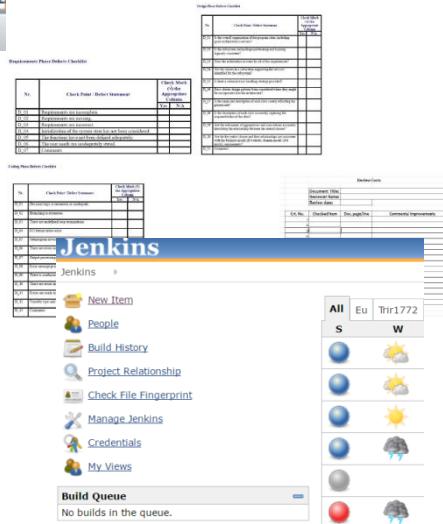
Sales paradigm - SSVV

- Overcome STUDENTS's objections



What testing looks like in real life

Alex & Ru - Altom



VERIFICATION & VALIDATION DURING THE SOFTWARE DEVELOPMENT LIFE CYCLE

PRESENTERS
ROXANA ONA



- Thank You For Your Attention!