Testing Metrics and Tools

SE401: Software Quality Assurance and Testing

Outline

- Testing Metrics
- Testing Tools
- Interview Questions

What is a Metric?

- Metrics can be defined as “STANDARDS OF MEASUREMENT”
- Metric is a unit used for describing or measuring an attribute
- Test metrics are the means by which the software quality can be measured
- Test metrics provides the visibility into the readiness of the product, and gives clear measurement of the quality and completeness of the product

Testing Metrics

- "We cannot improve what we cannot measure" and Test Metrics helps us to do exactly the same.
- Software Testing Metrics are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process.
- The goal of software testing metrics is to improve the efficiency and effectiveness in the software testing process and to help make better decisions for further testing process by providing reliable data about the testing process.

Testing Metrics

- Software testing metrics or software test measurement is the quantitative indication of extent, capacity, dimension, amount or size of some attribute of a process or product.
- Example for software test measurement: Total number of defects

Why we need Metrics ?

- You cannot improve what you cannot measure
- You cannot Control what you cannot measure
  - Without measurement it is impossible to tell whether the process implemented is improving or not
  - Metrics helps in taking decisions for next phase of activities
  - Metrics helps in understanding the type of improvement required and helps in taking decisions on process or technology change

Why Test Metrics are important

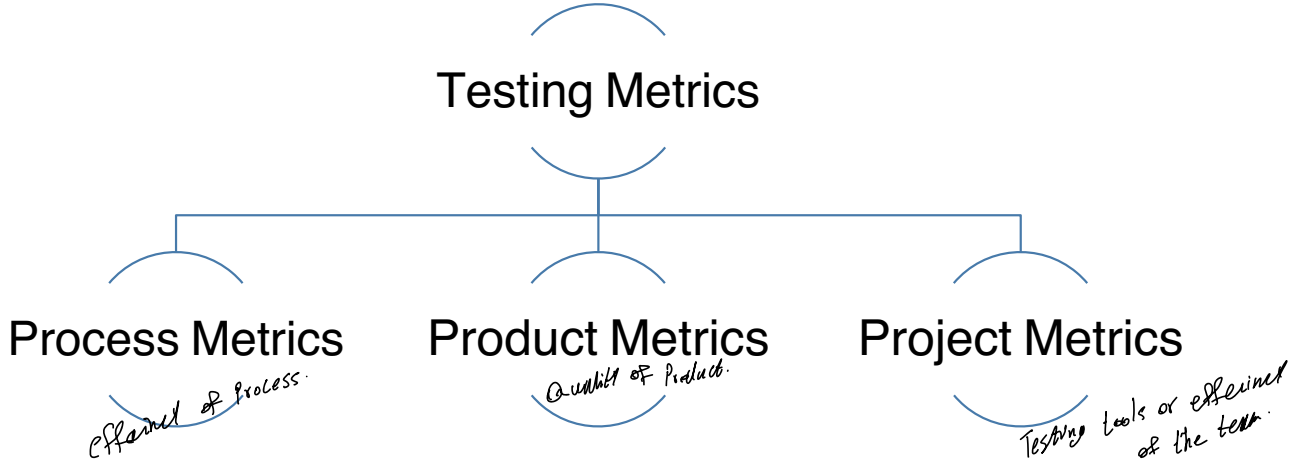
- Take decision for next phase of activities
- Evidence of the claim or prediction
- Understand the type of improvement required
- Take decision or process or technology change

Why Test Metrics

- The aim of collecting **test metrics** is to use the data for **improving the test process**. This includes finding answers to the questions like:

– How long will it take to test? – How much money will it take to test? – How bad are the bugs? – How many bugs found were fixed? reopened? closed? deferred? – How many bugs did the test team did not find? – How much of the software was tested?	– Will testing be done on time? Can the software be shipped on time? – How good were the tests? Are we using low-value test cases? – What is the cost of testing? – Was the test effort adequate? Could we have fit more testing in this release?

Types of Test Metrics



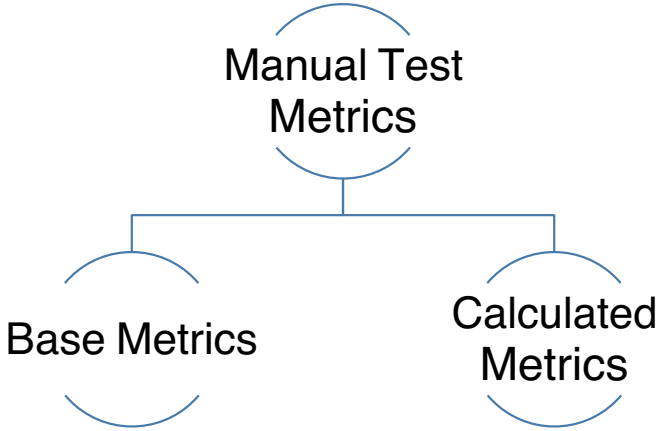
- **Process Metrics**: used to improve the process efficiency of the SDLC (Software Development Life Cycle)
- **Product Metrics**: deals with the quality of the software product
- **Project Metrics**: used to measure the efficiency of a project team or any testing tools being used by the team members

Identifying Metrics

- Identification of correct testing metrics is very important. Few things need to be considered before identifying the test metrics
  - Fix the target audience for the metric preparation
  - Define the goal for metrics
  - Introduce all the relevant metrics based on project needs
  - Analyze the cost benefits aspect of each metrics and the project lifestyle phase in which it results in the maximum output

Manual Test Metrics

- Manual test metrics are classified into two classes
  - Base Metrics
  - Calculated Metrics



Manual Test Metrics

- Base metrics is the raw data collected by Test Analyst during the test case development and execution (# of test cases executed, # of test cases).
- Calculated metrics are derived from the data collected in base metrics. Calculated metrics is usually followed by the test manager for test reporting purpose (% Complete, % Test Coverage).

Testing Metrics

- **Major Base Metrics:**

1. Total number of test cases	7. Number of defects rejected
2. Number of test cases passed	8. Number of defects deferred
3. Number of test cases failed	9. Number of critical defects
4. Number of test cases blocked	10. Number of planned test hours
5. Number of defects found	11. Number of actual test hours
6. Number of defects accepted	12. Number of bugs found after shipping

- Base Metrics are a great starting point, and can then be used to produce calculated metrics

Testing Metrics

- **Major Calculated Metrics:**

Know what you're calculating.

1. **Test Coverage Metrics:**
   - Test coverage metrics help answer, "*How much of the application was tested*?".
   - Two major coverage metrics:
     - Test execution coverage
     - Requirement coverage

Testing Metrics

- **Major Calculated Metrics:**
  1. **Test Coverage Metrics:**

Know what the number you Got represents?

1.1 Executed Tests or Test Execution Coverage Percentage $= \left(\frac{\text{Number of tests run}}{\text{Total number of tests to be run}}\right) \times 100$

- This metric gives us an idea of the total tests executed compared to the total number of tests to be run. It is usually presented as a percentage value

Testing Metrics

- **Major Calculated Metrics:**
  1. Test Coverage Metrics:

1.2 $$\text{Requirements Coverage} = \left(\frac{\textit{Number of requirements covered}}{\textit{Total number of requirements}}\right) \times 100$$

- This metric gives us an idea on the percentage of the requirements that have been covered in our testing compared to the total number of requirements

Testing Metrics

- **Major Calculated Metrics:** ≠ efficiency
  2. **Test Effectiveness Metrics:** Test effectiveness answers, *"How good were the tests?"* or *"Are we running high value test cases?"* It is a measure of the bug-finding ability and quality of a test set. It can be calculated as follow:

High Percentage = High effectiveness

2.1 $$\left(\frac{\text{Bugs found in Test}}{\text{Total bugs found(bugs found in test + bugs found after shipping}}\right) \times 100$$

  - The higher the test effectiveness percentage, the better the test set is and the lesser the test case maintenance effort will be in the long-term.
  - **Example:** If for a release the test effectiveness is 80%, it means that 20% of the defects got away from the test team.

Testing Metrics

- **Major Calculated Metrics:**
  3. **Test Effort Metrics:**
     - Test effort metrics will answer *"how long, how many, and how much"* questions about your **test effort**.
     - These metrics are great to establish **baselines** for future test planning.
     - Major Effort Metrics include:
       - Number of test runs per time, number of defects per hour, number of bugs per test, and average time to test a bug fix.

Testing Metrics

- **Major Calculated Metrics:**
  3. Test Effort Metrics:

3.1 $$\text{Number of tests run per time period} = \frac{\text{Number of tests run}}{\text{Total time}}$$

- This metric gives us an idea on the number of test runs over a certain period of time. (i.e. 30 tests per one day)

Pay attention to the unit Given and convert.

Testing Metrics

- **Major Calculated Metrics:**
  3. Test Effort Metrics:

3.2 $$\text{Bug find rate or Number of defects per test hour} = \frac{\text{Total number of defects}}{\text{Total number of test hours}}$$

- This metric provides information about the rate of finding defects by showing the number of detected defects per test hour.

Testing Metrics

- **Major Calculated Metrics:**
  3. Test Effort Metrics:

3.3 $$\text{Number of bugs per test} = \frac{\text{Total number of defects}}{\text{Total number of tests}}$$

- This metric gives an estimation for the number of defects found in one test. This is calculated by dividing the total number of defects over the total number of conducted tests.

Testing Metrics

- **Major Calculated Metrics:**
  3. Test Effort Metrics:

3.4 $$\text{Average time to test a bug fix} = \frac{\text{Total time between defect fix to retest for all defects}}{\text{Total number of defects}}$$

- This metric presents the average time required to test a bug fix.

- **Example:**

  Average time to test a defect fix= 4/3 = 1.3 defects per day

Defect	Time between fixing the defect and retesting the fix
D1	1 day
D2	1 day
D3	2 days
Total	4

4/3

Testing Metrics

- **Major Calculated Metrics:**
  4. **Test Tracking and Quality Metrics:**

4.1 Passed Test Cases Percentage $= \left(\frac{\textit{Number of Passed Tests}}{\textit{Total number of tests executed}} \right) \times 100$

- This metric gives an indication on the quality of the tested application. It shows the percentage of passed test cases in relation to the total number of executed tests.

Testing Metrics

- **Major Calculated Metrics:**
  4. Test Tracking and Quality Metrics:

4.2 $$\text{Failed Test Cases Percentage} = \left(\frac{\text{Number of Failed Tests}}{\text{Total number of tests executed}}\right) \times 100$$

- This gives an indication on the quality of the tested application. It shows the percentage of failed test cases in relation to the total number of executed tests. It also gives an indication on the effectiveness of the conducted tests.

Testing Metrics

- **Major Calculated Metrics:**
  4. **Test Tracking and Quality Metrics:**

4.3

$$\text{Critical Defects Percentage} = \left(\frac{\textit{Critical Defects}}{\textit{Total defects reported}}\right) \times 100$$

- This metric tracks the percentage of the critical defects in relations to the total number of reported defects.

Testing Metrics

- **Major Calculated Metrics:**
  4. **Test Tracking and Quality Metrics:**

4.4

$$\text{Fixed Defects Percentage} = \left(\frac{\textit{Defects Fixed}}{\textit{Defects Reported}}\right) \times 100$$

- This metric calculates the percentage of the fixed defects in relations to the number of the reported defects. This also gives an indication on the efficiency of testing.

Testing Metrics

- **Major Calculated Metrics:**
  5. Test Efficiency Metrics:

5.1 $$\text{Average time to repair defects} = \left(\frac{\textit{Total time taken for bug fixes}}{\textit{Number of bugs}}\right)$$

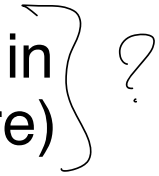
- This metric calculates the average time taken to repair a defect by dividing the total time taken to fix all bugs over the total number of bugs. This metric gives an indication on how efficient repairing defects is.

Testing Metrics

- **Major Calculated Metrics:**
  - More Metrics are used to monitor the testing team productivity, Cost of testing products, etc.
  - You can read more about testing metrics in:
    - https://www.qasymphony.com/blog/64-test-metrics/
    - https://softcrylic.com/blogs/top-25-metrics-measure-continuous-testing-process/
    - https://www.getzephyr.com/resources/whitepapers/qa-metrics-value-testing-metrics-within-software-development

Cumulative Test Time

- The total amount of time spent actually testing the product measured in test hours
- Provides an indication of product quality
- Is used in computing software reliability growth (the improvement in software reliability that results from correcting faults in the software)



Test Coverage Metrics

- Code Coverage (How much of the code is being exercised?)
  - Segment coverage (percentage of segments hit)
    - Every (executable) statement is in some segment
    - A segment corresponds to an edge in a program's directed graph
    - Segment coverage is especially useful during unit and integration testing
    - Segment coverage is cumulative
    - A goal of 85% is a practical coverage value
  - Call-pair coverage (percentage of call pairs hit)
    - An interface whereby one module invokes another
    - A goal of 100% is a practical coverage value
- Requirements coverage (Are all the product's features being tested?)
  - The percentage of requirements covered by at least one test

Quality Metrics

- Defect removal percentage
  - What percentage of known defects is fixed at release?
  - [Number of bugs fixed prior to release/ Number of known bugs prior to release] x 100
- Defects reported in each baseline
  - Can be used to help make decisions regarding process improvements, additional regression testing, and ultimate release of the software
- Defect detection efficiency
  - How well are we performing testing?
  - [Number of unique defects we find / (Number of unique defects we find + Number of unique defects reported by customers)] x 100
  - Can be used to help make decisions regarding release of the final product and the degree to which your testing is similar to actual customer use

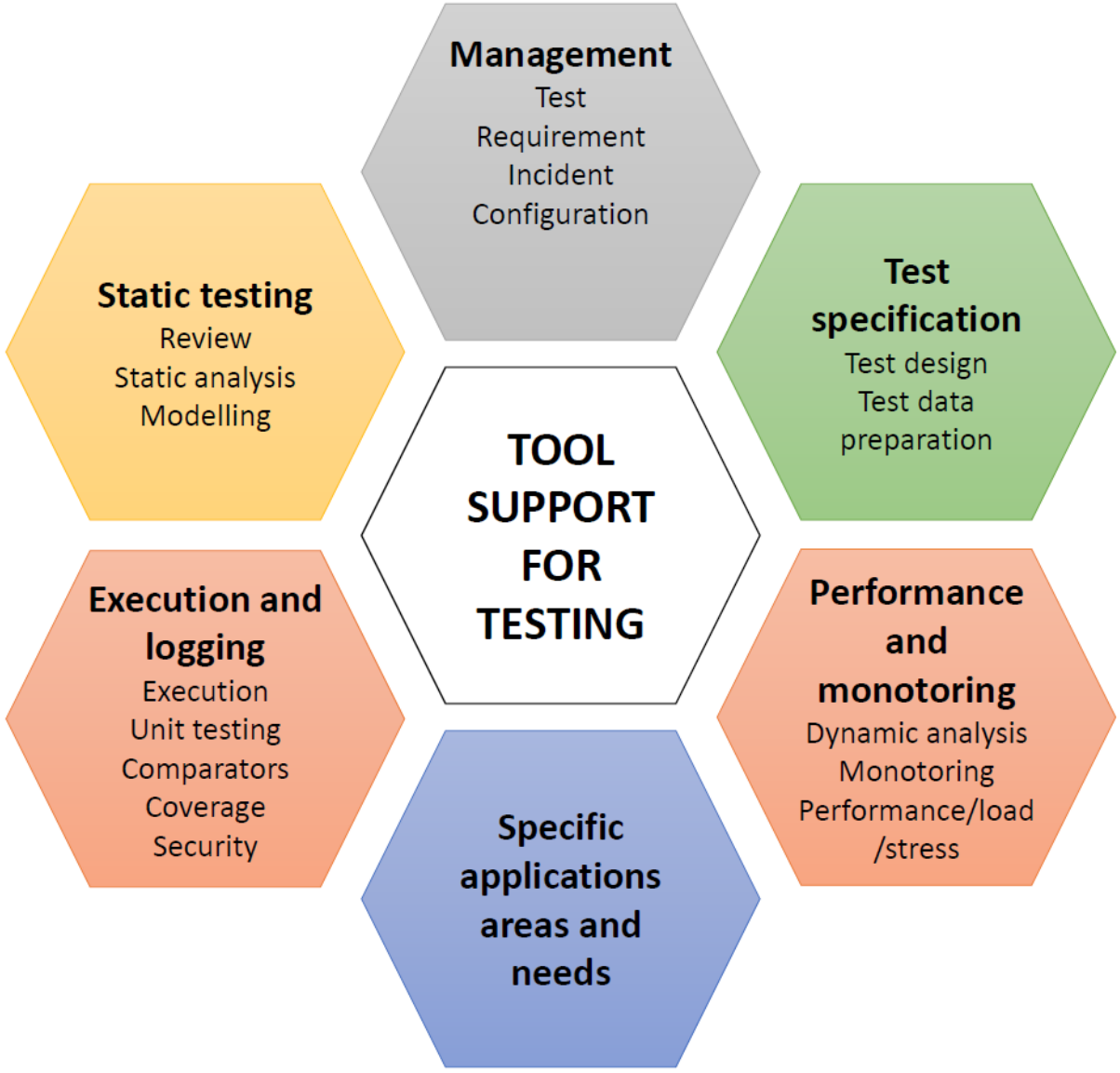
Summary

- Fundamental test metrics are a combination of Base Metrics that can then be used to produce Calculated Metrics.
  - Base Metrics → raw collected data
  - Calculated Metrics → calculated from the base metrics

Summary

- Major Base Software Test Metrics
  - Total number of test cases
  - Number of test cases passed
  - Number of test cases failed
  - Number of test cases blocked
  - Number of defects found
  - Number of defects accepted
  - Number of defects rejected
  - Number of defects deferred
  - Number of critical defects
  - Number of planned test hours
  - Number of actual test hours
  - Number of bugs found after shipping
- Major Calculated Software Test Metrics
  - Coverage
    - Test Execution Coverage
    - Requirement Coverage
  - Effectiveness
  - Effort
    - Number of tests run per time
    - Number of defects per hour
    - Number of defects per test
    - Average time to test a bug fix
  - Quality
    - Passed test cases
    - Failed test cases
    - Critical test cases
    - Fixed Defect percentage
  - Efficiency

Testing Tools


Management
Test
Requirement
Incident
Configuration
Static testing
Review
Static analysis
Modelling
Test specification
Test design
Test data preparation
TOOL SUPPORT FOR TESTING
Execution and logging
Execution
Unit testing
Comparators
Coverage
Security
Performance and monotoring
Dynamic analysis
Monotoring
Performance/load /stress
Specific applications areas and needs

Tool support for testing - types of tools

- **Test tools** can be used for one or more activities that support testing:
  - Used directly in testing (e.g. test execution tools, test data generation tools, result comparison tools)
  - Help in managing the testing process (e.g. test results, requirements, incidents, defects) and for monitoring and reporting the test execution
  - Used in exploration (e.g. tools that monitor the file activity for an application)
  - Any tool that aids in testing

Tool support for testing - purposes

- Tools can have one or more purposes, depending on the context:
  - improve the efficiency of the test activities (e.g. by automating repetitive tasks)
  - automate activities that require significant resources when done manually (e.g. static testing)
  - automate activities that cannot be done manually (e.g. large scale performance testing of client server applications)
  - increase reliability of testing (by automating large data comparisons or simulating complex behavior)

Test tool classification

- **Tools** are classified according to the testing activities that they support.
  - one activity
  - more than one activity, but classification falls under the main activity

- **Notes**
  - Some types of test tool can be intrusive-the tool itself can affect the outcome of the test. (e.g. timing measurements may be different depending on how you measure it with different performance tools).
  - The consequence of intrusive tools is called the probe effect.
  - Some tools offer support more appropriate for developers. Such tools are marked with “(D)”.

Management of testing & tests Tools

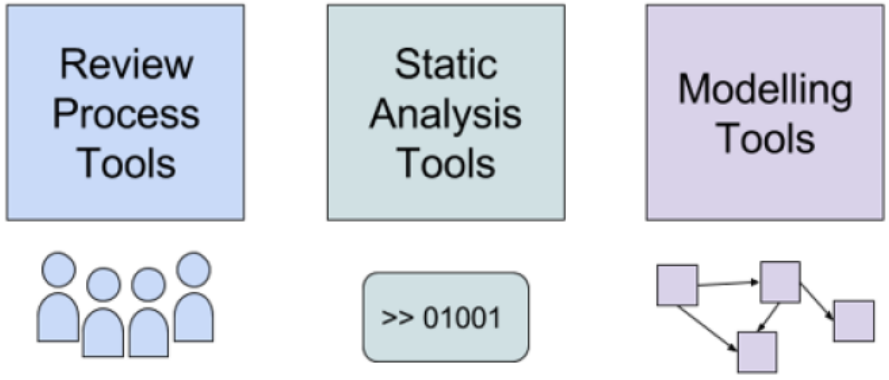
- Characteristics
  - Support for the management of tests and the testing activities
  - Support for traceability of tests , test results and incidents to source documents, such as requirements specifications.
  - Generation of progress reports
  - Logging test results (Monitoring)
  - Offer info on metrics related to the tests.

Management of testing & tests Tools

- Test management tools and application lifecycle management tool (ALM)
- Requirements management tools
  - Store requirements
  - Check for consistency and undefined (missing) requirements
  - Allow prioritization
  - Enable individual tests to be traceable to requirements
- Defect management tools
- Configuration management tools
  - Necessary to keep track of different versions and builds of SW and tests
  - Useful when developing on more than one configuration of the HW/SW environment
- Continuous integrations tool (D)

Static Testing Tools

- Tools for static testing
  - Aid in improving the code/work product, without executing it
- Categories
  - Review tools
    - Supports the review process
  - Static analysis tools
    - Supports code examination
  - Modeling tools
    - Validate models of system/software

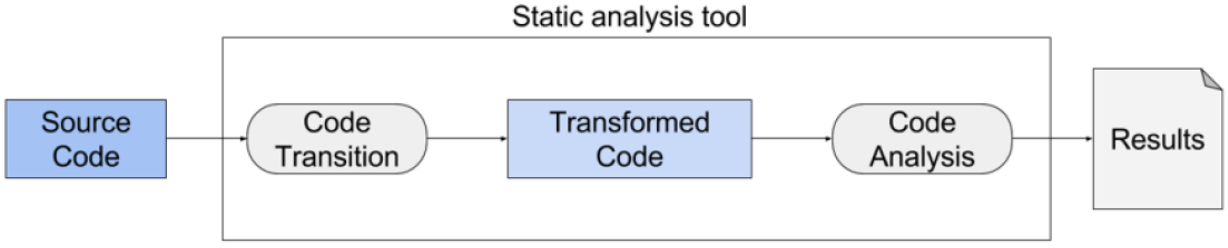


Static Testing Tools

- Review process tools
  - Common reference for the review processes conducted
  - Keep track of all the information from the review process
  - Store and communicate review comments, report on defects and effort
  - Monitoring review status --> Passed, passed with corrections, require re-review
- When to use?
  - Suitable for more formal review processes
  - Geographically dispersed teams

Static Testing Tools

- Static analysis tools (D)
  - Mostly used by developers --> Component (unit) testing
  - Tool is executed --> Code is not
    - The source code serves as input data to the tool
- Extension of compiler technology

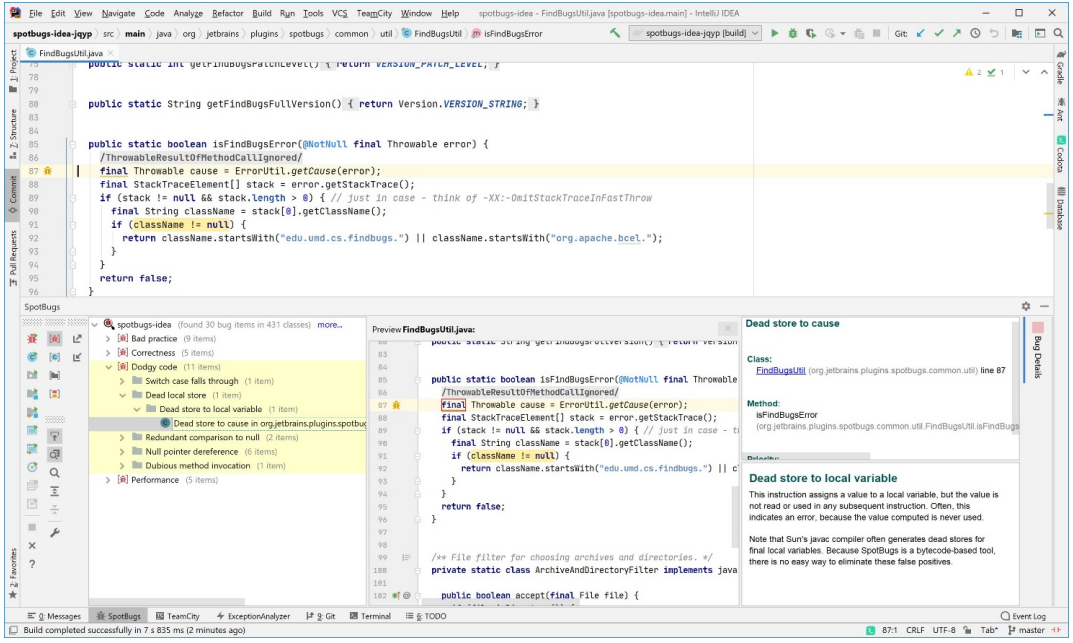


Static Testing Tools

- Static analysis tools (D)
  - Supports developers and testers in finding defects before dynamic testing
- Purpose
  - To better understand the code, and find ways of improving it
- Common features
  - Calculate metrics --> Complexity, nesting levels --> identify areas of risk
  - Enforce coding standards
  - Analyze code structures and dependencies

Static analysis tools (D)

- SpotBugs
  - look for bugs in Java code
  - checks for more than 400 bug patterns



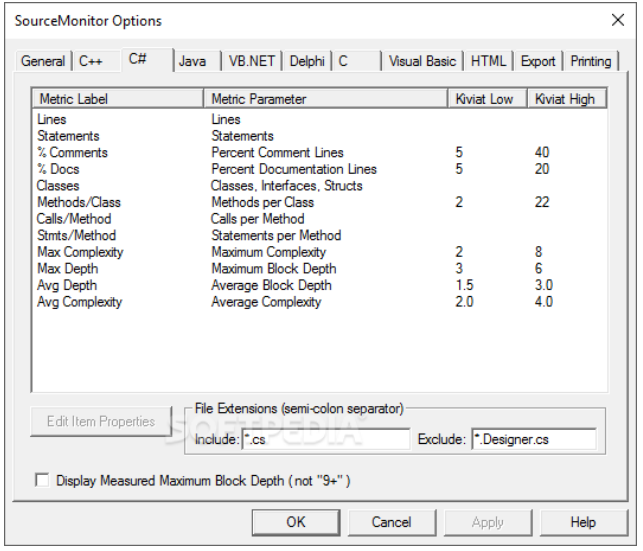
Static analysis tools (D)

- Static analysis tool example: Source Monitor
  - Collects metrics from source code files
  - Display and prints metrics in tables and charts

SourceMonitor -

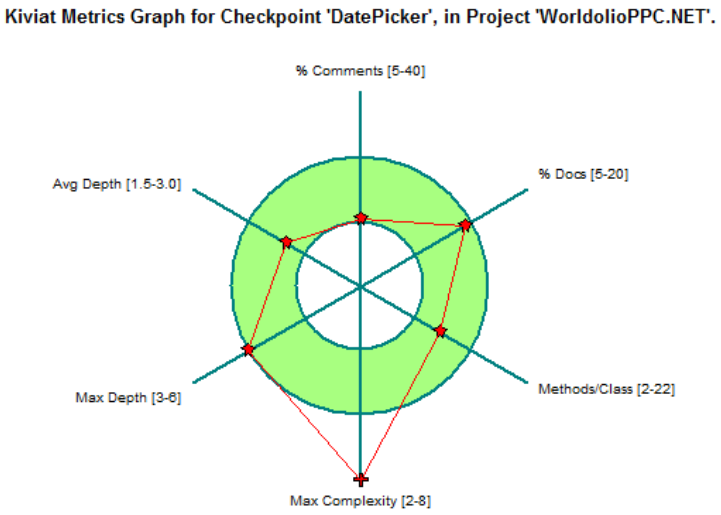
File Edit View Window Help

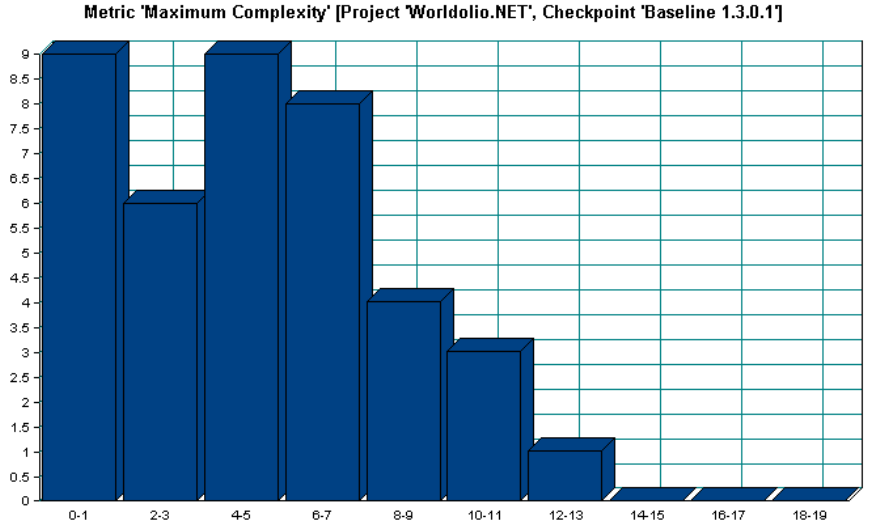
Class	Method Name	Complexity	Statements	Maximum Depth	Calls
monthTool	getMonthNameV1()	14	27	4	0
monthTool	getMonthNameV2()	14	26	2	0
monthTool	getMonthNameV3()	2	4	3	1
monthTool	main()	1	7	2	7
monthTool	monthTool()	1	13	2	13
monthTool	printV1()	1	1	2	2
monthTool	printV2()	4	4	2	3
monthTool	printV3()	1	1	2	2



Static analysis tools (D)

- Static analysis tool example: Source Monitor





Static analysis tools (D)

- JDepend
  - Generates design quality metrics for each Java package.
  - Measure the quality of a design in terms of its extensibility, reusability, and maintainability.

Summary

[summary] [packages] [cycles] [explanations]

Package	TC	CC	AC	Ca	Ce	A	I	D	V
com.caplin.datasrc.enums	1	1	0	1	2	0.0%	67.0%	33.0%	1
com.caplin.datasrc.fields	3	1	2	4	2	67.0%	33.0%	0.0%	1
com.caplin.datasrc.interfaces	8	0	8	2	3	100.0%	60.000004%	60.000004%	1
com.caplin.event	1	0	1	1	2	100.0%	67.0%	67.0%	1
com.caplin.net.udp	6	3	3	2	4	50.0%	67.0%	17.0%	1
com.caplin.server.jni	3	2	1	1	4	33.0%	80.0%	13.0%	1
com.caplin.server.logging	4	3	1	4	8	25.0%	67.0%	8.0%	1
com.caplin.transformer.module	28	14	14	6	15	50.0%	71.0%	21.0%	1
com.caplin.transformer.module.datasrc	11	7	4	6	7	36.0%	54.000004%	10.0%	1
com.caplin.transformer.module.datasrc.fields	2	2	0	2	5	0.0%	71.0%	29.0%	1
com.caplin.transformer.module.exceptions	2	2	0	1	1	0.0%	50.0%	50.0%	1
com.caplin.transformer.module.subscriber	12	12	0	1	7	0.0%	88.0%	12.0%	1
com.caplin.transformer.module.udp	1	1	0	1	3	0.0%	75.0%	25.0%	1
com.caplin.transformer.module.wrappers	1	1	0	0	2	0.0%	100.0%	0.0%	1
com.caplin.transformer.modules.cfm	7	7	0	2	7	0.0%	78.0%	22.0%	1
com.caplin.transformer.modules.cfm.container	4	3	1	1	6	25.0%	86.0%	11.0%	1
com.caplin.transformer.modules.cfm.filter	1	1	0	2	4	0.0%	67.0%	33.0%	1
com.caplin.transformer.modules.cfm.record	2	2	0	1	4	0.0%	80.0%	20.0%	1
com.caplin.util	1	1	0	1	3	0.0%	75.0%	25.0%	1
com.caplin.util.base64	2	2	0	1	1	0.0%	50.0%	50.0%	1
com.caplin.util.classloader	1	1	0	0	4	0.0%	100.0%	0.0%	1

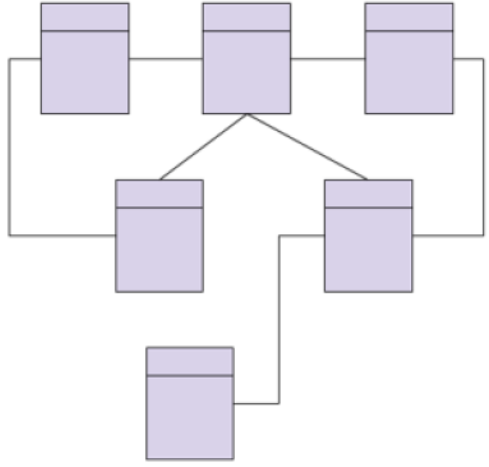
Packages

[summary] [packages] [cycles] [explanations]

com.caplin.datasrc.enums

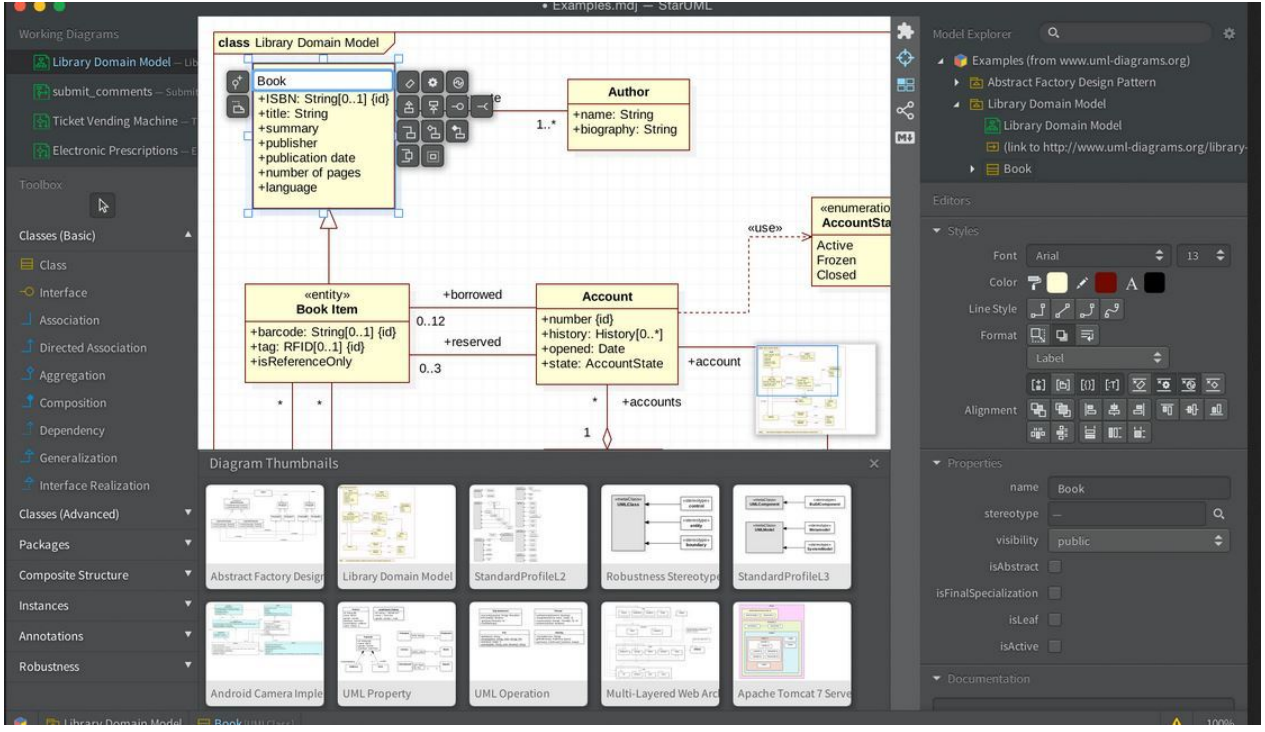
Static Testing Tools

- Modeling tools (D)
  - Validate models of the system/software
- Purpose
  - To better aid in designing the software
- Common features and characteristics
  - Identify inconsistencies and defects within the models
  - Identify and prioritize risk areas
  - Predicting system response and behavior under various situations



Static Testing Tools

- Modeling tool example: StarUML
  - UML tool
  - Variety of diagrams
    - Class/Domain
    - Use case
    - Sequence



Static Testing Tools

- The major benefit of static testing tools and modeling tools is the cost effectiveness of finding more defects at an earlier time in the development process.
- As a result, the development process may accelerate and improve by having less rework.

Test design and specification Tools

- Test design tools
- Model-based testing tools
- Test data preparation tools
- TDD -Test driven development tool (D)
- ATDD -Acceptance test-driven development and
- BDD -Behavior-driven development

Test design and specification Tools

- **Test design tools**
  - generating test inputs values from:
    - requirements
    - test conditions
    - design models (state, data or object)
    - code
    - graphical user interfaces
  - generating expected results, if an oracle is available to the tool.

Test design and specification Tools

- **Model-based testing**
  - Generating test inputs values or test cases from stored information the describes a model of the system, e.g. state transition model

- **Test data preparation tools**
  - Generating extensive range or volume of data, if needed
  - Manipulate databases or files to set up test data to be used during the execution of the tests

Test design and specification Tools

- **TDD -Test driven development tool (D)**
  - part of EXtremeProgramming
  - used in Agile development

- **ATDD -Acceptance test-driven development and**
- **BDD -Behavior-driven development**
  - Natural language syntax, Given/When/Then:

    Given<some condition> , When<something is done>, Then <result should happen>

    As a <role, e.g. costumer>

    In order to <achieve something, e.g. product>

    I want <do something>

Test execution & logging Tools

- Test execution tools
- Coverage tools
- Test harnesses (D) and Unit test framework tools (D)
- Test comparators

Test execution & logging Tools

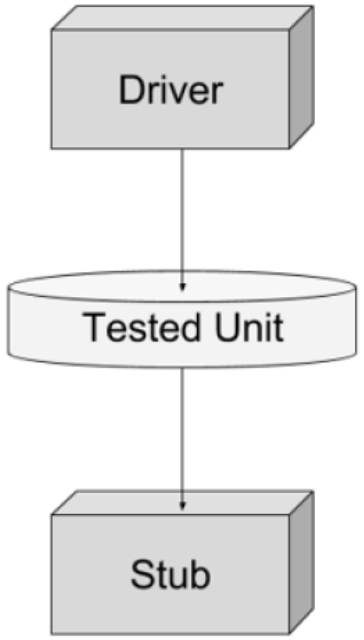
- **Test execution tools**
  - Enable tests to be executed automatically using stored
  - Inputs and
  - expected results
- **Coverage tools**
  - Identifying coverage items(instrumenting the code)
  - calculating the percentage of coverage items that were exercised by a suite of tests
  - reporting coverage items that have not been exercised as yet
  - identifying test inputs to exercise as yet uncovered items
  - generating stubs and drivers (if part of a unit test framework).

Test execution & logging Tools

- **Test harnesses (D) and Unit test framework tools (D)**
  - supplying inputs to the software being tested
  - receiving outputs generated by the software being tested
  - executing a set of tests within the framework or using the test harness
  - recording the pass/fail results of each test (framework tools)
  - storing tests (framework tools)
  - support for debugging (framework tools)
  - coverage measurement at code level (framework tools)

Test execution & logging Tools

- Drivers
  - Calls the component to be tested
  - In other words: A component that calls the Tested Unit
- Stubs
  - Called from the software component to be tested
  - In other words: A component the Tested Unit depends on
  - Partial implementation
  - Fake values



Test execution & logging Tools

- **Test harnesses (D) and Unit test framework tools (D)**
  - The two types are similar
  - Support tools for testing individual componenets or software units
  - Harness: Stubs and drivers --> Small programs that interact with software
  - Unit test framework tools --> Support for object-oriented software
  - When are these tools used?
    - During test execution and logging

Test execution & logging Tools

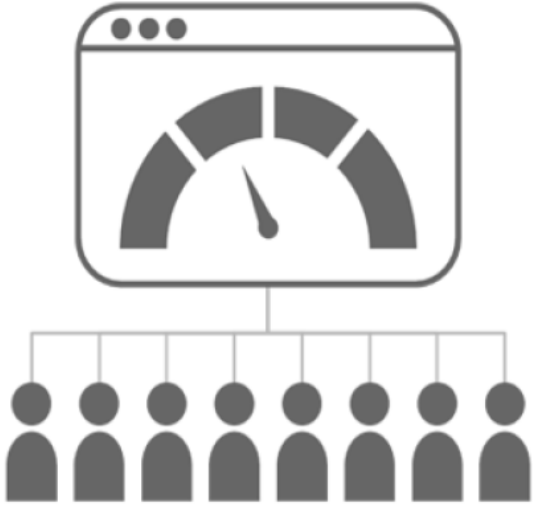
- Characteristics of test harness and unit test framework tools
  - Supply inputs to the software being tested
  - Receive outputs generated by the software being tested
  - Execute a set of tests within the framework
  - Record pass/fail results of each test
  - Store tests
  - Coverage measurement at code level
  - Provide support for debugging

Test execution & logging Tools

- **Test comparators**
  - used when the executed test generates a lot of output.
- Testing is more than providing inputs
  - Need to check if software produces the correct result
  - Compare actual outcomes to expected results
- Two ways of comparing results
  - Dynamic comparison --> Comparison done during test execution
  - Post-executing comparison --> Comparison performed after test has finished
    - Software under test is no longer executing

Performance & monitoring Tools

- Test data for performance testing
  - Real data
    - Test data obtained from users
  - Load
    - Large amounts of test data can be produced
  - Maintenance
    - Test data from the production environment



Performance & monitoring Tools

- Tests should reflect realistic (correct) scenarios
  - Systems are often required to handle significant load/interactions
  - Inadequate/insufficient testing compromises system quality
- Setting up test data --> Significant effort
  - Extensive range or volume of data needed
  - Creating this data can be very resource-consuming
- Test data preparation tools help us manage this effort

Performance & monitoring Tools

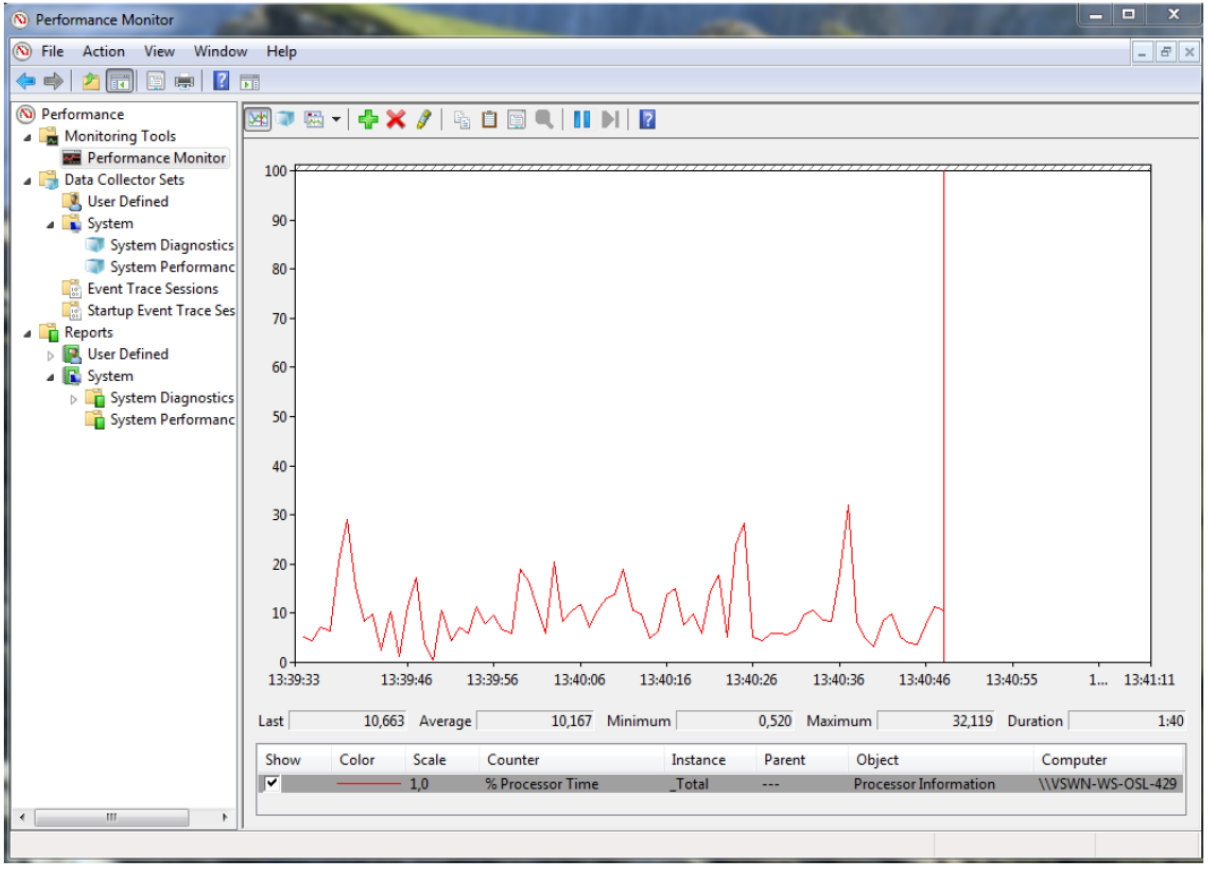
- Common features of test data preparation tools
  - Data can be selected from an existing database
  - Data can be created, generated, and altered for use in tests
  - Construct a large number of similar records --> Volume tests
- When to use?
  - During test specification and control --> Test data management is difficult
  - Ensure the system under test is being tested realistically
  - Useful for performance and reliability testing

Monitoring tools

- **Monitoring tools** identifying problems and sending an alert message to the administrator (e.g. network administrator)

  - logging real-time and historical information
  - finding optimal settings
  - monitoring the number of users on a network
  - monitoring network traffic (either in real time or covering a given length of time of operation with the analysis performed afterwards).

Monitoring tools



Performance & monitoring Tools

- Features or characteristics of dynamic analysis tools include support for
  - detecting memory leaks;
  - identifying pointer arithmetic errors such as null pointers;
  - identifying time dependencies

Dynamic analysis tools (D)

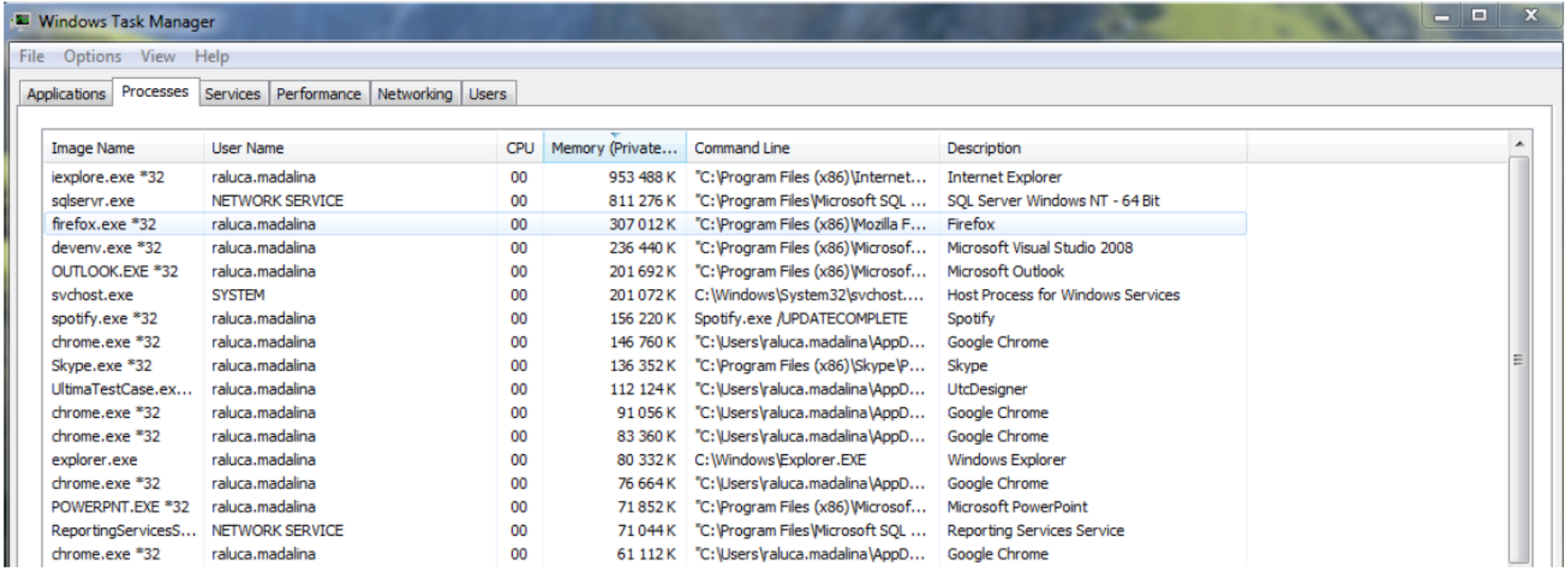


Image Name	User Name	CPU	Memory (Private...	Command Line	Description
iexplore.exe *32	raluca.madalina	00	953 488 K	"C:\Program Files (x86)\Internet...	Internet Explorer
sqlservr.exe	NETWORK SERVICE	00	811 276 K	"C:\Program Files\Microsoft SQL ...	SQL Server Windows NT - 64 Bit
firefox.exe *32	raluca.madalina	00	307 012 K	"C:\Program Files (x86)\Mozilla F...	Firefox
devenv.exe *32	raluca.madalina	00	236 440 K	"C:\Program Files (x86)\Microsof...	Microsoft Visual Studio 2008
OUTLOOK.EXE *32	raluca.madalina	00	201 692 K	"C:\Program Files (x86)\Microsof...	Microsoft Outlook
svchost.exe	SYSTEM	00	201 072 K	C:\Windows\System32\svchost....	Host Process for Windows Services
spotify.exe *32	raluca.madalina	00	156 220 K	Spotify.exe /UPDATECOMPLETE	Spotify
chrome.exe *32	raluca.madalina	00	146 760 K	"C:\Users\raluca.madalina\AppD...	Google Chrome
Skype.exe *32	raluca.madalina	00	136 352 K	"C:\Program Files (x86)\Skype\P...	Skype
UltimaTestCase.ex...	raluca.madalina	00	112 124 K	"C:\Users\raluca.madalina\AppD...	UtcDesigner
chrome.exe *32	raluca.madalina	00	91 056 K	"C:\Users\raluca.madalina\AppD...	Google Chrome
chrome.exe *32	raluca.madalina	00	83 360 K	"C:\Users\raluca.madalina\AppD...	Google Chrome
explorer.exe	raluca.madalina	00	80 332 K	C:\Windows\Explorer.EXE	Windows Explorer
chrome.exe *32	raluca.madalina	00	76 664 K	"C:\Users\raluca.madalina\AppD...	Google Chrome
POWERPNT.EXE *32	raluca.madalina	00	71 852 K	"C:\Program Files (x86)\Microsof...	Microsoft PowerPoint
ReportingServicesS...	NETWORK SERVICE	00	71 044 K	"C:\Program Files\Microsoft SQL ...	Reporting Services Service
chrome.exe *32	raluca.madalina	00	61 112 K	"C:\Users\raluca.madalina\AppD...	Google Chrome

Specific application areas Tools

- There are tools specialized for use in a particular type of application
  - Data quality assessment
  - Data conversion and migration
  - Usability testing and Accessibility testing
  - Localization testing
  - Security testing
  - Portability testing

Specific application areas Tools

- There are tools specialized for use in a particular type of application.
- Example:
  - performance testing tools specifically for web-based applications
  - dynamic analysis tools specifically for testing security aspects.
- Example of targeted areas: embedded systems

Specific application areas Tools

- Testers may use:
  - word processor
  - spreadsheets

  as a testing tool, but they are often used to store
    - test designs
    - test scripts
    - test data.
- Testers may also use SQL to set up and query databases containing test data.
- Tools used by developers when debugging , to help localize defects and check their fixes, are also testing tools.
- It is a good idea to look at any type of tool available to you for ways it could be used to help support any of the testing activities

Effective use of test tools

- Simply purchasing or leasing a tool does not guarantee success with that tool!
- Each type of tool may require additional effort to achieve real and lasting benefits.

Potential benefits and risks

- Reduce repetitive work (running regression tests, re-entring the same test data, etc.)
- Greater consistency and repeatability (tests executed by a tool, tests derived from requirements)
- Objective assessment (static measures, coverage)
- Ease of access to information about tests, or testing (statistics/graphs about test progress, incident rates, performance)

Potential benefits and risks

- Greater consistency and repeatability
  - People tend to do the same tasks in a slightly different way
    - Distractions affect human performance
    - Doing more than one task simultaneously
    - Interruptions by peers / co-workers
    - Fatigue and personals issues
    - External pressures
  - Tools will reproduce the exact same procedure as previously

Potential benefits and risks

- Objective assessment
  - Humans are prone to make errors
  - Subjective preconceived notions and bias toward verification
  - Testing tools on the other hand …
    - Objective “preconceived notions”
    - Assessment → Repeatable and consistently calculated
    - Cyclomatic complexity, nesting levels
    - Coverage, system behavior, incident statistics

Potential benefits and risks

- Ease of access to information about the tests or test effort
  - Information presented visually
  - Easier for the human mind to understand
    - Chart, graphs > Long list of numbers
  - Special purpose tools provide features directly
    - Statistics and graphs
    - Incident rates
    - Performance

Potential risks

- Unrealistic expectations for the tool (functionality and ease of use)
- Underestimating time, cost, effort for the introduction of a tool (training, external expertise)
- Underestimating the time and effort needed to achieve significant and continuing benefits from the tool
- Underestimating the effort required to maintain the test assets generated by the tool
- Over-reliance on the tool (replacement where manual testing would be better)

Potential benefits and risks

- Tools are not magic!
- They can do very well what they have been designed to do, but they can not do everything.
- The tester concentrates on
  - What should be tested
  - What the test cases should be
  - How to prioritize the testing
- The tool user concentrates on
  - How best to get the tool to do its job effectively
  - How to give increasing benefit from tool use

Special considerations: Test execution tools

Not important.

- This type of tool often requires significant effort in order to achieve significant benefits.
  - Capturing tests by recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of automated tests. This type of script may be unstable when unexpected events occur.
  - Data-driven approach: separates out the test inputs (the data) and uses a more generic script that can read the test data and perform the same test with different data.
  - In a keyword-driven approach: the spreadsheet contains keywords with the actions to be taken (also called action words), and test data. Testers can then define tests using the keywords.

Special considerations: Performance testing tools

- The design of the load to be generated by the tool
- Timing aspects probe effect
- How to interpret the information gathered.
- These tools need tester with expertise in performance testing to design the tests and interpret results

Special considerations: Static analysis tools

- There is a risk that the changes to make old code to conform to new standard will introduce an unexpected side effect
- These tools applied to source code can enforce coding standards, but if applied to existing code may generate a lot of messages
- A gradual implementation with initial filters to exclude some messages would be an effective approach.

Special considerations: Test management tools

- They need to interface with other tools or spreadsheets in order to produce information in the best format for the current needs of the organization.

Interview Questions and Answers

Fundamental Questions in Testing

- When can we stop testing?
  - Test coverage
- What should we test?
  - ✓Test generation
- Is the observed output correct?
  - ✓Test oracle
- How well did we do?
  - Test efficiency
- Who should test your program?
  - ✓Independent V&V

Interview Questions

- What is difference between QA, QC and Software Testing?
- What is verification and validation?
- Explain Branch Coverage and Decision Coverage.
- What is pair-wise programming and why is it relevant to software testing?
- Why is testing software using concurrent programming hard? What are races and why do they affect system testing.
- Phase in detecting defect: During a software development project two similar requirements defects were detected. One was detected in the requirements phase, and the other during the implementation phase.
- Why do we measure defect rates and what can they tell us?
- What is Static Analysis?

What is difference between QA, QC and Software Testing?

- **Quality Assurance (QA)**: QA refers to the planned and systematic way of monitoring the quality of process which is followed to produce a quality product. QA tracks the outcomes and adjusts the process to meet the expectation. QA is **not** just testing.
- **Quality Control (QC)**: Concern with the quality of the product. QC finds the defects and suggests improvements. The process set by QA is implemented by QC. The QC is the responsibility of the tester.
- **Software Testing**: is the process of ensuring that product which is developed by the developer meets the user requirement. The motive to perform testing is to find the bugs and make sure that they get fixed.

Verification And Validation

What is verification and validation?

- **Verification:** process of evaluating work-products of a development phase to determine whether they meet the specified requirements for that phase.
- **Validation:** process of evaluating software during or at the end of the development process to determine whether it meets specified requirements.

Branch Coverage and Decision Coverage

Explain Branch Coverage and Decision Coverage.

- **Branch Coverage** is testing performed in order to ensure that every branch of the software is executed at least once. To perform the Branch coverage testing we take the help of the Control Flow Graph.
- **Decision coverage** testing ensures that every decision taking statement is executed at least once.
- Both decision and branch coverage testing is done to ensure the tester that no branch and decision taking statement will lead to failure of the software.
- To Calculate Branch Coverage:
  - **Branch Coverage = Tested Decision Outcomes / Total Decision Outcomes.**

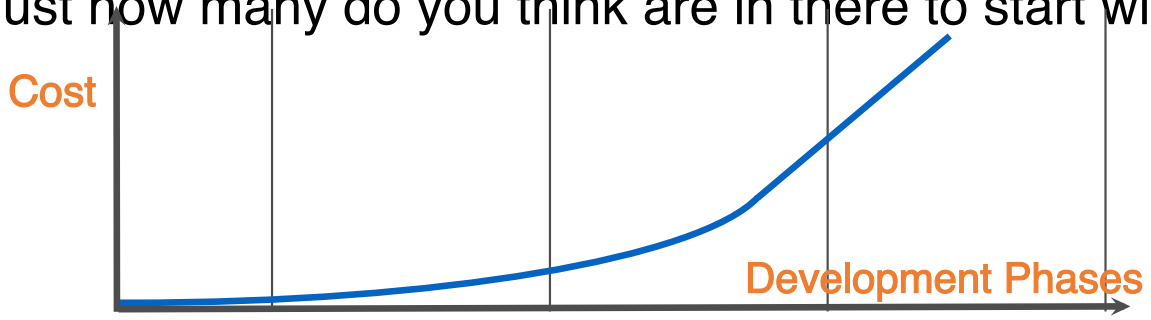
What is Static Analysis?

- The term "static analysis" is conflated, but here we use it to mean a collection of algorithms and techniques used to analyze source code in order to automatically find bugs.
- The idea is similar in spirit to compiler warnings (which can be useful for finding coding errors) but to take that idea a step further and find bugs that are traditionally found using run-time debugging techniques such as testing.
- Static analysis bug-finding tools have evolved over the last several decades from basic syntactic checkers to those that find deep bugs by reasoning about the semantics of code.

Defect Costs

Questions:

- When you find one, how much will it cost to fix?
  - How much depends on when the defect was created vs. when you found it?
- Just how many do you think are in there to start with?!



The cost of fixing a defect rises exponentially by lifecycle phase

But this is simplistic

- When were the defects injected?
- Are all defects treated the same?
- Do we reduce costs by getting better at fixing or at prevention?

“QA” & Testing

- Testing “Phases”
  - ✓Unit
  - ✓Integration
  - ✓System
  - ✓User Acceptance Testing
- Testing Types
  - ✓Black-box
  - ✓White-box
- Static vs. Dynamic Testing
- Automated Testing
  - Pros and cons
- Integration: 2 types
  - ✓Top down
  - ✓Bottom up

Quality Assurance (QA)

- **Definition - What does *Quality Assurance (QA)* mean?**
- Quality assurance (QA) is the process of verifying whether a product meets required specifications and customer expectations. QA is a process-driven approach that facilitates and defines goals regarding product design, development and production. QA's primary goal is tracking and resolving deficiencies prior to product release.

- The QA concept was popularized during World War II.

Software Quality Assurance

- **Software quality assurance** (SQA) is a process that ensures that developed **software** meets and complies with defined or standardized **quality** specifications.
- SQA is an ongoing process within the **software** development life cycle (SDLC) that routinely checks the developed **software** to ensure it meets desired **quality** measures.

Software Quality Assurance

- The area of Software Quality Assurance can be broken down into a number of smaller areas such as
  - Quality of planning,
  - Formal technical reviews,
  - Testing

  and
  - Training.

Professional Ethics

- If you can't test it, **don't build it**
- Put **quality first** : Even if you lose the argument, you will gain respect
- Begin test activities **early**
- **Decouple**
  - **Designs** should be independent of language
  - **Programs** should be independent of environment
  - Couplings are **weaknesses** in the software!
- **Don't take shortcuts**
  - If you lose the argument you will **gain respect**
  - **Document** your objections
  - **Vote** with your feet
  - Don't be afraid to be **right**!

Professional Ethics

- Recognizing the ACM and IEEE code of ethics for engineers:
- PUBLIC - Software testers shall act consistently with the public interest.
- CLIENT AND EMPLOYER - Software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
- PRODUCT - Software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible.
- JUDGMENT - Software testers shall maintain integrity and independence in their professional judgment.
- MANAGEMENT - Software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing.
- PROFESSION - Software testers shall advance the integrity and reputation of the profession consistent with the public interest.
- COLLEAGUES - Software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers.
- SELF - Software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.