Software Testing Overview on Different Generalization Levels

Journal:	*Computer*
Manuscript ID:	COMSI-2013-07-0127
Manuscript Type:	Special Issue: Software Testing
Date Submitted by the Author:	10-Jul-2013
Complete List of Authors:	Kuļešovs, Ivans; University of Latvia, Computer Science Arnicāne, Vineta; University of Latvia, Computer Science Arnicāns, Guntis; University of Latvia, Computer Science Borzovs, Juris; University of Latvia, Computer Science
Keywords:	D.2.5.k Testing strategies < D.2.5 Testing and Debugging < D.2 Software Engineering < D Software/Software Engineering, D.2.19 Software Quality/SQA < D.2 Software Engineering < D Software/Software Engineering, D.2.5 Testing and Debugging < D.2 Software Engineering < D Software/Software Engineering

Software Testing Overview on Different Generalization Levels

Ivans Kuļešovs
Faculty of Computing
University of Latvia
19 Raina *Blvd.*, Riga,
LV-*1586*, Latvia

Vineta Arnicane
Faculty of Computing
University of Latvia
19 Raina *Blvd.*, Riga,
LV-*1586*, Latvia

Guntis Arnicans
Faculty of Computing
University of Latvia
19 Raina *Blvd.*, Riga,
LV-*1586*, Latvia

Juris Borzovs
Faculty of Computing
University of Latvia
19 Raina *Blvd.*, Riga,
LV-*1586*, Latvia

1. Introduction

There are many different views on software testing co-exist even within the borders of one organization. That is why we have decided to prepare software testing overview on meta-level indicating main influencers that make this difference. While gathering the details about meta-level elements we have performed some structuring of elements from lower level of software testing such as testing oracles and testing approaches, methods, and techniques. The overview preparation has resulted into laying the scientific basis under proper use of such terms as testing approach, testing method, and testing technique.

Our overview could be useful for making ordered introduction into software testing for fresh minds of testing newbies, while we recognize that in practice it can be hard to make sharp-cut edges between some software testing elements described here.

It is worth to mention that test management itself is out of scope of this study.

2. Software Testing Overview on Meta-level

Software testing mainly consists from testing strategy and testing tactics on the meta-level (i.e. on the higher level of abstraction). Other things like contexts and schools influence on the selection of the strategy and/ or tactics. Software testing overview on meta-level is depicted on Figure 1.

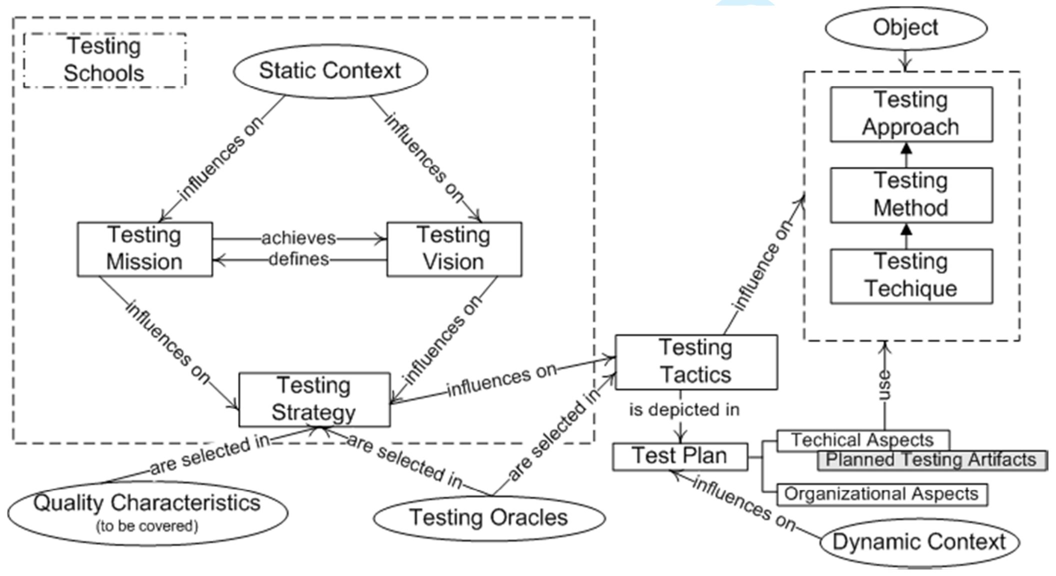

Figure 1. General testing process schematic view

Static context influences very much on the testing vision and testing mission. Static context depends on the type of the organization (i.e. governmental, outsourcer, start-up etc.) and on the type of the software produced (enterprise software, commercial software, web page etc.). It is generally static during the whole product lifecycle. Testing vision is what we want to achieve by testing. In some cases it can be to make software with all high and critical bugs discovered and fixed, and 95% of medium severity bugs identified. In some cases it can be to receive an acceptance sign-off of the product from the customer. Testing mission is what we do to achieve the testing vision. For example, we use only scripted testing, or we use the benefits of the exploratory testing as well, to receive an acceptance sign-off of the product from the customer. Or we prepare automated tests before the development to keep our product always deliverable to the customer as test-driven development suggests. Testing schools are frameworks that define testing vision and testing mission based on the static context.

All aspects of testing schools (it can also be the mix of aspects from different schools) that prevail within the organization and is common for definite product type influence on the testing strategy of the given software project. Testing strategy describes general approach for testing. Testing strategy consists of the specification of the roles and responsibilities, test levels, environment requirements, test schedule, testing tools, risks and its' mitigations, testing priorities, testing status reporting etc.

Testing oracles that define testing exit-criteria and that are used as the source of the derivation of test cases should be chosen within the testing strategy definition. The selection of quality characteristics to be covered by testing process should occur during the definition of testing strategy as well. Test results completeness oracles can be defined when selecting testing tactics, because often there are much more details about expected results available during tactics selection process.

Dynamic context depends on the project phase and influences on the choice of the testing tactics that are appropriate for the given time frame, for the definite object under test, and for the current micro testing goal. Examples of dynamic context factors are fulfillment of test entry criteria in time, availability of shared testing resources, the stabilization and bug fixing phase of the development etc. Testing tactics should be consistent with the testing strategy.

Testing tactics for each object under test are depicted in the test plan. Test plan consists of organizational and technical aspects. Testing tactic also influences on the choice of the testing approach to be used to fulfill current micro testing goals. Thus, technical aspects of the test plan should include the selection of the appropriate testing approaches, methods, and techniques that make a graph. Testing artifacts (like test cases, test suites, traceability matrix, test data etc.) to be produced by the testing process should be mentioned in the test plan as well. It is worth to notice that some schools do not make formal and written test plans as mandatory artifact of testing process.

3. Testing Schools[1]

Testing society distinguishes five testing schools. They are:

- Analytic School

- Standard School
- Quality School
- Context-Driven School
- Agile School

The schools are frameworks for categorization of test engineers' believes about testing and they guide the testing process. Testing schools are not competitive; they can be used in collaborative mode as well. They all have exemplar techniques or paradigms, but they are not limited to them. Usage of schools can vary within the organization from project to project, but it is often hard to move the whole organization from one school to another.

Analytic school assumes that software is a logical artifact. It concentrates on technical aspects, and it is keen on the white-box testing. Analytic school is associated with academia institutions, and it is assumed to be the most suitable for safety-critical and telecom software.

Standard school assumes that testing should be very well planned in advance and managed. According to this school, testing main goal is to validate that software meets contractual requirements and/or governmental standards using the most cost-effective model, thus it is mostly applied for governmental and enterprise IT products. Requirements traceability matrix is the most common testing artifact for the school. Software testing can be seen like assembly line through V-model prism. IEEE standards' boards and testing certifications are the most valued institutions by this school.

Quality school prefers "Quality Assurance" over "Testing". Thus testing defines and controls the development processes. QA manager or test lead is like a gatekeeper who can decide if software is ready or not. ISO and CMMI are the most valued institutions for followers of this school.

Context-driven school concentrates about (skilled) people and their collaboration. The goal of context-driven testing is to find bugs that can bother any of the stakeholders. What to test right now is defined according to the current situation in the project. Test plans to be constantly adapted based on the test results. Exploratory testing is this school exemplar technique. Context-driven testing is mostly applied for the commercial, market-driven software. The Los Altos Workshop on Software Testing held by Cem Kaner and Brian Lawrence are thought to be the main events of this school.

Agile school main postulate is that tests must be automated. Testing answers the question if user story is done. Test-driven development is one of the agile testing school paradigms, thus unit tests are demonstrative exemplar of the school.

It is worth to mention that categorization of beliefs and testing goals into testing schools helps testers to understand and to evaluate each other experience through the prism of the specific organizational context.

4. Testing Strategy

There are many things to be covered in testing strategy that define the overall approach of testing. Here we only look into details of its most important aspects that are noticeable on the software testing meta-level.

4.1. Testing Oracles

Testing Oracles can be divided into three major groups based on their purpose. They and oracles per each group are shown on Figure 2.

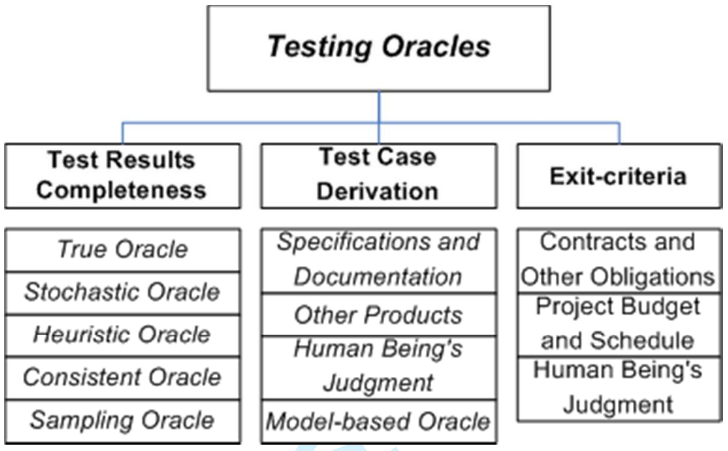

Figure 2. Testing Oracles

Test results completeness oracles are differentiated based on the completeness of the set of the expected test results. There are five main types of test results oracles.[2] They are:

- True oracles – they have the complete set of expected test results;
- Stochastic oracles – they verify a randomly selected sample;
- Heuristic oracles– they can verify correctness of some values and consistency of other values;
- Consistent oracles – they verify current test run results with previous test run results (regression);
- Sampling oracles – they select the specific collection of inputs or results.

They all have their advantages and disadvantages, as well as their cost decreases in top-down manner, but speed increases in the same manner.

Test case derivation oracles are differentiated based on the source test cases are derived from.

Exit-criteria oracles define when testing can be finished. The most common, but not complete testing exit-criteria are:

- All planned test cases are executed (Contracts and other obligations);
- All high and critical priority bugs are fixed (Contracts and other obligations);
- All planned requirements are met (Contracts and other obligations);
- Scheduled time to finish testing has come (Project budget and schedule oracle);

- Test manger has signed off the release (Human being's judgment oracle).

It is worth to mention that multiple oracles of each group are often used together depending on the software project phase.

4.2. Quality characteristics

There are 8 quality characteristics shown in the new revision of ISO/IEC 9126 standard - ISO/IEC 25010.[3] All quality characteristics are depicted on Figure 3.

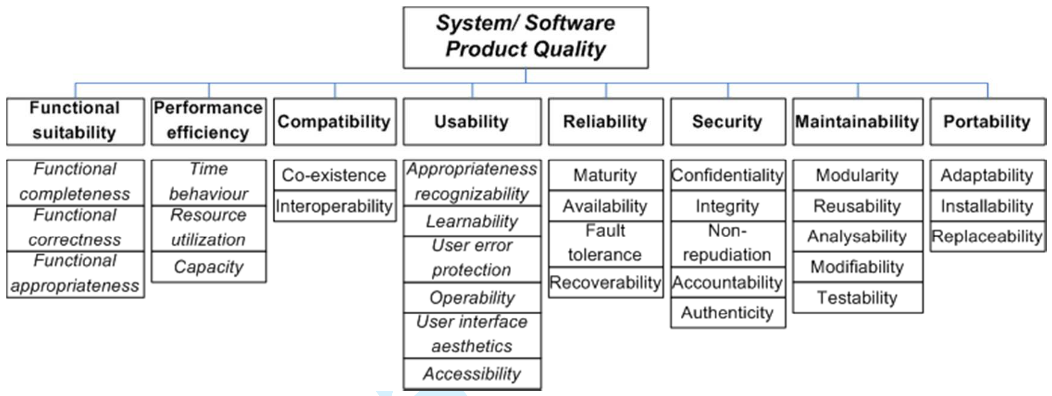

Figure 3. Product Quality Model[3]

Functional testing is a testing of functional suitability characteristic. Almost all formal testing methods and techniques are concentrated around functional suitability quality characteristic as well and especially are related to the functional correctness and functional completeness sub-characteristics.

5. Testing Tactics

Testing tactics can differ depending on the phase of the project and other changeable circumstances of the environment. Testing tactics should be consistent with testing strategy. Thus tactics often are chosen within the static boundaries of the influencer schools. Appropriate testing approaches, methods, and techniques should be selected for micro testing goals fulfillment and should be depicted in the testing plan. We have structured testing methods and techniques under black-box and white-box approaches. The borders of grey-box testing approach are quite ambiguous, and methods and techniques under this approach are not formally described yet in the testing theory. They do not have settled definitions in the testing practice as well. It is worth to mention that detailed definitions of techniques mentioned bellow can be found in the works of such authors as Black[4], Jorgensen[5], Beizer[6], and Kaner[7].

5.1. Black-box Testing

Black-box is a software testing approach when test engineer designs test cases as if she does not know anything about the internal structure of the software under test.

Black-box testing approach consists of six testing methods that are differentiated based on the source used for test case design process and based on the level of formality of test case designs. The relation between black-box testing methods and techniques is shown in Figure 5.

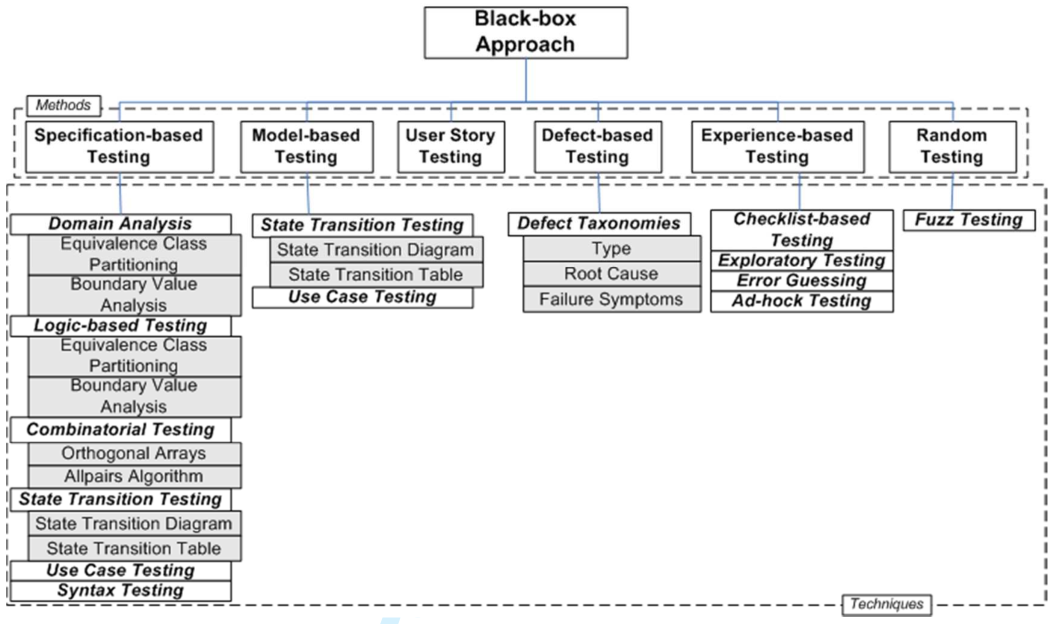

Figure 5. Black-box Approach

5.2. White-box Testing

White-box is a software testing approach when test engineer designs test cases based on the internal structure of the software under test. There are three most known white box testing methods are control flow testing, data flow testing, mutation testing. The relation between white-box testing methods and techniques is shown in Figure 6.

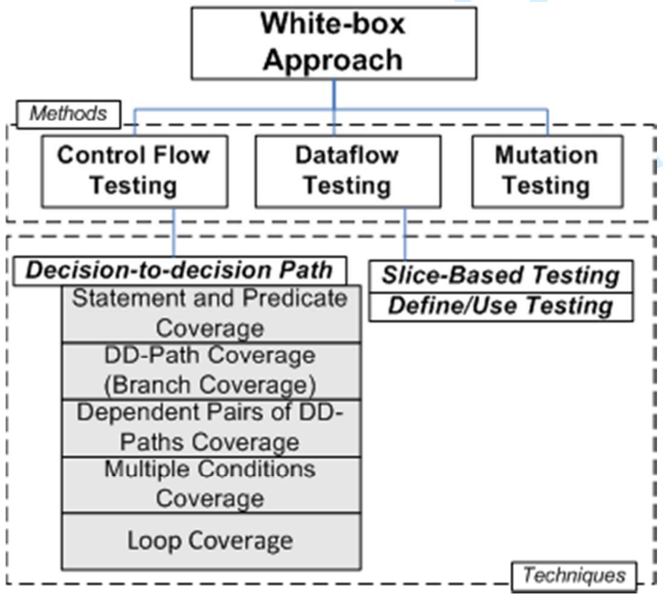
White-box Approach
Methods
Control Flow Testing
Dataflow Testing
Mutation Testing
Decision-to-decision Path
Statement and Predicate Coverage
DD-Path Coverage (Branch Coverage)
Dependent Pairs of DD-Paths Coverage
Multiple Conditions Coverage
Loop Coverage
Slice-Based Testing
Define/Use Testing
Techniques

Figure 6. White-box Approach

Some static testing techniques are used for software code testing. They differ based on the formality and thoroughness of the process. Code review is often used to improve the overall quality of the code and to educate less experienced developers. This process helps to deliver more qualitative and tested code from development to testing right at the moment, but educative aspects help to improve the quality of the code for the future deliveries. Inspections and walkthroughs are used when there is less time available to conduct the static testing process.

6. Conclusions

We suppose that our work will help software testing practicians and those who just have started to learn software testing to understand aspects of software testing in more holistic and structured way, as well as to start using such terms as testing approach, testing method, and testing technique in a proper way. We plan to continue the inventory of software testing on lower levels that potentially can result into structuring and categorization of testing terms and ideas. We also encourage the readers to share the testing methods and techniques they think we have missed to make the full picture.

References

1. B. Pettichord, *Schools of Software Testing*, 2008; http://www.prismnet.com/~wazmo/papers/four_schools.pdf.
2. D. Hoffman, *A Taxonomy for Test Oracles*, in Quality Week, 1998; http://www.softwarequalitymethods.com/Papers/OracleTax.pdf.
3. ISO (2011), *ISO/IEC 25010:2011*
4. R. Black, *Advanced Software Testing – Vol.1*, Santa Barbara, CA: Rock Nook Inc., 2009
5. P.C. Jorgensen, Software Testing: A Craftsman's Approach, 3rd Edition, Boca Raton, FL: Auerbach Publications, 2008
6. B. Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems*, New York: John Wiley & Sons, Inc., 1995.
7. C. Kaner, J. Falck, and H. Nguyen, Testing Computer Software, 2nd Edition, John Wiley & Sons, Inc., 1999.
8. ISTQB, in van Veenendaal, E. Ed., *Standard glossary of terms used in Software Testing, 2012;* http://www.istqb.org/downloads/finish/20/101.html.
9. R. Pressman, *Software Engineering: A Practitioner's Approach*, 6th Edition, Singapore: McGraw-Hill, 2005.
10. J. Sommerville, *Software Engineering*, 8th Edition, Harlow, Essex: Pearson Education Limited, 2007.
11. B. Kumaravadivelu, *UNDERSTANDING LANGUAGE TEACHING: From Method to Postmethod*, Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 2006.
12. G. Hall, *Exploring English Language Teaching: Language in Action*. New York: Routledge, 2011.

Ivans Kuļešovs is a PhD student in Computer Science at the University of Latvia and Test Manager in C.T.Co Ltd. software development company. His research interests include software testing in general and mobile applications testing in particular, as well as enterprise mobility platforms.
Kuļešovs received his master degree with distinction in Computer Science from University of Latvia and MBA degree from Blekinge Institute of Technology, Sweden. Contact him at ivans.kulesovs@gmail.com.

Vineta Arnicane is a Senior Researcher in the Faculty of Computing at the University of Latvia. Her research interests include software engineering, software testing, and artificial intelligence. Arnicāne received her PhD in computer science from the University of Latvia. Contact her at vineta.arnicane@lu.lv.

Guntis Arnicans is a Professor and Director of Bachelor program "Computer science" in the Faculty of Computing at the University of Latvia. His research interests include software engineering, software testing, and artificial intelligence, with a focus on creating concept map and ontology for software testing domain. Arnicāns received a PhD in computer science from the University of Latvia. He is a member of IEEE and ACM. Contact him at guntis.arnicans@lu.lv.

Juris Borzovs is currently Professor and Dean of the Faculty of Computing at the University of Latvia. His research interests include software engineering, software quality, software testing, and IT terminology. Borzovs received his candidate of science degree from the Institute of Mathematics of Belarusian Academy of Science, doctor of science degree and doctor habilitatus degree from the University of Latvia.He is a member of several organizations that focus on information technology. Contact him at juris.borzovs@lu.lv.

Sidebars

Testing Controversies

There are many controversies exist in software testing. Some of them clearly belong to definite testing school. Others are controversial because of other reasons, for example project phase. It is worth to mention that controversies mentioned below, despite their difference, make good testing when used together proportionally.

The controversy we should start with is **testing vs. debugging** controversy. The goal of testing is to discover the defect while the goal of debugging is to find why the defect occurs. Some schools see debugging as job of software developer only, but nowadays it is more common for good test engineer to investigate the root cause of the defect by himself or together with software developer.

The most known testing controversy is **black-box testing vs. white-box testing**. The difference between them is the point of view on the knowledge of the internal structure of the software that test engineer takes when designing the test cases.

Functional testing vs. non-functional testing is another important testing controversy. Functional testing verifies software against the specification. Non-functional testing checks software against its non-functional requirements where non-functional quality characteristics are addressed.

Another quite old controversy is **manual testing vs. automated testing**. Return on investment is taken into consideration when testing is automated, as it requires skillful workforce and additional scripting and maintenance effort. Still, only part of the testing can be automated. UI automation is often used for regression testing, while unit and integration tests can be written in advance to development.

These two testing ideas are very different by their nature: **scripted testing vs. exploratory testing**. Scripted testing can show the thoroughness of the testing to stakeholders, while exploratory testing can find bugs that hardly could be discovered when using scripted testing, because it is sometimes even hard to imagine the appropriate test cases before investigating the behavior of the new functionality under test.

Another controversy consists of one of the oldest testing ideas: **verification vs. validation**. Controversy **contract vs. client happiness** is closely connected to the testing missions represented above, thus, depending on this, different testing strategies are chosen. Verification evaluates if product meets the requirements that usually are part of the contract while validation check if product satisfy the clients (or other stakeholders) expectations, i.e. makes them happy.

Positive testing vs. negative testing controversy parts are hardly separated if it is needed to make testing as complete as possible. Positive testing tends to prove that software behaves in the way it is supposed to. Negative testing shows that software does not do that it is not supposed to.

Testing of design vs. testing of implementation identifies different testing needs depending on the software project phase. Thus different testing tactics to be used during each phase. Testing of design also uncovers the idea that testing should be started as early as possible.

Static testing vs. dynamic testing controversy intersects with previously mentioned controversy. Testing of designs is always a **static testing**, i.e. testing process without executing the software itself. Testing of implementation (except the review of the code) in most cases is a **dynamic testing**, i.e. testing of the running software.

Hierarchical vs. big bang are different approaches of the integration testing. There are two hierarchical integration testing approaches: **bottom-up** and **top-down**. When bottom-up approach is used then testing is started from the components on the lowest level and goes up to the testing of integration of the next level components. Integration testing between top level components is the first point of the top-down approach. It goes to the lower level components testing afterwards till the lowest level is reached. On the contrary, integration on all levels occurs simultaneously when big bang approach is used.

Final controversy we want to mention, but not the last one in the software testing is **traditional testing vs. agile testing**. Agile school has completely different mission then other ones and discovers the role of software engineer in test, the great automation specialist and the main participant of test driven development.

Systematization of Testing Terms: Approach, Method, and Technique

The connection and clear border between testing approach, testing method, and testing technique are not defined in testing theory. For example, Beizer[6] defines test technique as a systematic method: “A **test strategy** or **test technique** is a systematic **method** used to select and/or generate tests to be included in a test suite.” In the same time, he uses test technique and test method as completely equal statements: “… here I present you with ready-made **equivalence class partitioning methods** (or **test techniques**) …” “**[T]est execution technique:** The **method** used to perform the actual test execution, either manual or automated”[8]. Other authors, such as Kaner et al. [7], Pressman[9], and Sommerville[10] have a mix of using words technique, method, approach, and strategy in regard to testing as well.

The attempts of making a distinction between approach, method, and technique were already performed by language teaching specialists in 1963, 12 years before the first theoretic foundation of testing by Goodenough & Gerhart was published. In 1963 Anthony provided “much needed coherence to the conception and representation of elements that constitute language teaching:”[11]

- **An approach** is “a set of correlative assumptions dealing with the nature of language and the nature of language teaching and learning. It describes the nature of the subject matter to be taught. It states a point of view, a philosophy, an article faith…”
- **A method** is “an overall plan for the orderly presentation of language material, no part of which contradicts, and all of which is based on the selected approach. An approach is axiomatic, a method is procedural”.
- **A technique** is described as “a particular trick, stratagem, or contrivance used to accomplish an immediate objective”.

"The arrangement is hierarchical. The organizational key is that techniques carry out a method which is consistent with an approach."

In 1982 Richards & Rogers[11] performed an attempt to enhance the framework developed by Anthony through dividing language teaching process into approach, design, and procedure. But, despite rather vague definition of terms *approach*, *method*, and *technique*, and not considering in any way of complex connections between them, exactly these terms are in favor of the most current teacher training manuals.[12]

We suggest systemizing testing approach, testing method, and testing technique in the same hierarchical way, using the experience and keeping in mind the mistakes of language teaching specialist. Schematic relation between terms mentioned above is shown on Figure 4.

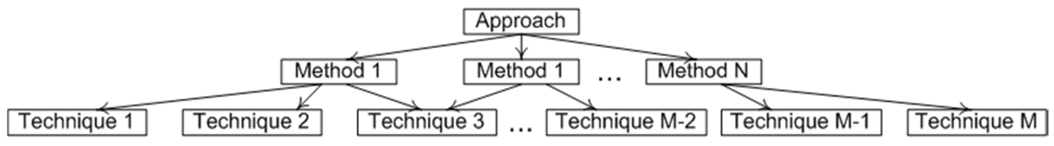

Figure 4. Relation between approach, method, and technique

Testing approach *“state a point of view, a philosophy, an article faith”* that a test engineer takes when designing test cases.

Testing method is *"an overall plan for the orderly presentation"* of testing techniques.

Testing techniques are united based on test case design formality (for black-box testing approach) or based on other common pronounced attributes (for white-box approach).

Testing technique is *"a particular trick, stratagem, or contrivance"* to design the test case.

The *"organizational key"* stays the same as suggested by Anthony – "*techniques carry out a method which is consistent with an approach"*.