Software Quality Assurance Research: Achievements, Challenges, Dreams

Sagar Deshmukh

DOI: 10.29322/IJSRP.11.05.2021.p11351
http://dx.doi.org/10.29322/IJSRP.11.05.2021.p11351

Abstract- Software engineering comprehends many disciplines dedicated to forestall and remedy malfunctions and to warrant adequate behavior. Testing, the topic of this paper, could be a widespread validation approach in trade, however it's still mostly unexpected, expensive, and erratically effective. Indeed, computer code testing could be a broad term encompassing a variety of activities on the event cycle and on the far side, geared toward completely different goals. Hence, computer code testing analysis faces a set of challenges. The same roadmap of the foremost relevant challenges to be self-addressed is here proposed. In it, the start line is deep-rooted by some important past achievements, whereas the destination consists of 4 known goals to that analysis ultimately tends, however that stay as unapproachable as dreams. The routes from the achievements to the dreams square measure sealed by the outstanding analysis challenges, that square measure mentioned within the paper at the side of attention-grabbing current work.

Keywords: Software Quality, Software Testing, Quality Assurance.

1. INTRODUCTION

Testing is a vital activity in package engineering. Within the simplest terms, it amounts to perceptive the execution of a software to validate whether or not it behaves as meant and determine potential malfunctions. Testing is wide utilized in trade for quality assurance: so, by directly scrutinizing the package in execution, it provides a sensible feedback of its behavior and intrinsically it remains the ineluctable complement to alternative analysis techniques.

Beyond the apparent straightforwardness of checking a sample of runs, however, testing embraces a range of activities, techniques and actors, and poses several advanced challenges. Indeed, with the complexness, generality and criticality of package growing incessantly, making certain that it be- haves consistent with the specified levels of quality and depend- ability becomes additional crucial, and progressively tough and pricey. Earlier studies calculable that testing will consume half, or maybe additional, of the event prices [1].

Correspondingly, novel analysis challenges arise, like as an example the way to conciliate model-based derivation of take a look at cases with trendy dynamically evolving systems, or the way to effectively choose and use runtime information collected from real usage when preparation. These fresh rising challenges attend augment long open issues, like the way to qualify and appraise the effectiveness of testing criteria, or the way to minimize the quantity of retesting when the package is changed.

In the years, the subject has attracted increasing interest from researchers, as testified by the various specialized events and workshops, further as by the growing proportion of testing papers in package engineering conferences; for in- stance at the twenty eighth International Conference on package Engineering (ICSE 2006) four out of the twelve sessions within the analysis track targeted on “Test and Analysis”.

This paper organizes the various outstanding analysis challenges for package testing into a homogenous roadmap. The known destinations area unit a group of 4 final and un- realizable goals referred to as “dreams”. Assuming to those dreams, researchers area unit addressing many challenges, that area unit here seen as attention-grabbing viable aspects of the larger downside. The ensuing image is planned to the soft- ware testing researcher’s community as a work-in-progress material to be tailored and swollen.

In Section two we have a tendency to discuss the multifaceted nature of package take a look at and determine a group of six queries underlying any test approach. In Section three we have a tendency to then introduce the structure of the planned roadmap. We have a tendency to summarize some additional mature analysis areas that represent the start line for our journey within the roadmap, in Section four. Then in Section five, that is that the main a part of the paper, we have a tendency to summary many outstanding analysis challenges and therefore the dreams to that they have an inclination. Transient last remarks in Section VI shut the paper.

SECTIONS:

7) Dreams
8) Conclusion
9) Acknowledgements

2. THE MANY FACES OF SOFTWARE TESTING

Software testing may be a broad term encompassing a good spectrum of various activities, from the testing of a tiny low piece of code by the developer (unit testing), to the customer validation of an oversized system (acceptance testing), to the observation at run-time of a network-centric service-oriented application. Within the numerous stages, the check cases may well be devised aiming at completely different objectives, like exposing deviations from user's necessities, or assessing the agreement to a typical specification, or evaluating lustiness to nerve-wracking load conditions or to malicious inputs, or measure given attributes, like performance or usability, or estimating the operational responsibility, and so on. Besides, the testing activity may well be carried on ac- cording to a controlled formal procedure, requiring rigorous designing and documentation, or rather informally and circumstantial (exploratory testing).

As a consequence of this sort of aims and scope, a multiplicity of meanings for the term "software testing" arises, that has generated several peculiar analysis challenges to prepare the latter into a unifying read, within the remainder of this section we tend to try a classification of issues common to the various meanings of software package testing the primary construct to capture would be what's the common de- proposer, if it exists, between all potential completely different testing "faces" we tend to propose that such a standard divisor is the terribly abstract read that, given a bit of software package (whichever its assortment, size and domain) testing invariably consists of observant a sample of executions, and giving a finding over them.

Starting from this terribly general read, we will then concretize completely different instances, by distinctive the precise aspects that may characterize the sample of observations:

WHY: why is it that we tend to create the observations? This question issues the check objective, e.g.: square measure we tend to searching for faults? Or, ought we to decide whether or not the merchandise is released? Or rather ought we to appraise the usability of the User Interface?

HOW: that sample will we observe, and the way will we opt for it? This can be the matter of check choice, which might be done circumstantial, at random, or in systematic method by applying some recursive or applied math technique. It's impressed a lot of analysis that is comprehensible not solely as a result of its intellectually engaging, however conjointly as a result of however the check cases square measure chosen -the check criterion- greatly influences check effectualness.

HOW MUCH: however huge of a sample? Twin to the question of however we will decide the sample observations (test selection), is that of what number of them we will take (test adequacy, or stopping rule). Coverage analysis or responsibility measures represent 2 "classical" approaches to answer such question.

WHAT: what's it that we tend to execute? Given the (possibly composite) system below check, we will observe its execution either taking it as an entire, or focusing solely on a district of it, which might be a lot of or less huge (unit check, component/subsystem check, integration test), a lot of or less defined: this side provides rise to the varied levels of testing, and to the required staging to allow check execution of a district of a bigger system.

WHERE: wherever will we perform the observation? Strictly associated with what we will execute, is that the question whether or not this can be worn out house, in a much simulated setting or within the target final context. This question assumes the best connectedness once it involves the testing of embedded systems.

WHEN: once is it within the product lifecycle that we tend to per- type the observations? The traditional argument is that the earliest, the foremost convenient, since the price of fault removal will increase because the lifecycle payoff. But, some observations, especially people who depend upon the encompassing context, cannot invariably be anticipated within the laboratory, and that we cannot keep it up any pregnant observation till the system is deployed and operating.

These queries give an awfully straightforward and intuitive characterization schema of software package testing activities that may facilitate in organizing the roadmap for future analysis challenges.

3. SOFTWARE TESTING RESEARCH ROADMAP

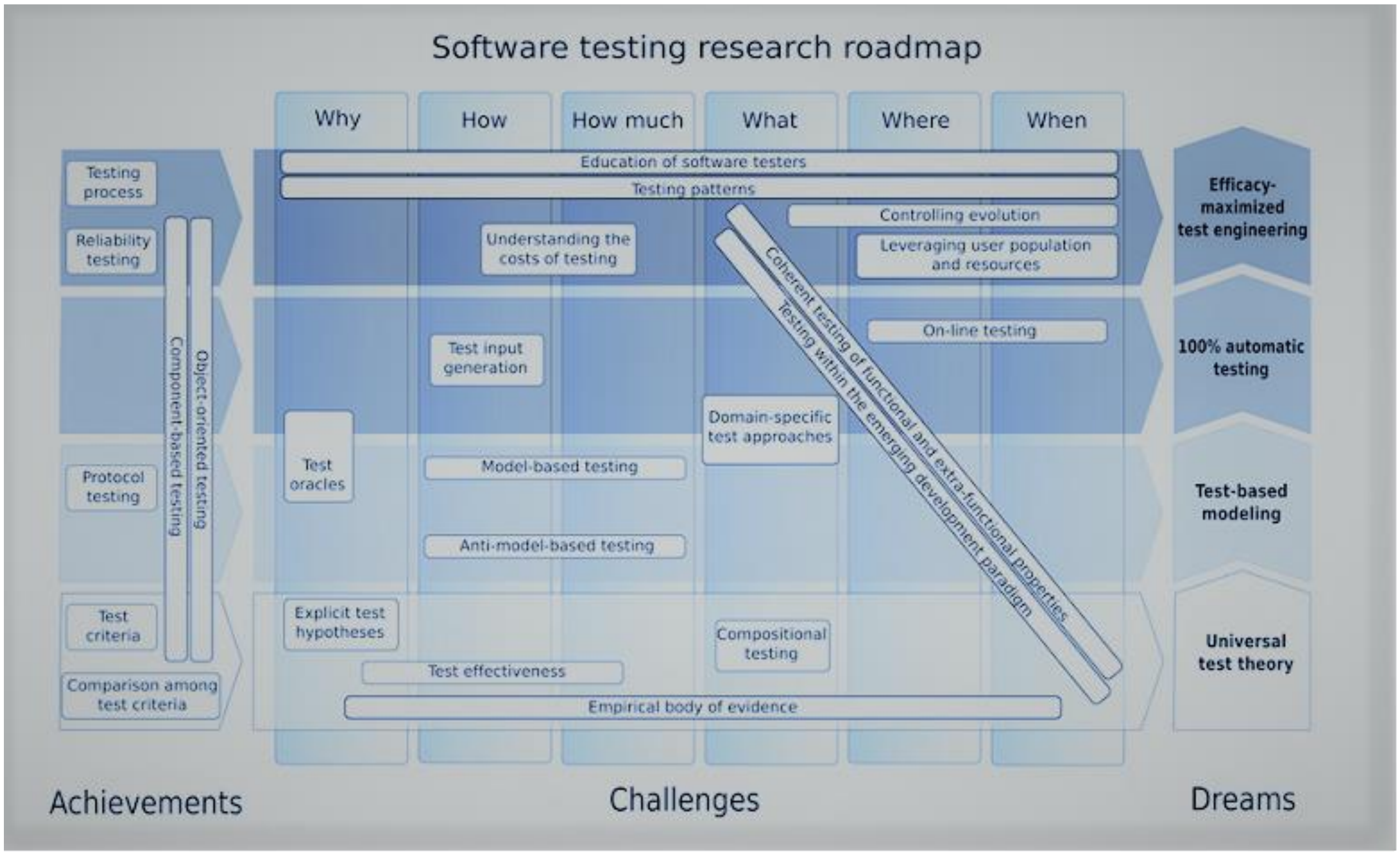
A roadmap provides directions to achieve a desired destination ranging from the “you area unit here” red dot. The soft- ware testing analysis roadmap is organized as follows:

The “you area unit here” red dot consists of the foremost notable achievements from past analysis (but note that a number of these efforts area unit still ongoing);

The desired destination is portrayed within the sort of a group of (four) dreams: we have a tendency to use this term to indicate that these area unit straight line goals at the top of 4 known routes for analysis progress. They’re out of reach by definition and their price specifically stays in acting because the poles of attraction for helpful, farsighted research;

In the middle area unit the challenges Janus-faced by current and future testing analysis, at a lot of or less mature stage, and with a lot of or less probabilities for fulfillment. These challenges represent the directions to be followed within the journey towards the dreams, and intrinsically they're the central, most vital a part of the roadmap.

The roadmap is illustrated in Figure one. In it, we've got located the rising and current analysis directions within the center, with a lot of mature topics -the achievements- on their A roadmap provides directions to achieve a desired destination ranging from the “you area unit here” red dot. The soft- ware testing analysis roadmap is organized as follows:



The “you area unit here” red dot consists of the foremost notable achievements from past analysis (but note that a number of these efforts area unit still ongoing);

The desired destination is portrayed within the sort of a group of (four) dreams: we have a tendency to use this term to indicate that these area unit straight line goals at the top of 4 known routes for analysis progress. They’re out of reach by definition and their price specifically stays in acting because the poles of attraction for helpful, farsighted research;

In the middle area unit the challenges Janus-faced by current and future testing analysis, at a lot of or less mature stage, and with a lot of or less probabilities for fulfillment. These challenges represent the directions to be followed within the journey towards the dreams, and intrinsically they're the central, most vital a part of the roadmap.

The roadmap is illustrated in Figure one. In it, we've got located the rising and current analysis directions within the center, with a lot of mature topics -the achievements- on their left, and therefore the final goals -the dreams- on their right. Four horizontal strips depict the known analysis routes toward the dreams, namely:

1. Universal take a look at theory
2. Test-based modeling
3. 100% automatic testing
4. Efficacy-maximized take a look at engineering.

The routes area unit bottom-up ordered according somehow to progressive utility: the speculation is at the idea of the adopted models that successively area unit required for automation that is instrumental to cost-efficient take a look at engineering. The challenges horizontally span over six vertical strips such as the WHY, HOW, HOW MUCH, WHAT, WHERE, and once queries characterizing software package testing faces (in no specific order). Software testing analysis challenges realize their place during this set up, vertically betting on the long run dream, or dreams, towards that they chiefly tend, and horizontally consistent with that question, or queries, of the introduced software package testing characterization they chiefly center on. In the remainder of this paper, we are going to discuss the elements (achievements, challenges, dreams) of this roadmap we are going to typically compare this roadmap with its 2000's predecessor by Harrold [2], that we are going to refer henceforward as FOSE2000.

4. ACHEIVMENTS

Before outlining the longer term routes of software package testing research, a pic is here tried of some topics that represent the body of data in software package testing (for a prepared, additional elaborated guide cf. [3]), or during which important analysis achievements are established within the roadmap of Figure one, these area unit diagrammatic on the left facet.

The origins of the literature on software package testing originate to the first 70's (although one will imagine that the terribly notion of testing was born at the same time with the primary experiences of programming): Hetzel dates the primary conference dedicated to program testing to 1972. Testing was conceived like AN art, and was exemplified because the "destructive" method of execution a program with the intent of finding errors, critical style that habitual the "constructive" party it's of those years Dijkstra's uppermost cited apothegm regarding software package testing, that it will solely show the presence of faults, however ne'er their absence.

The 80's saw the idea of testing to the standing of AN designed discipline, and a read amendment of its goal from simply error discovery to an additional comprehensive and positive read of bar. Testing is currently characterized as a broad and continuous activity throughout the event method, whose aim is that the activity and analysis of software package attributes and capabilities, and Bezier states: quite the act of testing, the act of coming up with tests is one in all the simplest bug preventers familiar.

Testing method. Indeed, a lot of analysis within the early years has matured into techniques and tools that facilitate build such "test-design thinking" additional systematic and in- company it at intervals the event method many take a look at method models are projected for industrial adoption, among that most likely the "V model" is that the hottest. All of its several variants share the excellence of a minimum of the Unit, Integration and System levels for testing.

More recently, the V model implication of a phased and formally documented take a look at method has been argued by some as being inefficient and unnecessarily officialdom, and in distinction additional agile processes are advocated. Concerning testing above all, a special model gaining attention is test-driven development (TDD), one in all the core extreme programming practices.

The institution of an appropriate method for testing was listed in FOSE2000 among the elemental analysis topics and so this remains a lively analysis these days.

Test criteria extraordinarily wealthy is that the set of take a look at criteria de- vised by past analysis to assist the systematic identification of take a look at cases historically these are distinguished between white-box (a.k.a. structural) and black-box (a.k.a. functional), reckoning on whether or not or not the ASCII text file is exploited in driving the testing. A additional refined classification may be ordered in step with the supply from that the take a look at cases area unit derived [3], and plenty of textbooks and survey articles exist that give comprehensive descriptions of existing criteria. Indeed, such a lot of criteria among that to settle on currently exist, that the $64000 challenge becomes the potential to form an even alternative, or rather to under- stand however they'll be most expeditiously combined. In recent years the best attention has been turned to model-based testing.

Comparison among take a look at criteria. In parallel with the investigation of criteria for take a look at choice and for take a look at adequacy, ton of analysis has self-addressed the analysis of the relative effectiveness of the assorted take a look at criteria, and especially of the factors that build one technique higher than another guilty finding. Past studies have enclosed many analytical comparisons between totally different techniques. These studies have permissible to determine a hierarchy of relative painstakingness between com- parable criteria, and to know the factors influencing the likelihood of finding faults, focusing additional in particular on comparison partition (i.e., systematic) against random testing. "Demonstrating effectiveness of testing techniques" was really known as an elementary analysis challenge in FOSE2000, and still these days this objective concerns additional analysis, whereby the stress is currently on empirical assessment.

Object-oriented testing. Indeed, at any given amount, the dominating paradigm of development has catalyzed testing analysis for adequate approaches, as we have a tendency to additional develop in Section five.5 within the 90's the main focus was on testing of Object-oriented (OO) software package. Rejected the parable that increased modularity and employ brought forward by OO programming might even forestall the requirement for testing, researchers shortly accomplished that not solely everything already learnt regarding software package testing generally conjointly applied to OO code, however conjointly OO development introduced new risks and difficulties, therefore increasing the requirement and quality of testing [4] above all, among the core mechanisms of OO development, encapsulation will facilitate hide bugs and makes take a look at harder; inheritance needs intensive retesting of heritable code; and polymorphism and dynamic binding necessitate new coverage models. Besides, applicable ways for effective progressive integration testing area unit needed to handle the complicated spectrum of potential static and dynamic dependencies between categories.

Component-based testing within the late 90's, component- based mostly (CB) development emerged because the final approach that may yield speedy software package development with fewer resources. Testing at intervals this paradigm introduced new challenges that we'd distinguish between technical and theoretical in a similar way. On the technical facet, elements should be generic enough for being deployed in several plat- forms and contexts, so the element user has to retest the element within the assembled system wherever it's deployed. However the crucial downside here is to face the dearth of knowledge for analysis and testing of outwardly developed elements. In fact, whereas element interfaces area unit delineated in step with specific element models, these don't give enough info for practical testing. so analysis has advocated that applicable information, or perhaps the take a look at cases themselves (as in intrinsically Testing), area unit prepacked at the side of the element for facilitating testing by the element user, and conjointly that the "contract" that the elements abide to ought to be created express, to permit for verification.

The testing of component-based systems was conjointly listed as an elementary challenge in FOSE2000.

What remains AN open evergreen downside is that the theoretical facet of CB testing: however will we have a tendency to infer attention-grabbing properties of AN assembled system, ranging from the results of testing the elements in isolation? The theoretical foundations of integrative testing still stay a significant analysis challenge destined to last, and that we discuss some directions for analysis in Section five.1.

Protocol testing. Protocols area unit the principles that govern the communication between the elements of a distributed system, and these have to be compelled to be exactly per order to facilitate ability. Protocol testing is geared toward verifying the conformity of protocol implementations against their specifications. The latter area unit free by customary organizations, or by consortia of firms. In sure cases, conjointly a typical conformity take a look at suite is free.

Pushed by the pressure of enabling communication, re- search in protocol testing has proceeded on a separate and, in a sense, privileged path with relation to software package testing. In fact, due to the existence of precise state- based mostly specifications of desired behavior, analysis might terribly early develop advanced formal strategies and tools for testing conformity to those established customary specifications [5].

Since these results were planned for a restricted well- outlined field of application, they are doing not promptly apply to general software package testing. However, constant original downside of making certain correct interaction between remote elements and services arises these days on a broader scale for any mod- white-tailed sea eagle software package; so software testing analysis might fruit- absolutely learn from protocol testing the habit of adopting standardized formal specifications, that is that the trend in mod- white-tailed sea eagle service-oriented applications. Vice versa, whereas early protocols were straightforward and simply tractable, these days the focus is shifting to higher levels of communication protocols, and therefore the quality plague additional typical of software package testing starts conjointly to become pressing here. Therefore, the abstract separation between protocol testing and general software package testing issues is more and more vanishing.

Reliability testing. Given the omnipresence of software package, its dependability, i.e., the likelihood of failure-free operation for a fixed amount of your time in an exceedingly fixed atmosphere, impacts these days any technological product. Dependability testing recognizes that we will ne'er discover the last failure, and hence, by victimization the operational profile to drive testing, it tries to eliminate those failures which might manifest themselves additional frequently: intuitively the tester mimics however the users can use the system software package dependability is sometimes inferred supported dependability models: totally different models ought to be used, reckoning on whether or not the detected faults area unit re- touched, during which case the dependability grows, or not, once dependability is barely certified.

Research in software package dependability has intersected analysis in software package testing in several fruitful ways in which. Models for soft- ware dependability are actively studied within the years 80's and 90's [6]. These models area unit currently mature and may be engineered into the take a look at method providing quantitative guidance for the way and the way a lot of to check. as an example, this was done by Musa in his Software-Reliability-Engineered Testing (SRET) approach ([58], Chapt.6), and is additionally advocated within the Cleanroom development method, that pursues the appliance of applied math take a look at approaches to yield certified dependability measures [7].

Unfortunately, the follow of dependability testing has not proceeded at constant speed of theoretical advances in soft- ware dependability, most likely as a result of its (perceived as) a complex and high-ticket activity, however conjointly for the inherent difficulty of characteristic the specified operational profile. However these days the demand for dependability and different dependability qualities is growing and therefore the requirement arises for monetary unit approaches to coherently take a look at practical and extra- practical behavior of contemporary software-intensive systems.

5. CHALLENGES

Challenge: leverage user population and resources

We have already mentioned the rising trend of continuous validation once readying, by means that of on-line testing approaches, once ancient off- line testing techniques become ineffective. Since software package intensive systems will behave terribly otherwise in varied environments and configurations, we want sensible ways that to proportion on-line testing to hide the broad spectrum of possible behaviors. One rising approach to deal with this challenge is to enhance in-house quality assurance activities by victimization information dynamically collected from the sphere. {this is| this is often| this will be} promising therein it can facilitate to reveal real usage spectra and expose real issues on that to focus the testing activities and on that testing is lacking as an example, by giving every user a special default configuration, the user base is leveraged to additional quickly expose configurations conflicts or issues, like in [8]. This high-level challenge involves many additional specific challenges, among which: - How will we have a tendency to collect runtime information from programs running within the field while not imposing an excessive amount of overhead? - How will we have a tendency to store and mine the collected (potentially huge) quantity of information thus to effectively extract relevant information? - How will we have a tendency to effectively use the collected information for augmenting and up in-house testing and maintenance activities? This challenge proposes that the users instantiate the wherever and once to scrutinize software package runs.

Challenge: Testing patterns

We have already mentioned underneath the dream of a helpful take a look at theory, that we want to grasp the relative effectiveness of take a look at techniques within the styles of faults they tar- get. To engineering the take a look at method, we want to gather evidences for such info to be able to notice the foremost effective pattern for testing a system this is often habitually done, once as an example practical testing supported the requirements is combined with measures of code coverage adequacy. Another revenant recommendation is to mix operational testing with specific verification of special case inputs. However, such practices ought to be protected by a scientific effort to extract and organize revenant and proved effective solutions to take a look at issues into a catalogue of test patterns, equally to what's currently a well-established theme for style approaches. Patterns supply well tested solutions to revenant problems, or, in alternative words, they create specific and document problem-solving experience. As testing is recognized as ex- pensive and effort-prone, creating specific that square measure successful procedures is very fascinating. A connected effort is Vegas and coauthors characterization schema of however take a look at techniques square measure elect. They surveyed the sort of information that practitioners use to decide on the testing techniques for a software package project and have produced a formalized list of the relevant parameters. How- ever, organizations that use the planned schema won't lose all needed info, thence additional recently they're conjointly work the sources of data, and how these sources square measure to be used [9]. Similar studies square measure required to formalize and document in practices for the other testing-related activity. In fact this challenge spans over all six queries.

Challenge: Understanding the prices of testing

Since testing doesn't ensue in abstract, however at intervals the concrete world, with its risks, and safety moreover as economic constraints, ultimately we would like to be able to link the testing method and techniques with their price. Each and each analysis article on software package testing starts from claiming that testing could be a terribly costly activity, however we have a tendency to lack up-to-date and reliable references; it's somehow appalling that also these days references to quantify the high price of testing cite textbooks geological dating back to over twenty years ago. This may true ensue to the sensibility of failure information, that square measure company confidential all the same, to usefully transfer analysis advances to practice we want to be able to quantify direct and indirect prices of software package testing techniques. Unfortunately, most analysis in software package testing takes a value-neutral position, as if each fault found is equally vital or has an equivalent price, however this is often in fact not true; we want ways that to include value into the testing method, to assist take a look at managers apply their judgement and choose the foremost applicable approaches. Jakob Boehme and colleagues have introduced the value-based software package engineering (VBSE) paradigm [10], within which quantitative frameworks to support software package managers selections square measure wanted that enhance the worth of delivered software package systems specifically, numerous aspects of software package quality assurance are investigated together with value-based and risk-based testing, e.g., [13]. VBSE issues chiefly the management of processes, as an example, w.r.t. testing, different styles of neutral utility functions square measure thought of two trade-off time of delivery vs. market price we might conjointly ought to be able to incorporate estimation functions of the cost/effectiveness magnitude relation of accessible take a look at techniques. The key question is: given a set testing budget, however ought to its utilized most effectively? This challenge clearly addresses chiefly the however and the way abundant of testing.

Challenge: Education of software package testers

Finally, for software package testing as for the other software package engineering activity, an important resource remains the human issue. on the far side the provision of advanced techniques and tools and of effective processes, the testers' talent, commitment associate degreed motivation will build an enormous distinction between a tin take a look at method or an ineffective one analysis on its facet ought to attempt for manufacturing designed effective solutions that square measure simply integrated into development and don't need deep technical experience however we have a tendency to conjointly ought to add parallel for empowering the human potential this is often done by each education and motivation. Testers ought to be educated to grasp the essential notions of testing and also the limitations and also the prospects offered by the obtainable techniques. Whereas it's analysis that may advance the state of the art, it's solely by awareness and adoption of these results by the next-coming generation of testers that we will conjointly advance the state of observe. Education should be continued, to stay the pace with the advances in testing technology. Education by itself poses many challenges, as mentioned in [11]. It is evident that education should cowl all characterizing aspects of testing.

6. DREAM

DREAM: 100% AUTOMATION TESTING

Far-reaching automation is one in all the ways in which to stay quality analysis and testing in line with the growing amount and quality of computer code computer code engineering analysis puts massive stress on automating the assembly of soft- ware, with a bulk of recent development tools generating ever larger and additional complicated quantities of code with less effort the opposite aspect of the coin is that the massive danger that the ways to assess the standard of the therefore made computer code, particularly testing ways, cannot keep the pace with such computer code construction ways.
A large a part of current testing analysis aims at improving the degree of getable automation, either by developing advanced techniques for generating the take a look at inputs (this challenge is dilated below), or, on the far side take a look at generation, by finding innovative support procedures to alter the testing method.
The dream would be a strong integrated take a look at environment that by itself, as a chunk of computer code is completed and deployed, will mechanically beware of possibly instrumenting it and generating or sick the required staging code (drivers, stubs, simulators), generating the foremost appropriate take a look at cases, corporal punishment them and eventually issuance a take a look at report. This idea, though chimera, has attracted followers, for example within the early bureau sponsored initiative for Perpetual take a look at (also mentioned in FOSE2000) and additional recently in Saff and Ernst' Continuous Testing approach [11], that specifically aims to run tests within the background on the developer's machine whereas they program.

7. CONCLUSION

We believe that software system testing could be richly articulated analysis discipline, and hope that this paper has provided a helpful summary of current and future challenges. Covering into one article all in progress and fore- seen analysis directions is impossible; we've got privileged width against depth, and also the contribution of this paper ought to be seen rather as an endeavor to depict a comprehensive and protractible roadmap, within which any current and future analysis challenge for software system testing will notice its place the image that emerges should be taken as a work- in-progress material that the community might want to adapt and expand.

8. ACKNOWLEDGEMENTS:

Summarizing the quite broad and active field of software system testing analysis has been a troublesome challenge. Whereas I stay the sole accountable of imprecisions or omissions, there are a unit ton of individuals whom I'm indebted. First, with the aim of being as comprehensive and unbiased as potential, I asked many colleagues to send American state each a press release of what they thought-about the upmost outstanding challenge moon-faced by soft- ware testing analysis, and a respect to relevant work (in- otherwise different authors' or their own paper) that this paper couldn't miss to cite.

9. REFERENCES

[1] B. Beizer. *Software Testing Techniques (2nd ed.)*. Van Nos- trand Reinhold Co., New York, NY, USA, 1990.

[2] M. J. Harrold. Testing: a roadmap. In A. Finkelstein, edi- tor, *The Future of Software Engineering*, pages 61–72. IEEE Computer Society, 2000. In conjunction with ICSE2000.

[3] A. Bertolino, E. Marchetti, and H. Muccini. Introducing a reasonably complete and coherent approach for model- based testing. *Electr. Notes Theor. Comput. Sci.*, 116:85–97, 2005.

[4] R. V. Binder. *Testing Object-Oriented Systems Models, Pat- terns, and Tools*. Addison Wesley Longman, Inc., Reading, MA, 2000.

[5] G. V. Bochmann and A. Petrenko. Protocol testing: review of methods and relevance for software testing. In *Proc. ACM/SIGSOFT Int. Symp. Software Testing and Analysis*, pages 109–124, 1994.

[6] M. Lyu (ed.). *Handbook of Software Reliability Engineer- ing*. McGraw-Hill, New York, and IEEE CS Press, Los Alamitos, 1996.

[7] J. Poore, H. Mills, and D. Mutchler. Planning and certifying software system reliability. *IEEE Software*, pages 87–99, Jan. 1993.

[8] C. Yilmaz, A. M. A. Porter, A. Krishna, D. Schmidt,A. Gokhale, and B. Natarajan. Preserving distributed sys- tems critical properties: a model-driven approach. *IEEE Software*, 21(6):32–40, 2004.

[9] C. Yilmaz, A. M. A. Porter, A. Krishna, D. Schmidt,A. Gokhale, and B. Natarajan. Preserving distributed systems critical properties: a model-driven approach. *IEEE Software*, 21(6):32–40, 2004.

[10] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Gru- enbacher, editors. *Value-Based Software Engineering*. Springer-Verlag, Heidelberg, Germany, 2006.

[11] D. Saff and M. Ernst. An experimental evaluation of contin- uous testing during development. In *Proc. ACM/SIGSOFT Int. Symp. on Software Testing and Analysis*, pages 76–85. ACM, July, 12-14 2004.

10. AUTHORS

Sagar Deshmukh

Email- deshmukhsagar92@gmail.com

New York, U.S.A