Holistic Research of Software Testing and Challenges

Kiran Jammalamadaka, Nikhat Parveen

Abstract— In early 2000, there were many popular magazines published software losses as headlines, records show that in US alone had $60 billion losses occurred due to software failures and nearly fifty percent of them can be prevented, if those systems had properly tested. These kinds of situations led to focus more on the importance of the software testing for the last two decades. Hence, the software testing growth was exponential. Software testing is a broad term consists of many types and phases of testing. However, the testing is predominantly ad hoc and expensive till today. To make it more systematic and cost-effective researchers are studying all the facets of software testing to increase the reliability and making it more affordable to suite all kinds of development models. In this review paper, we discussed and highlighted utmost facets of testing in detail including but not limited to the evolution of software testing to its goals and challenges we face today.

Keywords— Testing, Regression testing, Test Case selection, Test Prioritization techniques, fault isolating, Test Improvement Models Testing challenges.

INTRODUCTION

Most of the brain surgeries fail during the long and painful process of skull drilling, to avoid such long process and save lives of the patients, healthcare technology companies invented the skull drilling machines, which can reduce the drilling time from 120 minutes to 2.5 minutes with an improved precision by fifty times, that mean by using this machine one can avoid the manual error by 50 more times. This example describes how much important these machines to mankind. The machine designing, and development is one aspect of it and through testing of the machine is also as much as important as the development of these machines to commercialize, as a simple mistake can result in human life and similar other examples to quote are aircraft engines, space machines, oil drilling machines etc., a single malfunction can cause loss of billions of dollars

Computers have become an inseparable part of our normal life and mankind is very much dependent on computers for every activity in their daily routine. Many tasks have been automated at domestic and industrial places, computers are introduced to increase the productivity, and reach where human interactions are dangerous, and used to achieve higher precision in the task. Oxford dictionary defined computer as an electronic device or a machine that can store large amount of data and performs calculations, operations based on the instructions provided by the software program.

Revised Manuscript Received on April 12, 2019.

Kiran Jammalamadaka, Department of Computer science and Engineering, KL University, Vaddeswaram Vijayawada, AP, India. (E-mail: vskkiran@gmail.com)

Nikhat Parveen, Department of Computer science and Engineering, KL University, Vaddeswaram Vijayawada, AP India. (E-mail: nikhat0891@gmail.com)

A machine or a device is a combination of software, hardware and operating system to be precise, however in this context we discuss more about the software and its development, a life cycle consists of all stages of development from inception to delivery and maintenance of the software. In any field the consistency and predictability are the main ingredients to sustain the product or service for a long time [1], thus leads to systematic development of software is required.

In 1954, a first business computer was installed at General Electric Company, inexperienced developers followed trail and error method to accomplish the given tasks and they developed a list of to do to avoid the errors and increase the success probability [4]. Hence, this has become a recommended way for the next generations which is nothing but the methodology. Methodology describes about how to do, and Model describes what to do.

Software Development models

Over the period, there are many models evolved and each model has its pro and cons, Selection of development model should depend on the type of the project, Nayan B [5] categorized the models into three categories

1. Software, that serves as back end to other applications.
2. Software that serves the end users
3. Software that has User Interface which allows users to interact with the software.

And, Software development models also categorized into three categories,

Linear- this is a sequential model. Each stage will be completed before initiating the next stage.

Iterative- each stage will be visited iteratively,

A few models consist of both Combination of Linear and iterative.

Although, there are many software development models proposed by many researchers. In this paper we will discuss a few important models.

Waterfall Model

Waterfall model is a sequential execution of different stages, like how a water flows on the steps, Whole project has been divided into different phases like Requirements, Design, implementation, verification and Maintenance.

In 1956, Benington [6], proposed Waterfall model. In waterfall development model, development happens in stages. Next stage won't get initiated till the previous stage completed, this model has dominated its peer development models. However, the Bonington's waterfall model has a few pitfalls like cannot handle unforeseen issues with design or clarity with requirements, these pitfalls forced Royce [7] to modify the initial waterfall model an propose another waterfall model by adding a feedback loop to the original waterfall model, this loop helps to revisit the previous stage and modify if required and serve the purpose of the software development.

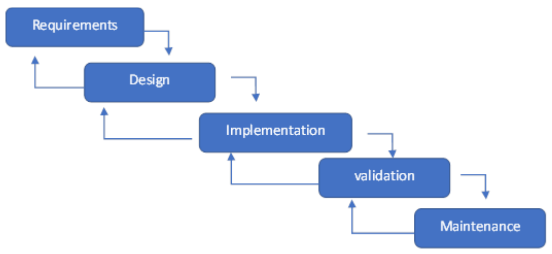

Fig. 1. Royce Waterfall model.

Royce initially suggested to visit the previous stage only. However, during the testing stage one can find an issue with the design itself, then Royce model would not allow this to accommodate, so Royce modified the model by adding a complex feedback loop where in revisit can happen from previous stage to any stage to get the things right.

Quality assurance has been built into waterfall model by splitting each stage into two one is verification, which ensures whether it fits the purpose by checking the requirements have been captured correctly or not and validation which consists of actual testing, whether the built product is working according to the specifications. Waterfall models are dominant for category 1 software's like compilers or operating systems [5].

V model

Another popular model is V model also knows as Vee model, it was first proposed by NASA in 1991[8], Vee model is also a sequential model, it can be treated as another variation of waterfall model, stages will be divided at a granular level and placed sequentially and then fold to middle to form a ‘ V ‘ shape.

Left part of the V shape represents the conversion of requirements into working software and right part represents the integration and verification of the system, Depth of V is proportional to the complexity of the system.

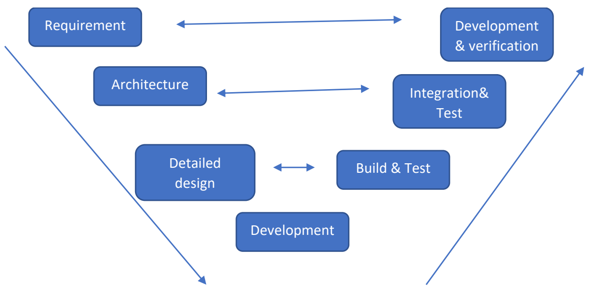

Fig. 2. V model

Quality assurance and verification procedures have been defined in the left leg for the corresponding right leg phases, this helps to verify requirements and designs in a SMART [9] way, (Specific, Measurable, Achievable, Realistic and Timebound).

Like waterfall model ‘V’ model is apt for category1 software, and to cater other category needs ‘V+’ and ‘V++’ models have been proposed [10], where user involves at every stage which helps the building the large systems in line with user or customer requirements.

Spiral Model

Once the requirements get frozen in waterfall model, it’s very much difficult to change the requirements, and changing requirements are not uncommon in software development process. To address the shortcomings of sequential models, Boehm proposed a model called “Spiral model” [11]. Spiral model is a combination of waterfall and prototype models and spiral model is a paradigm shift from requirements driven model to risk driven model. In spiral, development process is in iterations and each iteration consists of defining the objectives, risk management, Develop and test & Plan the next iteration. A prototype is built at every iteration and validated against requirements through testing. This ensures the prototype built in the spiral is meeting requirements and working as expected.

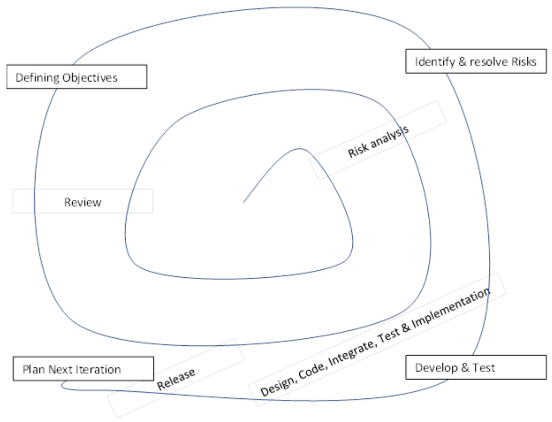

Fig. 3: Spiral Model.

Spiral model is also best fit for the Category 1 software like waterfall and vee models, however by modifying the Spiral model researches used to develop the category 2 and category 3 models as well.

In spiral model, the differentiator is the review which happens at end of each iteration looking and reviewing the commitments from the stakeholders and risk evaluation and including the coming plan for the next iterations. Quality assurance is part of the development at the quadrant three, however the iterations goes volume of testing increases in the form of regression testing as there will be no guarantee that delivered software was untouched. Every mistake in evaluating the risk causes the re testing due to incorrect implementation, though the amount will be less as it confines to the iteration only.

Rational Unified Process Model

Spiral is a risk driven and Waterfall is a specification driven models, and Rational Unified Process (RUP) Model is a Model based and use case drive [12].It was proposed by Rational a division of IBM.

RUP divides the whole process into 4 major parts

1. Inception, idea of the project will be discussed and assessed whether it is worth proceeding or not, and if it is worth proceeding, then determine what resources are needed.
2. Elaboration, at this stage required resources are further evaluated and software architecture and costing will be carried out.
3. Construction, the project will be developed and tested.
4. Transition, here the software project will be delivered to the end users and feedback will be incorporated if possible.

Each phase must go through six disciplines Business Modeling, Requirements, Analysis & design, Implementation, Test and Deployment. And these disciplines are supported by three more disciplines Configuration management, project management and Environment.

PHASES	INCEPTION	ELOBRATION	CONSTRUCTION	TRANSITION
DISCPLINES				Business Modeling
				Requirements
				Analysis & Design
				Implementation
				Test
				Deployment
SUPPORTED DISPLINES				Configuration management
				Project Management
				Environment

Table 1: Rational Unified Process

RUP is more suitable for category 2 and 3, by managing risks and by better management it could be applied for category 1 applications.

Rapid Application Development

RAD model is an iterative prototyping model proposed by martin, and it focuses more on UI intense applications, where Users or customers got to see the application in quick time, which enables customer to provide their feedback. In RAD components are treated as an individual project and integrated into a working prototype, each development activity is a time boxed activity. RAD encourages collaboration and makes stakeholders to participates in all the activities like prototyping, test case generation etc. Decision will be a decentralized process where all the stake holders has a say in the final decision.

There are 5 phases in Rapid Application Development model [13].

1. Business Modeling, Information flow will be identified between business components.
2. Data Modeling, data objects will be identified using the information obtained during the business modelling
3. Process Modeling, Data modeling objects are converted to achieve a designated business purpose
4. Application generation, converting the process to actual code by using tools
5. Testing and Turnover, All the components and interfaces are tested.

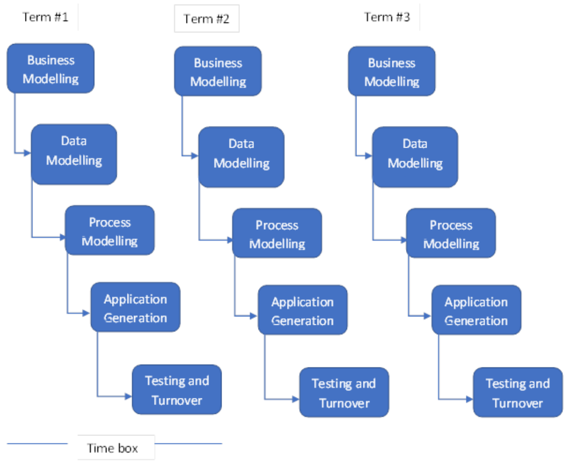

Fig. 4: RAD model.

Agile Development Methodology

In early 90s industry started addressing the issues to meet the customer requirements, and identified the need of lightweight development models [14]. A group of individuals including Kent Beck, Dave Thomas, Jeff Sutherland, Ken Schwaber, Jim Highsmith, Ward Cunningham, Alistair Cockburn, and Robert C. Martin. published the Manifesto for Agile Software Development [15]

Agile development is a method in which requirements and solutions evolve, through collaborative effort, Agile advocates adaptive model and evolutionary development model. It also encourages quick alignment to the change and continual improvement and incremental delivery.

Agile has become popular by the buzz words like Agile manifest and Agile Development principles

Agile Manifesto [16]:

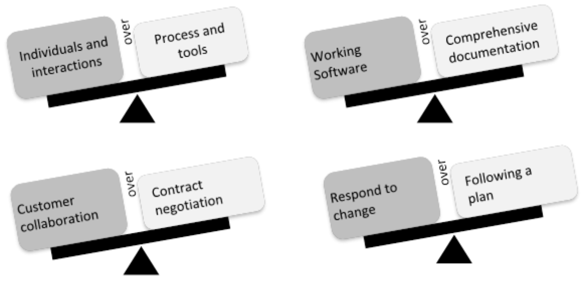

Fig. 5: Agile Manifesto.

- Individuals and Interactions over processes and tools.
- Working Software over comprehensive documentation.
- Customer Collaboration over contract negotiation.
- Responding to Change over following a plan.

Agile manifesto suggests to value more for the left side over the right side, However, Agile did not suggest ignoring the left side and based on the need and value add, one has to balance between these two sides.

Agile software development principles

12 principles have been proposed by Agile Manifest for Agile software development

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile software development methods

Many agile software development methods have been proposed to serve various purposes of the developments like Extreme Programming, and few focuses on the process, work flow like Scrum. We will be discussing Extreme programming and SCRUM methodologies.

Extreme Programming [17]

Changing requirements from the customers made to come up with a development methodology called extreme programming it is called XP. Extreme programming is intended to provide high quality software and high productivity, this agile development methodology expects the change and delivers software in frequent and short cycles releases to the stake holders.

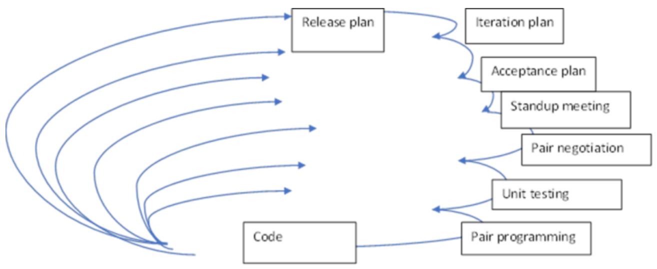

Fig. 6: XP Planning -feedback loops

XP includes only the code what a feature demands, not in future perspective, encourages to program in pairs and through code review and a flat management structure.

XP recognized 4 values, those are communication, Simplicity, Feedback and Courage.

Scrum

Scrum methodology is the most popular methodology in Agile software development methodologies. Scrum is an implementation methodology of Agile software development model.Team starts with a prioritized product backlog which are also known as requirements, the team takes a few high priority backlog items and implements them in an agreed time box which is called a sprint. At the end the working software will be delivered to the stake holders and next sprint begins with next prioritized back log items.

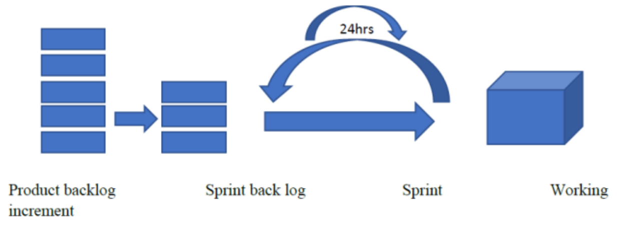

Fig. 7: Scrum methodology.

Scrum Artifacts:

Provide important information about the product development to the team and stakeholders, the below artifacts are defined in the Scrums [18]

- Product Vision: Defines the long-term goal of the project/product
- Sprint Goal: Defines the Sprint Objective
- Product Backlog: Defines the list of things needed for the product.
- Sprint Backlog: A subset of Product back logs, targeted for this sprint.
- Definition of Done: Defines the when a item is completed, by meeting what all the criteria
- Burn-Down Chart: A chart that gives overview of the product development over the time.

Scrum ceremonies:

There are 4 ceremonies suggested for scrum

Sprint Planning: Is a planning session, to make sure team members are sure of what they are supposed to deliver.

Daily Scrum: A time to team to get-together, to discuss the progress and hurdles for their task

Sprint Review: At the end of sprint, team cross check what team is supposed to deliver vs what the team delivered.

Sprint Retrospective: A meeting place where team discusses the improvement areas where team can improve and starts implementing from the next sprints.

Software testing

Everybody praises the thought, design even the developers who coded but not the engineers who tested and ensured the functionality is meeting the customer expectations or requirements of any product, application or a complex device. However, if any of the feature does not meet the requirement or expectations, immediately only question raises in one’s mind is “Who tested it?”. This shows the impact of the testing on the customer experience and importance of the testing in contributing to meet the customer expectations.

As we continue with the example of skull cutting device, which was mentioned in the beginning of this paper, functionality of the device is to cut through the hard part of the skull and before cutting any further brain, it should automatically stop, it’s no way tolerable to allow further 1/10 th. of the centimeter or even millimeter, such mission critical devices require more testing than development. Similarly, device space agencies use in their missions, and health care divisions and a lot of more critical applications, and financial institutions heavily depend on the software applications with high availability and reliability, such applications demand more testing of the devices/ applications. These category applications need to be almost ‘bug’ free, as a single minor mistake can take lives or loss of Assets or billions of dollars’ money loss.

Testing is an underlyingly important job, this has been realized very early in the game and researchers and industrialists started working for its betterment.

The International Software Testing Qualifications Board (ISTQB) defined testing as the below

"The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects."

And IEEE Std 829-1998 defined the below

"The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items."

James A. Whittaker *defined [19] as follows*

“The process of executing a software system to determine whether it matches its specification and executes in its intended environment.”

Software testing has become an integral part of software engineering, from an ad hoc, self-testing by programmers to a separate discipline, though it is matured but not enough to meet the development needs like low cost and high quality.

Evolution of Software testing

Software testing has evolved towards achieving objectives like, low cost and high quality most of the challenges fall under any of these categories.

In 1822, when Charles Babbage invented his first mechanical computer called ‘difference engine’ to calculate values of polynomial functions [22], and it is believed that he tested his computer to validate whether it is calculating the values correctly, from then testing has evolved and the evolution is still continuing.

In 1957, Charles L. Baker [20], described the difference between testing and debugging, since then testing has many influences over the period, debugging is first mile stone in the evolution of software testing, before that lets know some history about the bug and debugging terms.1945, US navy office Grace Hopper found a moth in the computer, which was causing a computer to malfunction by shortened one of the computer circuit, and she reported the process of taking out the moth from the computer as debugging, Grace made it popular, though there are some evidence which says she is not the first one[25] to use the term debug or debugging.

Dave Galperin and William C. Hetzel [23], classified the influences on the software testing, their findings are represented in the below table.

Until 1956	Debugging oriented period, till this period debugging is not clearly defined
1957–1978	Demonstration oriented period, during this period the focus is to ensure software satisfies the requirements, through demonstrations or displays.
1979–1982	*Destruction oriented period, focus is on the fault detection*
1983–1987	Evaluation oriented period, focus is on the evaluation of software during the software life-cycle
Since 1988	Prevention oriented period, Focus is more on the prevention rather detecting
later	software testing embedded into every stage of the software development life cycle stages.

Table 2: Software test evolution

At later stages, other poplar testing methods like Agile testing, Exploratory testing and risk-based testing, Test Driven Development, Behavior -Driven development etc., have been emerged.

Testing was made an integral part of the software development life cycle to measure and improve the quality of the software being tested [30].

Principles of software testing

There are 7 fundamental principles, defined to help testers to their job in a better way, testing needs to be done keeping these principles in mind.

- Testing only shows existence of defects not the absence of defects
- It is impossible to test all the possible combinations of data and scenarios.
- Early involvement of testing team yields good results, by ensuring the all the requirements were met at every stage.
- Pesticide paradox, executing same tests on the same code cannot find new defects.
- Defect clustering, if a defect gets uncovered in one module, there is a high probability that there can be more defects in the same areas/module, and there is a belief that 80 percent of the defects comes from 20 percent of the modules.
- Testing should be driven by context, in case of mission critical as explained in this paper it takes more time in testing than to development of the product.
- If the system is unusable by the users, then no point in testing and finding the applications.

Software testing Life cycle

Software test life cycle involves a sequence of activities to perform software testing

Activities can be defined as

Analysis, Planning and preparation, execution and closure [31].

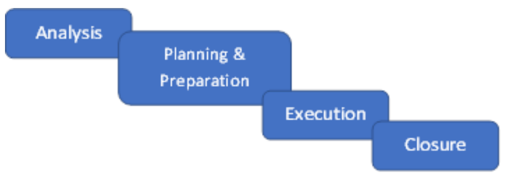

Fig 8: Software Test life Cycle

Test Analysis Phase: The requirements collection will be done from a test perspective, testable requirements also will be determined, requirements include functional, non-functional.

Planning Phase: Test plan will be created to outline the scope and out of the scope testing, roles and responsibilities, features to be tested. And any discrepancies will be communicated to the customer or the stake holders [32], In this phase, Test case development, test data & environment preparation also will be done, and for each test case expected result will be defined and the guidelines to raise a defect in the defect tracking tool.

Execution Phase: Test plan prepared in the previous phase will act as an input for the execution phase, as the test team will be executing the described test cases as per the definition in the test plan and observations will be recorded, and in case of any un matching expected result a defect will be logged in the specified defect tracking system.

Closure: A critical review and reporting will happen at this phase, all the test plan execution results will be verified, and a decision will be made by the stake holders on the release of the product.

A detailed view of the various phases of the test life cycle are below

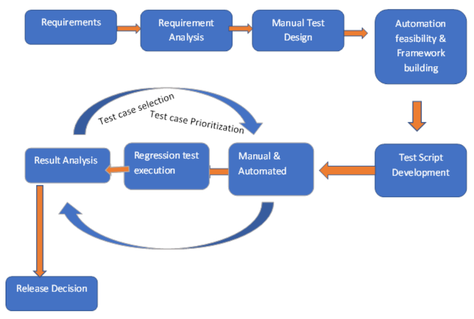

Fig 9: Detailed Test Life Cycle

Software testing Types and levels

Software testing is a process of uncovering the defects from various phases of the development process, testing can be an intermediate like check point to reach the quality product, to achieve a good quality product. Various levels of testing levels have been introduced at every stage of the development process.

Broadly the Test levels have been classified into

- Unit Level testing
- Integration level testing
- System Testing
- Acceptance testing.

Each level should represent a level of testing or as a mile stone of testing, competition of each level is important for the next level to start. Hence these levels of testing will follow a structured hierarchy of testing.

There are various software testing techniques according to the research like black box, white box, grey box [33], regression [34] reliability usability performance, security to mention a few.

The below picture depicts the levels of testing and various types of testing for a better understanding,

Note: not representing all the available test techniques, mentioned a few for understanding.

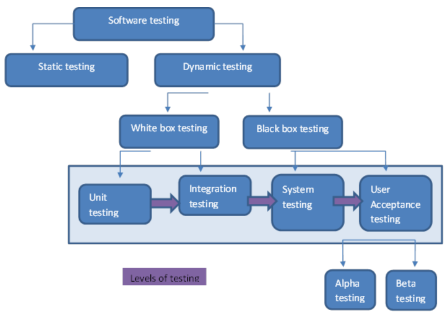

Fig 10: Levels of testing

In the below table, different types of testing techniques have been summarized [35].

Testing Level	Responsibility	Testing Techniques
Unit or Component level testing. Smallest piece of testable software	Typically, done by a programmer	• Buddy Testing • Automated Unit testing
Integration Level testing. Combining different units of testing and validating for functional requirements	Generally, done by the programmer, or in some cases a test engineer.	• Bottom up testing • Top Down testing • Integration testing • Big bang testing
System Level Testing Major level of testing, not only functional but also other aspect of quality like performance and security	Predominantly by Test engineers typically involves Performance test groups/teams, security groups/teams and Functional test groups or teams	• System Testing ▪ Structural • Stress Testing • Recovery Testing • Operation Testing • Compliance Testing • Security Testing ▪ Functional • Requirement Testing • Regression Testing • Control Testing • Parallel Testing
Acceptance level testing **Software will be tested by the external members to the organization**	External potential users or existing customers	User Acceptance testing

Table 3: Levels of testing and related testing techniques.

Brian Marick, Lisa Crispin and Janet Gregory introduced agile test quadrants [39].By placing the test techniques into a quadrant.

Business facing			
Supporting the team	**Automated & Manual** • Functional Tests • Prototype	**Exploratory Testing** • Scenario Based Testing • UAT	**Critique to the Product**
	Automated • Unit Test • Component tests	**Tools** • Performance • Load • Security	
Technology Facing			

Table 4: Agile Test Quadrants

These Testing Quadrants acts as a check list to guide the team to cover all the types of the testing during the product development.

Automated Software testing

Software quality has gained more focus these days due to the increased dependency on the software systems by many organizations accompanied with increased complexity of the software. The software 'escapes' that impact the customer can be made minimal only by improving the quality of the systems and there by the losses are also taken care from getting bigger [25]. Improved quality of the systems cannot be achieved just by increasing the no. of testing resources or by increasing the effort to test the system, which might make the system worthless by the time it gets released to the market. This led to the era of automation testing. Automation testing ways or methods designed for different needs are called automation frameworks. Using these automation frameworks, the software test engineers can improve the quality and at the same time they can reduce the time spent for repeated tasks on different software environments. To achieve these goals using test automation framework, the development of the test automation should also follow the processes and have a well-organized approach [26]. The different automation testing frameworks are described in detail below.

Automated software testing frameworks

Test Driven Development or Code Driven Testing

Test driven development, though it is present for 10 years [29] it has gained prominence in the recent years due to the Xtreme programming techniques of agile methodology. Test Driven Development (TDD) aims for development of the tests for the code or unit or object that is yet to be developed. This increases the efficiency of design and the code coverage as it creates less possibilities of developing the objects which are thought to be required contrary to developing all the objects which might not be used and writing the unit tests to test those functions/ units. So, in TDD framework the tests are written first and then to make them pass the implementation is done, then again tests are written followed by implementation. This helps in even creating large regression suites and helps in understanding the impact of changing a code corresponding to a function.

Behavior-Driven Development

The BDD is an extension of TDD, in a way the test scenarios are written from the acceptance criteria, instead of unit tests, and then the respective methods to full fill the test scenarios are developed. The test scenarios mainly target the business cases. This will increase the scope of system behavioral testing which will also adds to increase the confidence for the stakeholders on the system. Apart from the improved quality this framework also helps in fast delivery in this agile world. Hence BDD is considered the apt framework in agile development which helps in better communication, understanding of requirements, better quality and no slippages in delivery among the entire team.

Model-based testing

In model-based testing a model of the system is developed which is used for test case generation and execution. A model can be referred to as actions performed by the user on the system and the expected state of the system to be achieved on completing all the user actions. The model generated can be used as specifications and for deriving the test cases. Model based testing mainly concentrates on specific feature or behavior of the system and the respective state of the system. The focused approach to a specific feature or behavior of the system increases the coverage.

Modularity-Driven Testing

Modular driven test cases can be described as simple linear test cases confining to a single module. Multiple modular test scripts can be combined to form larger test scripts. As the modular test scripts have implementations starting at a very low level, any changes to the application under test will have lesser impact when compared to other frameworks. Though the modular test scripts are easy to generate, having the test data inside the code makes them vulnerable to code changes when there is a demand to test the system with different test data. This drawback has lead to the data driven testing framework.

Library Architecture Testing

Library architecture testing is very much like the modularity testing framework in a way instead of scripts the system under test is decomposed by its modules, sections or functions. The libraries or the utilities are built for these modules, sections or functions which can be reused in the test scripts. Though the maintenance and code reusability is improved, the software test engineers may experience the challenges of test data getting hard coded in the test script and the software test engineers should have expertise in coding and debugging for developing the test scripts using the utilities.

Data Driven Testing

A simple or a linear test script can have the test data hard coded inside the script. However, if any test engineer wants to verify the same test script with a different set of test data either he/she needs to update the existing test script or create a whole new test script with the same code copied and differing in test data. In either scenario if there is a change in system under test the testing or the maintenance of the test scripts is impacted a lot [27]. Hence then the automation approach for reading the test data from external sources came into existence. This kind of reading test data from external sources like database, excel files, CSV files etc., and executing the same script with different sets of test data is called Data Driven testing.

For instance, let’s consider a test case, which validates credentials of the user and navigates to the predefined page, same code logic needs to be tested with various usernames ,password, and expected page title in linear test approach, automation test engineer needs to replicate the same code logic by changing the user name , password and expected page title, data driven suggests to separate data from the business logic.

Username	Password	Expected: Page title
Admin	admin@123	Admin Page
User1	User1@123	User page
User1	asd@123	Invalid credentials

Table 5: User name and Password data example

However, here the test scripts and the test data are coupled together so tightly that in any case if there needs any new addition in the test data, the test script must be updated accordingly, like in the above example if we need to add another column then the whole business logic needs to be updated, which could be a major change.

Keyword Driven Testing

Keyword driven testing framework is the solution that has been proposed by Fewster and Graham (1999) and Kaner et al. (2001) [28] not only separates the test data but also the test actions that must be performed on the test data are taken out of code and are handled through external files. The test actions being defined are called Keywords. These keywords can be used by the software test engineers to develop the test cases. Hence to summarize the keyword driven framework can be treated as an extension of data driven testing framework [28]

The below table explains how a test script reads the data and action from data source

Here, keywords are associated with operation, which are to be performed on the control with the data specified in the value column.

The below test cases are trying to login a user by entering 'username' as user name and '@1234' as password and trying to click the Login button. The same can be extended to the further steps to complete the objective of the test case.

Parent Object	Control	Control property	Action	Value
Browser: page	Web Edit	Username	Set	username
Browser: page	Web Edit	Password	set	@1234
Browser: page	Web Button	Login	Click	

Table 6: keyword driven framework example

Hybrid Testing

As the name suggests it is the combination of two or more frameworks listed above. The juvenile stage of any automation framework can be called as hybrid framework which in later stages might get developed into acceptable new framework.

Software testing challenges

Testing has become an important activity to increase confidence and realistic feedback of the system, as we mentioned earlier, Agile testing is slightly different from traditional testing. In traditional testing models like waterfall, Vee models testing is a separate phase, and in Agile methodologies, testing is an integral part of development, there is no such separate phase or time has been allocated to testing in Agile methodologies, however, some product teams have an hardening sprint to ensure the overall quality of the product.

As testing evolving, testing challenges are also evolving over the time, as per the recent survey 75% of the organizations adopted agile to accelerate software delivery and 46% to enhance software quality.

The challenges are now different from earlier, we have recently conducted a survey across the globe and found a few real time challenges faced by the real time test engineers from the various small, medium and large corporates [36]. And a most of the challenges were categorized in "time consuming" tasks bucket and a few related to rapid development of their automation and a few challenges comes under the trust on their automation bucket.

Either in traditional sequential models or iterative models like agile, cycle time reduction has become a necessary task and teams with quality background like six sigma, lean specialists have been deployed to optimize the development and test process, in this paper we confine our scope to test activities.

If we observe the activities irrespective of the development model, as tasks won't be changing as per the model, and if a task is important and adding value to the project, then irrespective of the development model the task is necessary.

We can list out the tasks in two separate categories.

1. Defect prevention focused tasks
   Pre-execution focus on the prevention of the quality pitfalls, in this category the focus is to strengthen the test related process, like requirement analysis from the test perspective, and writing the test cases in an optimized way, and selecting the test cases, prioritizing the test cases, preparing test data for test automation, and building the roust test frameworks and building the test scripts.
   This "prevention focus" plays a major role in over all cost of the project, as it is always costly to fix an implemented defect in the system, rather stopping the defect entering into the system is more reasonable and time and cost saving, hence the above mentioned tasks focuses on the keeping checks for the defects and tries to stop the defects to enter into the system.
2. Defect Uncovering focused tasks
   These activities start once the implementation is done and system is ready for test or use, these tasks involve, executing the test cases, analyzing the failures to determine the application failures or test script failures, and defect slippage analysis by conducting Root cause analysis and other processes for betterment.
   Both the tasks are important, as prevention tasks cannot guarantee until team proves it by executing the scenarios, and these are in a cycle, each miss or escape acts as a input to the other, focused towards continues improvements of each.

Higher Quality in Less Time.

Often, we hear that testing the major "time consuming". Testing activities can be done in parallel at least verifications. However, regression testing can be done only after development activity is completed, and often developers intrude into regression test time and eventually regression testing time shrinks. For the benefit of the product and organization and organization credibility, testers need to act smart to achieve, higher quality in less time.

There are a few options available for test teams to minimize the effort required, a few to mention in this paper.

Test case selection and Test case prioritization is once aspect one can look at, and industry and researchers are been looking into this already, we will discuss in detail later in this paper.

Another option, teams are looking into is leveraging test automation to optimize the testing activity, however automation comes with a lot of baggage called maintenance. And further studies have been going into optimize the failure analysis and debugging process. [37] [38].

Test case selection in Regression Testing

Not only the automation address time optimization, however there are other areas like, test case reduction.

Many researchers proposed many models to reduce the test suite, and categorized as below

In the below table, various concepts have been proposed to select and prioritize the test cases [40]

Test case reduction techniques	Description	Pros	Cons
Requirement Based	Minimum set of test cases will be identified to test all the requirements.	Good percentage of reduction of test cases	Need more memory to represent requirements and Time consuming
Genetic Algorithms	Computational intelligence based, history based initial population will be built and calculates the fitness value and coverage cost, these will continue in a loop till a minimum set has been found. [41]	Test cases will be reduced and execution time as well	Fault detection capability is not optimized
Clustering	A combination of Clustering and datamining technique has been used to reduce the redundant test cases [42], A selection based on the coverage has been proposed [43]	Smaller sets of test cases	Not enough fault detection capability
Fuzzy Logic	Fuzzy logic is objective based, and function as a human, It is said to be safe for reducing the test suite size[44][49]	Relatively reliable technique and reduces the testcases set and execution time and cost effective	Not enough experimental studies.
Coverage Based	Coverage aspect is important in regression testing, it is to ensure majority of the paths have been executed of the given program. [45]	Reduction of test set is good and reduces the time.	For large systems it is ineffective, it requires more time to identify the paths
Program Slicing	Program slicing has been introduced by [48], and Program slicing eliminates the non-impacted program lines and focuses more on the impacted code lines on the output, A minimal set of test cases are required for such impactful code lines.	Decreases the number of test cases and execution time	Not enough fault detection capability on large amount of data.

Greedy Algorithm	A well know algorithm to reduce the test suite based on the relation between requirements and Test cases, this activity will be repeated till we cover all the requirements [47],	Good amount of reduction	Random selection of test cases in tie.
Hybrid Algorithm	Combination of one or more algorithms to reduce the test suite reduction such as genetic algorithms with bee colony [46]	Good reduction	Complexity is high

Table 7: Test case reduction Techniques

Test case Prioritization in Regression testing

Test case prioritization is about re ordering the test cases to yield the maximum quality with less effort,
In this technique a set of test cases will be ordered, and executed., this will be helpful to uncover the defects the early and gives maximum output when constraints like budget and time. To explain it further, in case testing must be halted or stopped due to cost or time, testing done till that moment should yield the maximum result.
Major test case prioritization techniques have been classified [50] in the below table.

Test case Prioritization Technique	Description	Remarks
Coverage Based Technique	Coverage based technique based on the code coverage achieved by the test case or test suite, the more the coverage the more probability of finding the defects, Rothaermel [51] proposed statement, branch total and additional. Erlbaum [52] proposed version specific prioritization, 8 techniques were proposed Bryce and Memon [54] proposed 5 new testing techniques interaction coverage-based prioritization by length (longest to shortest and shortest to longest) of the tests. Srivastava and Thigarajan [54] proposed a binary based prioritization technique. Bellie [55] proposed a graph model-based prioritization using fuzzy clustering Adaptive random test case prioritization technique was proposed by Jiang [56], and categorized as maximum, maxavg and maxmax	Faster Fault detection capability was the criteria in comparing the different types of techniques, APFD metric was used to calculate the efficiency and found total coverage is better than additional coverage prioritization technique
Modification Based Technique	This technique is to prioritize the test cases based on the modified code lines of a program, Korel [57] proposed a system model bases selective test prioritization using extended finite state machines. Korel [58] propose a few heuristic based techniques	Fault detection potential was the measure to the asses the efficiency of the models and techniques. Extended finite state machines were expensive.
Fault Based Technique	Rothaermel [61] proposed a fault-based prioritization technique, As per this the ability of the fault depends up on the faulty statement which can contribute to the test case failure. Based on the fault exposing capability of the test case, two methods have been proposed, Fault exposing potential and Additional fault exposing potential	Fault detection potential was the metric and, study showed Additional fault exposing potential is superior to fault exposing potential.

Requirement Based Technique	The objective of requirement bases technique is to uncover the serious defects as soon as possible to improve the quality for customer or stake holders. Srikanth [39] proposed a an approach based on the four factors, customer assigned priority of the requirements, developer assigned complexity in implementation, fault proneness of the requirement and requirement un predictability	This technique focused customer highest priority requirement and also improved fault detection rate.
History Based Technique	This technique based on the historical execution, a test cases will be assigned a value based on the past history, like number of times the test case uncovered the defects. Kim and Porter [59] proposed this technique to increase the effectiveness of the test regression suite.	Context may not be the same for every regression run, selecting a test case based on the fault detection capability from the past execution history of the test case may not give good confidence on the test case prioritization

Table 8: Test case Prioritization

Automated debugging

As mentioned above the automation come with a lot of baggage called maintenance, which typically include test script updating, and analysis of the automation failures etc. Automated debugging helps in analyzing the failures is one area which can be improved in terms of time hence cost.

Automated debugging has been an interesting topic for many researchers and many contributions went into this area.

This process starts when team receives the defect report, each list of failures needs to be analyzed to determine application failure or test script or test case failure or any other environmental failure etc., this is obviously a time consuming, the regular you do the more time you spend.

The below table list down a few popular techniques proposed by many researchers in automated debugging.

Technique name	Concept
Delta debugging [62]	Programmers often face a situation, program was running till some time and now it is failing, thus delta debugging comes into picture. Delta debugging isolating the failure inducing code lines, by comparing the differences between successful run and failure run of a program, including code changes occurred, input provided etc.
Spectra based fault localization [63]	A rank based approach has been presented in this technique to guide the developer to the likely defect function or location, it will be determined by the number of successful and un successful runs of a test case, more number of failures hits of high probability of the location previously caused the failure, however, in spite of good results, there is a negative feedback on this approach is often developers gets diverted with this approach[67].
Program slicing [65]	Program slicing technique is based on decomposition of a program and eliminates the non-relevant part of the program by a slicing criterion to isolate the failure inducing changes. Sliced program forms an executable, However, there could be erroneous slices can be formed and the chances of increasing the debug time rather reduce, often slice formed may be of a big in size
Static and Dynamic slicing [66]	In a program P, and slice S which consists all statements from the P, statement x and variable v. In this context, Static slicing includes the statements which can affect the value of the variable v at the statement x, these are computed by backtracking dependencies in between statements In dynamic slicing, slice S contains only those statements during the execution which can affect the value of v

Statistical debugging [64]	This technique is about leveraging statistical models to guide the developer to isolate the root cause of a failure, by exposing the relations between the success or failure run of the program to the behaviors of the program, these tools needs a huge raw data which can get from home grown tests, needs real data for better utilization.

Table 9: Automated debugging research techniques

Trust on Test Automation Framework.

Quality conscious teams looking at automation is one of the resorts to improve the software quality with speed. Test framework development often out sourced to a third-party team or a dedicated test automation team or may be within the team, automation frame work will be developed. Test automation development is also a sort of feature development, one can expect defects and if these defects were not uncovered early or before using on the regular test activity, later the same defects turn out be costly. This leads questions on the test automation framework capabilities and some distrust on the automation frameworks and scripts will be put up, if it continues. [68]

When test automation uncovers any defect, team need to isolate the defect cause, by analyzing the defect and determining that the defect is not from the framework/automation script or its from the application genuinely, in any case team requires good amount of time in analyzing, the effort is worth if its turn out to be an application defect, however if its from the framework or the automation script, mean a faulty automation caused the script failure the above said effort is of no use and efforts went into improvement of the rest automation.

The same has been resulted in the survey conducted [36] and 92% of the respondents expressed that they have trust their automation but there is scope to improve.

From normal application development point of view, it appears to be a regular and normal exercise that improving the automation after discovering the faults in it, but this is a post event activity. hence, there is need to “prevention of the defects” in the automation framework or automated test scripts.

This leads to come up with models to measure their frameworks, however the below mentioned models are about general test improvement process and not focused on the automation.

According to W. Afzal, S. Alone, K. Glocksien, and R. Torkar s’ systematic literature review conducted by many maturity assessment models have been proposed [69], and popular models have been described in the below table.

Name of the Model	Description	Comments
TMMi-Test Maturity Model integration [70]	Test Maturity Model integration (TMMi) is based on the Test maturity model (TMM), and these were inspired on the capability maturity model (CMM) and Capability maturity model integration (CMMi), TMMi is published by TMMi foundation TMMi proposed maturity levels and process, each maturity level has different process areas.	TMMi Helps in reducing the cost and time [73], over all TMMi defines goals at every stages and does not focus on how to achieve them . The phases and processes talk in general at a boarder level
Test Maturity Assessment-Test Process Improvement TMA-PI [71]	TPI has been developed by leveraging a good number of professional test engineers, TPI guides the teams to develop and offers a closer look at their maturity of their test process. In TPI a Test matrix has been defined which consists of Key Ares and Levels and for them check points and Improvement suggestions have been proposed. Key areas could be Test strategy environment, defect managements and lifecycle models etc. And levels have been defined as ABC, where C is higher, and A is the lowest, for each key areas, a grade will be assigned, and thus gives a proper insight of the whole model, and the focus can be on the low scoring areas to improve it further	TPI focuses one level down, and depends on the user to assign the ratings which becomes a qualitative model and the understanding or the level of maturity of the user defines the over all process maturity.

TPI -Next [72]	TPI-Next is process-driven and business driven, it is also called as Business Driven Test Process Improvement (BDTPI),it’s the extension on built upon the TPI, so it has the same approach, however along with TPI goals TPI Next has business related goals like or Drivers like Cost reduction, increasing effectiveness, improved transparency and business continuity.	Since TPI-Next is an extension of the TPI, TPI-Next also carries the same challenges like qualitative approach and does not guide the team to how to achieve due to its broader scope.

Table 10: Test Automation Models

FUTURE WORK & RESULTS

In this paper so far we have discussed, different development models, automation frameworks and challenges and provided a glimpse of solutions on which industry is looking into. To meet the needs of the industry, focus should be on both prevention and uncovering. Future work can be done the below areas.

Prevention activity:

- Time consuming tasks like test case selection prioritization can be further worked by applying AHP and Fuzzy techniques for betterment and accurate results.
- A Quantitative, robust and practical frame work to access the Developmental frameworks, either an automation framework related, or development related.
- Defining best practices in automating the functional, especially in test automation focusing on the sprints and in sprint automation.

Uncovering activity:

- Further study can be done in the field of isolating the fault in an optimized way to reduce the debugging time.

And there are many areas can be improved, as there is always scoping to improve in every activity, the above the work items one can look at for betterment of the test process.

CONCLUSION

Software testing is a fundamental activity evolved over many years by addressing many challenges. However, the new software development strategies and methodologies are continuously throwing new challenges to the software testing activity, one such challenge is to improve the time regression test time and hence cost, this is making more room for researcher to focus on. Software testing is a time consuming and an intensive process, Thus, enhanced techniques and innovative methodologies are needed in the current era of software engineering. In this paper, we attempted to highlight a few challenges and work done so far, to provide researchers to have a detailed and holistic view of pre and post activity in software testing activity and come with a few areas for a better way of testing and approaches to align software testing to the latest trends and challenges.

REFERENCES

1. P.I. Okwu1 and I.N. Onyeje2, software evolution: past, present and future, 2014.
2. Jeleel Adekunle Adebisi, fundamentals of computer studies,2013
3. https://dictionary.cambridge.org/dictionary/english/computer
4. Gerald d. Everett, Raymond Mcleod, jr. software testing across the entire software development life cycle, wiley-interscience a john wiley & sons,inc publication,2007.
5. Nayan b. Ruparelia,software development lifecycle models,acm sigsoft software engineering notes 2010, volume 35 number 3.
6. Benington, h.d. (1956): production of large computer programs. in proceedings, onr symposium on advanced programming methods for digital computers, june 1956, pp 15-27.
7. Royce, w.: managing the development of large software systems: concepts and techniques. in: proc. ieee wescom. ieee computer society press, los alamitos (1970).
8. Forsberg, Kevin and Mooz, Harold (1991): the relationship of system engineering to the project cycle. at ncose, chattanooga, tennessee, october 21- 23, 1991.
9. Doran, George t. (1981): there's a s.m.a.r.t. way to write management’s goals and objectives. in management review, vol. 70.11.
10. Mooz, h and forsberg, k. (2001): a visual explanation of the development methods and strategies including the waterfall, spiral, vee, vee+, and vee++models, pp 4-6.
11. Boehm, barry w. (1986): a spiral model of software development and enhancement. in acm sigsoft software engineering notes, vol. ii, no. 4, 1986,pp 22-42.
12. Jacobson i., booch g. & rumbaugh j. (1999): the unified software development process; addison-wesley, reading, massachusetts.
13. http://tryqa.com/what-is-rad-model-advantages-disadvantages-and-when-to-use-it/
14. https://techbeacon.com/app-dev-testing/agility-beyond-history-legacy-agile-development
15. Kent beck; james grenning; robert c. martin; mike beedle; jim highsmith; steve mellor; arie van bennekum; andrew hunt; ken schwaber; alistair cockburn; ron jeffries; jeff sutherland; ward cunningham; jon kern; dave thomas; martin fowler; brian marick (2001). "manifesto for agile software development". agile alliance. retrieved 14 june2010.
16. http://agilemanifesto.org/
17. https://en.m.wikipedia.org/wiki/extreme_programming
18. https://www.visual-paradigm.com/scrum/what-are-scrum-artifacts/
19. james a. whittaker, florida institute of technology, ieee software 17(1), pp. 70-79, jan-feb 2000.

20. Baker, c., review of d.d. mccracken's "digital computer programming". mathematical tables and other aids to computation 1 i, 60, october. 1957, pp. 298–305.
21. https://itnext.io/concept-evolution-of-software-testing-6fb1401b6df0
22. http://www.testingreferences.com/testinghistory.php
23. Gelperin, d., and hetzel, b. "the growth of software testing", communications of the acm, volume 31 issue 6, june 1988, pp. 687–695.
24. https://thenextweb.com/shareables/2013/09/18/the-very-first-computer-bug/
25. Burnstein. practical software testing. springer, 2003
26. Kaner, j. bach, and b. pettichord. lessons learned in software testing: a context-driven approach. john wiley & sons, inc., 2001.
27. Strang. data driven testing for client/server applications. in proceedings of the fifth international conference of software testing, analysis & review, pages 389–400. software quality engineering, 1996.
28. M. fewster and d. graham. software test automation. addison-wesley, 1999.
29. Gelperin, d. and hetzel, w., " software quality engineering," presented at fourth international conference on software testing, washington d.c., june 1987.
30. Rick d. craig,and stefan p.,jaskiel, systematic software testing,2002.
31. itti hooda,rajender singh chillar,software test process,testing types and techniques,ijca,vol 111 -no 13,feb 2015.
32. Pressman, r.s. 1997. software engineering: a practitioner approach.4th edition. tata mcgraw hill.
33. Tarika,bindia. computer programmer, cse, gndec, ludhiana, punjab-india.ijritcc.2,1 .68-72.(2321-8169).
34. Ananda Rao Akepogu Kiran Kumar J, An approach to cost effective regression testing in black-box testing environment - ijcsi international journal of computer science issues. 8, 3, 1(may 2011),(1694-0814).
35. Mohd.ehmer khan,different software testing level for detecting errors,ijse,vol (2):issue(4):2011
36. Kiran jammalamadaka, an emprical study on the test automation challenges in the agile scrum teams, ijret, eissn: 2319-1163 | pissn: 2321-7308,vol:03,issue:01,2014.
37. K. yu, m. lin, j. chen, and x. zhang, "towards automated debugging in software evolution: evaluating delta debugging on real regression bugs from developers' perspectives," journal of systems and software, to appear.
38. A. zeller and r. hildebrandt, "simplifying and isolating failure inducing input," ieee transactions on software engineering, vol. 28,pp. 183–200, february 2002.
39. https://www.adaptavist.com/blog/agile-testing/
40. Marwah alian,dima suleiman,adnan shaout,test case reduction techniques -survey,ijacsa,vol 7,no 5,2016.
41. X. ma, b. sheng, and c. ye, test-suite reduction using genetic algorithm,vol. 3756 of the series lecture notes in computer science, 2005, pp. 253-262, springer
42. B.subashini, d.jeyamala, reduction of test cases using clustering technique. international journal of innovative research in science, engineering and technology vol 3, special issue 3, 2014, international conference on innovations in engineering and technology (iciet'14). 1992-1995.
43. R. singh and m. santosh, test case minimization techniques: a review, international journal of engineering research & technology (ijert). vol. 2, no. 12. 1048 -1056.
44. Z. anwar and a. ahsan, multi-objective regression test suite optimization with fuzzy logic, ieee. inmic 2013.
45. P. harris and n. raju, a greedy approach for coverage-based test suite reduction, the international arab journal of information technology, vol. 12, no.1, 2015 pp. 17-23.
46. B. suri, i. mangal, and v. srivastava, regression test suite reduction using an hybrid technique based on bco and genetic algorithm, special issue of international journal of computer science & informatics (ijcsi), issn (print) : 2006, 2231–5292, vol.- ii, no-1, 2
47. B. suri, i. mangal, and v. srivastava, regression test suite reduction using an hybrid technique based on bco and genetic algorithm, special issue of international journal of computer science & informatics (ijcsi), issn (print) : 2006, 2231–5292, vol.- ii, no-1, 2.
48. M. weiser, program slicing. in proceedings of the 5th international conference on software engineering, icse '81, 1981, pp. 439–449, piscataway, nj, usa, ieee pres0s.
49. Amanda schwartz,hyunsook do,cost-effective regression testing through adaptive test prioritization strategies,elsevier,2016.
50. Yogesh singh, arvinder kaur, bharti suri and shweta singhal, systematic literature review on regression test prioritization techniques, informatica 36 (2012) 379–408,2011.
51. G. rothermel, r.untch, c.chu, and m.j.harrold, "test case prioritization: an empirical study",proceedings of international conference software maintenance, pp. 179-188, aug. 1999.
52. S. elbaum, a.malishevsky, and g.rothermel,"prioritizing test cases for regression testing",proceedings of the international symposium on software testing and analysis, pp. 102-112,aug.2000.
53. A.srivastava, and j.thiagarajan, "effectively prioritizing tests in development environment",proceedings of the international symposium on software testing and analysis, pp.97-106, july 2002.
54. R.C.bryce, s.sampath, a.m.memon, "developing a single model and test prioritization strategies for event driven software", ieee transactions on software engineering, pp.48-63, 2010.
55. F.belli, m.eminov, n.gokco. "coverage-oriented,prioritized testing-afuzzy clustering approach and case study". in :bondavalli.a.,brasileiro, f.,rajsbaum, s.(eds.) ladc 2007, lncs, springer,heidelberg, vol. 4746, pp. 95-110, 2007.
56. B.jiang, z.zhang, w.k.chan, t.h.tse, ".adaptive random test case prioritizatation." in proceedings of international conference on automated software engineering, pp:233-243, 2009
57. B. korel, g. koutsogiannakis, l.h. talat, "modelbased test suite prioritization heuristic methods and their evaluation", proceedings of 3rd workshop on advances in model based testing (a – most),london, uk, pp. 34-43, 2007.
58. B. korel, l. tahat, m. harman, "test prioritization using system models", in the proceedings of 21st ieee international conference on software maintenance (icsm'08), pp. 247-256, 2005.
59. J.M.Kim, a.porter. "a history-based test prioritization technique for regression testing in resource constrained environment", proceedings of the 24th international conference software engineering, pp.119-129, may.2002.
60. H. srikanth, l. williams, j. osborne, "system test case prioritization of new and regression test cases", in the proceedings of international symposium on empirical software engineering, (isese), pp. 64-73, nov. 2005.
61. G. rothermel, R.untch, C.chu, and M.J.harrold, "test case prioritization: an empirical study", proceedings of international conference software maintenance, pp. 179-188, aug. 1999.
62. A. Zeller, "isolating cause-effect chains from computer programs," in proceedings of the 10th acm sigsoft symposium on foundations of software engineering. acm, 2002, pp. 1–10

63. l. Naish, h. j. lee, and K. Ramamohanarao, “a model for spectra-based software diagnosis,” acm trans. softw. eng. methodol., vol. 20, no. 3, pp. 11:1–11:32, aug. 2011.
64. S. Parsa, M. Vahidi-asl, s. arabi, and b. minaei-bidgoli, “software fault localization using elastic net: a new statistical approach,” in advances in software engineering. springer, 2009, pp. 127–134.
65. M. Weiser. programmers use slices when debugging. communications of the acm, 25(7):446–452, july 1982.
66. X. Zhang, R. Gupta, and Y. Zhang. precise dynamic slicing algorithms. in ieee/acm international conference on software engineering, portland, oregon, may 2003.
67. F. Keller, l. Grunske, S. Heiden, A. Filieri, A. Van Hoorn, and D. Lo. A critical evaluation of spectrum-based fault localization techniques on a large-scale software system. in 2017 ieee international conference on software quality, reliability and security, qrs’17, pages 114–125, 2017.
68. http://luxagile.blogspot.sg/2007/05/five-obstacles-on-way-to-successful.html
69. W. afzal, s. alone, k. glocksien, and r. torkar, ″software test process improvement approaches: a systematic literature review and an industrial, case study,″ journal of systems and software, vol. 111, pp. 1-33, 1// 2016.
70. E. V. Veenendaal and B. Wells, Test maturity model integration (Tmmi): guidelines for test process improvement: uitgeverij tutein nolthenius, 2012.
71. T. Koomen and M. Pol, test process improvement: a practical step-by-step guide to structured testing: addison-wesley, 1999
72. A. V. Ewijk, B. Linker, M. V. Oosterwijk, and B. Visser, tpi next: business driven test process improvement: kleine uil, 2013.
73. David consulting group, ―what is tmmi® – and why should you care?‖ , pages [1- 2], may 2013.