2.1 OVERVIEW

A software process is defined as a set of methods, practices, activities and transformations that are used for obtaining and establishing a software and its related products. The issue of how software development should be organized, for speedy delivery of, better, and cheaper solutions is being discussed in software engineering circles for decades. Many remedies have been recommended, from the standardization and assessment of the software process to a huge number of concrete tools, techniques, and practices. Recently, many of the suggestions for improvement have come from experienced practitioners, who have labeled their methods as agile software development. This movement had an enormous impact on the modes operandi of software development worldwide. However, though there are many agile methods, there is only limited knowledge relating to how these methods are carried out in practice and what their effects are. The manifestation of agile methods has been the most noticeable change in software process development in the last fifteen years. Many reviews, studies, and surveys have been conducted on agile methods [Kaushal Pathak and Anju Saha, 2013].

In agile development, testing is involved throughout the development lifecycle, testing the software throughout the development. The developer is taught the testing techniques (white box and black box as well). Agile testing experts expect to work in a team that emphasizes cutting to chase. Testers do not have a testing phase; however, the developers are engaged in testing tasks performing testing by writing unit tests and testing them with the help of automated tools [Dyba, T. and
The researcher’s goal is to provide better quality software, yet following the agile manifesto. With automated unit testing, it becomes easier to validate and test the individual build features and also it is important to expect the build daily so that the integration can be done as and when along with the new build arrival [Chhavi Malhotra and Anuradha Chug 2013].

Agile software testing is performed using Directed Acyclic Graph-based Model (DAG-bM). In agile methodology, all works are divided into a number of modules. Each module contains decision making, planning, coding, testing, implementation, and so on. Each module follows this process and integrates with the previously completed modules. Finally, the entire work is integrated and the duration of work is made shorter than other software development models.

Software Testing can be described as the process of validating and verifying the product. Testing also ensures whether the product meets the business and technical requirement. Testing is done manually or by using automated tools. Agile testing is the software testing practice, which involves all members of the cross-functional agile team to ensure the business values desired by the customer [Konstantin Beznosov and Philippe Kruchten 2006].

### 2.2 AGILE METHODS

Agile process includes various methods like Scrum, XP (eXtreme Programming), Feature Driven Development (FDD), Crystal Clear Methodology (CC), Dynamic System Development Methods (DSDM) and Adaptive Software Development (ASD), Rational Unified Process (RUP).
2.2.1 Scrum

Scrum is an agile methodology; it has been developed for managing the process for system development. This approach does not focus on software development techniques for the execution state. It basically guides the management to see how the team members can perform in order to achieve flexibility where the requirements change frequently. Scrum team is a cross-functional team where every one participates in developing the product. It contains a series of iteration called sprints, and each sprint consists of backlog items, where backlog item corresponds to the amount of work to be done.

Scrum approach has been developed for managing the systems development process. It’s an empirical approach applying the ideas of industrial process control theory to systems development resulting in an approach that reintroduces the idea of flexibility, adaptability and productivity. Scrum concentrate on how the team members should function in order to produce the system flexibly in constantly changing environment. The main idea of the scrum is that system variables involve several technical and environmental variables (e.g. Requirements, time frame, and resources) that is likely to change during a process. Scrum is a set of guidelines that govern the development process of a product, from its design stage to its completion. Scrum is implemented with the help of Sprint [Chhavi Malhotra, 2013].

2.2.2 eXtreme Programming (XP)

Extreme Programming (XP) is the most popular of the entire new breed of simple, fast, and easy-to-use software development approaches. It is a software development methodology that has earned its importance in the arena of agile software development methodologies. XP emphasizes a lightweight, often informal
approach. There are no large-scale requirements, analysis, and design phases, and so there are none of the traditional metrics associated with the requirements or design phases, such as function points. The customer and development team agree a series of user stories that concisely define the requirements. User story sizes can be estimated using relative size measures such as story points that can be used to create initial project plans. However, the definition of a user story is not as well defined as a function point. As such, user stories are limited in their ability to predict effort or quality. Managers of XP projects have just as great a need for such predictions as managers on any other software project [Ramya Krishna, T.S. et al.2011]. Xp is aimed for small and medium sized teams. The team sized to be limited between three and a maximum of twenty project members. The physical environment is also important in XP. Communication and Coordination between project members should be enabled at all times [Abrahamsson, P. et al.2008].

2.2.3 Feature Driven Development

Feature Driven Development is a client-centric, architecture-centric software process. It is a software management method which involves planning, up-front modeling, and design. FDD is a lightweight, iterative and incremental software development process. It emphasizes quality and delivers frequent, tangible working results at all steps. FDD also provides accurate and meaningful progress and status information, with the minimum of overhead and disruption for the developers. It blends a number of industry-recognized best practices into a cohesive whole. Discovery of a list of features is a critical process. The quality of this step largely defines how precise the project will be tracked, how maintainable and extensible the code will be. This process requires full-time participation of customers. FDD consists
of five high level activities: Domain Object Modeling: It advocates light modeling on a full time basis up front to understand the shape and scope of the application [Munazza Umbreen et al. 2015].

2.2.4 Crystal Clear Methodology

The crystal clear methodology is the smallest series methodology in the software development which involves small teams. It is prioritized for project safety, effectiveness and endurable. Crystal family is marked with color, indicating the heaviest of the methodology (darker the color heavier the methodology). Crystal methodology does not restrict any development practices, tool or work product rather it allows adopting of eXtreme Programming (XP) and scrum practices.

2.2.5 Dynamic System Development Methods (DSDM)

Dynamic System Development Methods (DSDM) is an iterative and incremental approach that embraces the principle of agile development including customer engagement. Adaptive software development is a design rule for the creation of software products which concentrate on rapid creation and evolution of software.

2.2.6 Rational Unified Process (RUP)

Rational Unified Process (RUP) divides a project into development cycles each of which is divided into phases: namely, inception, elaboration, construction, and transition. In turn, each phase consists of development iterations, where each iteration produces useful (ideally executable) artifacts. Rational Unified Process (RUP) identifies a series of “workflows”, or topics, involved in software development: business modelling, requirements, analysis and design, implementation, testing, deployment, Rational Unified Process (RUP) is actually a “meta-process” that can be
configured to form a wide variety of different software development processes. Frequently, Rational Unified Process (RUP) is presented with a configuration that makes it an agile software development process. Configuration and change management, project management, and environment management. All workflows are addressed in each phase, but each workflow the importance of and effort ebbs and peaks differently across the phases.

2.3 CHARACTERISTICS OF AGILE METHOD

➤ Modularity

Modularity is the key element of any good process. Modularity allows a process to be broken into components called activities. A software development process prescribes a set of activities capable of transforming the vision of the software system into reality.

➤ Iterative

Agile software processes acknowledge that we get things wrong before we get them right. Therefore, they focus on short cycles. A certain set of activities is completed within each cycle. These cycles are started and completed in a matter of weeks. However, a single cycle (called an iteration) will probably not be enough to get the element 100% correct. Therefore, the short cycle is repeated many times to refine the deliverables.

➤ Time-Bound

Iterations become the perfect unit for planning the software development project. Time limits (between one and six weeks is normal) can be set for each iteration and they can be scheduled accordingly. Chances are, (unless the process
contains very few activities) all the activities of the process are not scheduled in a single iteration. Instead, attempts are made only on those activities necessary to achieve the goals set out at the beginning of the iteration. Functionality may be reduced or activities may be rescheduled if they cannot be completed within the allotted time period.

- **Parsimony**

  Agile processes are more than just a traditional software development process with some chronological constraints. Attempting to create impossible deadlines under a process not suited for rapid delivery puts the onus on the software developers. This leads to burnout and poor quality. Instead, agile software processes focus on parsimony. That is, they require a minimal number of activities necessary to mitigate risks and achieve their goals.

- **Adaptive**

  During an iteration, new risks may be exposed which require some activities that were not planned. The agile process adopts the process to attack these new found risks. If the goal cannot be achieved using the activities planned during the iteration, new activities can be added to allow the goal to be reached.

- **Incremental**

  An agile process makes no attempt to build the entire system all at once. Instead, it partitions the nontrivial system into increments which may be developed in parallel, at different times, and at different rates. We unit test each increment independently. When an increment is completed and tested, it is integrated into the system.
Convergent

Convergence states that we are actively attacking all risks worth attacking. As a result, the system becomes closer to reality that we seek for each iteration. As risks are being proactively attacked, the system is being delivered in increments.

People-Oriented

Agile processes favour people over process and technology. They evolve through adaptation in an organic manner. Developers that are empowered raise their productivity, quality, and performance. After all, they are the best individuals in the organization to know how to make these changes.

Collaborative

Agile processes foster communication among team members. Communication is a vital part of any software development project. When a project is developed in pieces, understanding how the pieces fit together is vital to creating the finished product. There is more to integration than simple communication. Quickly integrating a large project while increments are being developed in parallel, requires collaboration [Gurleen Singh and Tamanna, 2014]

2.4 PRINCIPLES OF AGILE METHODS

The principles of agile methods typically lead to specific practices such as the following:

- Use regular rapid cycles which create executable deliverables
- Focus on coding rather than planning or documentation
- Re-factor continually to improve code
Communicate continually and extensively within the engineering development team

Communicate continually and extensively with customers

Continually measure project progress, extrapolate projections, adjust long-term project goals (project end date and feature set), and set short-term goals (work elements for the next iteration)

Use test-driven development to verify that code is initially correct, and emphasize regression testing to ensure that the code stays correct

2.5 INDUSTRY OPINIONS REGARDING AGILE METHODS

eXtreme Programming (XP) is the most well-known, most interesting and most polarizing agile method. It has generated much buzz and fervour in the industry and by now has attracted a sizeable share of practitioners. Many supporters view it as the leader of development practices tainted by the waterfall. Others view it as a pendulum swing that has passed beyond a reasonable medium in its contrarianism to past overly-rigid development practices. Speeches of moderation until lately, people discuss about the effectiveness of eXtreme Programming (XP) were greeted with scorn, but now notable voices of moderation or dissent have spoken, pointing out potential problems with eXtreme Programming (XP).

2.6 INTRODUCTION TO TEST CASE PRIORITIZATION TECHNIQUES

Test case prioritization techniques provide a way to schedule and run test cases, which have the highest priority in order to provide earlier detect faults [Junaid Arafeen, Md. and Hyunsook Do 2013]. This study presents numerous techniques developed, between 2002 and 2008, that can improve a test suite’s rate of fault detection. With current test case prioritization techniques researched in 1998-2008,
this paper presents and organizes a new “4C” classification of those existing techniques, based on their prioritization algorithm’s features, as follows:

2.6.1 Customer Requirement-Based Techniques

Customer requirement-based methods are approaches to order test cases constructed on requirement documents. Many researchers have done investigation in this area, such as many weight factors have been used in these techniques, including custom-priority, requirement complexity, and requirement volatility. A multi-faceted prioritization strategy called Prioritization of Requirements for Testing (PORT Version 1.1) by exploring three prioritization factors (PFs): (1) customer-assigned priority on requirements, (2) requirement complexity, and (3) requirements volatility. The researcher’s preliminary set of research goals is listed below. G1: To identify the most severe faults/failures earlier in system test. G2: To improve the software field quality. G3: To devise the minimal set of PORT PFs that can be used to effectively for TCP. The PFs, customer-assigned priority (CP), requirements complexity (RC) and requirements volatility (RV) are assigned values. CP is the value (1 to 10) assigned by the customer based on the importance of the requirement. RC is the value (1 to 10) assigned by the developer based on the perceived implementation difficulties of the requirement. RV is the number of times a requirement has changed. Higher factor values indicate a need for prioritization of test case related to that requirement [Hema Srikanth, et al. 2005].

2.6.2 Coverage-Based Techniques

Coverage-based methods are approaches to order test cases such as requirement coverage, total requirement coverage, additional requirement coverage and statement coverage. Many scholars have researched this area.
2.6.3 Cost Effective-Based Techniques

Regression testing is considered as one of the most expensive tasks in software maintenance activities. Such a technique uses the test-suite developed for an earlier version of a software system to conform the new added requirement in the current version. Selecting all or a portion of the test-suite to execute which is referred to as Regression Selection Techniques (RST) can be very costly. Furthermore using RST, testers do not have the option to adjust their test-effort to their budget. To provide the missing flexibility, researchers have introduced prioritization techniques by means of which testers can order the test cases based on certain criteria, and then run them in the specified order and as much as they can afford. [Mirarab S. and Tahvildari, L 2007] proposes novel test-suite prioritization framework which integrates various sources of information into one single model. This technique is based on a probabilistic specification of the problem. The prioritization approach is based on ordering test cases according to their success probability. The proposed process uses conditional probability and utilizes a Bayesian Network model which takes advantage of source code modification information, univariate measures of fault-proneness, and test coverage data. Cost effective-based techniques are methods to prioritize test cases on the basis of costs, such as cost of analysis and cost of prioritization. Many researchers have investigated this area, for instance.

2.6.4 Chronographic History-Based Techniques

Chronographic history-based methods are approaches to order test cases on the basis of test case execution. A few researchers have researched this area as following sections describe the above techniques in details. Software systems and their
environments change continuously. They are improved, modified, and ported to new platforms. These changes can affect a system adversely. Hence software engineers perform regression testing to ensure the quality of the modified systems. Regression testing is responsible for a significant percentage of the costs for software maintenance and the maintenance costs often dominate total lifecycle costs. Hence regression testing is one of the largest contributors to the overall cost of software. This has been done for improving the cost effectiveness of the regression test techniques. Many researchers have proposed and empirically studied various regression testing techniques, such as regression test selection, test suite minimization and test case prioritization.

2.6.5 Methodology

The light gray boxes depict the main activities, and the ovals depict inputs and outputs associated with the activities. The approach consists of five main activities: requirements clustering, requirements-tests mapping, prioritization of test cases for each cluster, cluster prioritization, and test case selection from the clusters. The following subsections describe each activity in detail. To cluster the requirements, we use textual similarity.

Textual similarity has been studied in the field of text mining for clustering documents and information retrieval, the requirements are grouped into clusters on the basis of the distribution of words that co-occur in the requirements. This process includes three tasks, namely, term extraction, term-document matrix construction, and k-means clustering. These tasks are explained below:
2.6.5.1 Term Extraction

Term extraction considers each user requirement as a bag of words or terms. In this process, words are extracted from each requirement. After eliminating these words, all distinct terms across all the requirements are identified and used in the subsequent tasks.

2.6.5.2 Term-Document Matrix

Term document matrix uses the distinct terms obtained from the previous step to create a term-document matrix. In this matrix, the rows correspond to the requirements, and the columns correspond to the distinct terms across all requirements. The matrix can be built in many ways. For instance, the matrix cells can list Boolean values indicating whether the terms are present in the corresponding requirements. The matrix can tilt the frequency of the word in the equivalent requirements or can list the term frequency-inverse document frequency.

2.6.5.3 Requirements

➢ Tests Mapping Resolution

Once the clusters of requirements are obtained, the requirement-test cases traceability matrix utilized for collecting test cases that are associated with each requirement cluster. The requirements-tests mapping resolution process obtains the clusters of test cases (the ovals on the right side of the figure) by reading the requirements in the clusters

➢ Test Case Prioritization

Having created clusters, prioritization techniques are applied to them. There are many ways to prioritize test cases as already explained. The code complexity
metric 1 to calculate a code complexity metric, three types of information obtained from source code (Lines of Code, Nested Block Depth and McCabe Cyclomatic Complexity) have been used because they are considered as good predictors for finding D. Cluster Prioritization Not all requirements are equally important to clients. Therefore, certain software components associated with requirements that are more important to clients could be more frequently utilized by users when software is deployed.[ Ho, C.-W. Johnson et al. 2006].

It means that certain requirements need to get more attention from testers. The requirement clusters could be ordered so that it can utilize their priority data when they select test cases from each cluster to gain the entire set of reordered test cases. That is, the cluster with the higher priority can be visited earlier or more tests can be selected from it. Often, companies order requirements and apply them incrementally on the basis of a customer’s need and the product delivery schedule. From one of the projects used in our experiment, the researcher observed that, before each iteration of development, software developers categorize the requirements on the basis of importance as follows:

- **Commit (C):** Developers implement the given requirements (High Priority).
- **Target (T):** Developers strive to implement the given requirements, but they will not guarantee to do so (Medium Priority).
- **No-Commit (NC):** Developers implement the given requirements if they have time (Low Priority).

**2.7 AGILE METHODOLOGY IN VARIOUS PLATFORMS**

Agile methodology is adopted mainly to increase software development, reduce documentation, and satisfy customers through continuous delivery of the
product. Most of the software companies accept only agile based projects because the time duration is smaller than for the waterfall model [Siripong Roongruangsuwan and Jirapun Daengdej 2010]. The Waterfall model takes considerable time duration and changes affect the entire work- from the beginning to the end, but, changes do not affect at any levels in the agile method. In agile methodology, user interaction is most important at all the level to produce the right product. Customers’ requirements are added at any level instantly, and these requirements are created as use cases i.e. user board story; these use cases are split into many test cases.

The tester designs the test cases depending upon the type of the testing. The collection or set of many test cases is known as Test Suite. Therefore, during the testing phase, tester can decide which test case to be tested first, then achieves the software with those test cases, and then checks and verifies the results produced by the execution.

2.8 EFFECTIVENESS OF TEST CASE PRIORITIZATION

Test case prioritization methods plan test cases for execution in a command that attempts to raise their efficiency in meeting some goal line. Various goals are possible; one involves the rate of fault detection a measure of the speed with which faults are sensed within the testing process. An upgraded rate of fault detection throughout testing can give a faster feedback on the method under assessment, and make software engineers initiate rectification of faults earlier than might be potential.

2.8.1 Coverage-Based Test Case Prioritization

Coverage-based TCP techniques involve ranking test cases based on the statement coverage they provide. Test cases are ranked based on the various
statements executed/enclosed by the test case such that the additional line of code executes with the higher level of test coverage [AlameluMangaiyarkarasi and Srinath, M.V. 2016]. For branch and function coverage techniques, tests are prioritized on the basis of the program branches or program functions covered, respectively. Research has exposed the dedication of prioritization of system configurable software not only by fault detection but as well by the cost of configuration and setup time progresses primary fault detection rate.

The benefits of code coverage-based TCP strategies were measured using the Average Percentage of Faults Detected (APFD) metric. The Average Percentage of Faults Detected (APFD) value is a measure of the quickness in identification of faults for a given test suite. The Average Percentage of Faults Detected (APFD) values range from 0 to 100 and are monitored during test suite execution [Vivekananda Reddy, D. and Rama Mohan Reddy, A. 2016]. The Average Percentage of Faults Detected (APFD) values represent the area under the curve in a plot of percentage of faults detected (y-axis) against the percentage of test suite run (x-axis) of a graph. Assuring a scenario with 10 faults labelled 1 through 10 and five test cases labelled A through E, there are 120 possible ways of ordering the five test cases. The faults are detected by each test suite. If the choice is to execute the tests in the order C-E-B-A-D, the Average Percentage of Faults Detected (APFD) score would be 84%, as determined by the area under the curve. A similar validation metric is analysed to assess the efficacy of PORT, Weighted Percentage of Failures Detected (WPFD).

When all faults are not equally severe, severity-neutral TCP strategies and the associated APFD metric can provide misleading information. As a result, a fault severity may be incorporated in a cost-cognizant Average Percentage of Faults
Detected (APFD) TCP strategy Instead of representing Test Suite Fraction on the x-axis, Percentage of Total Test Case Cost Incurred is considered in Average Percentage of Faults Detected (APFD). Additionally, instead of representing Percent Faults Detected on the y-axis (as for APFD), Percentage Total Fault Severity Detected is considered in Average Percentage of Faults Detected (APFD).

Average Percentage of Faults Detected (APFD) offers a higher reward to those test case orderings that reveal the most severe faults earlier. The ordering C-E-B-A-D gives an APFD score of 84% while E-C-B-A-D gives a score of 76%. If faults 1–10 do not have the same severity an ordering that yields a better Average Percentage of Faults Detected (APFD) score may not yield a better Average Percentage of Faults Detected (APFD) score.

For instance, if test cases A, B, D, and E take 1 h to complete and test C takes 10 h to complete then C-E-B-A-D would yield a lower Average Percentage of Faults Detected (APFD) score as we are running a higher cost test (C) earlier in the cycle. Average Percentage of Faults Detected (APFD) can be used for assessment of the prioritization orders post hoc, i.e. when the severity and cost values are known, and cannot be used for predicting cost and severity values. In the validation of Average Percentage of Faults Detected (APFD) metric, Elbaum assign six levels of severity to the faults in the program.

2.8.2 Software Reliability Engineered Test (SRET)

Software Reliability Engineered Testing (SRET) is a method test approach that involves testing software systems guided by reliability objectives, and the criticality of the system operations with the main goal of achieving the desired
software reliability while incurring minimum cost. The Software Reliability Engineered Testing (SRET) approach is discussed and comparison of the researcher’s and work with Software Reliability Engineered Testing (SRET) is made. Test planning using Software Reliability Engineered Testing (SRET) declares estimating the absolute client use of the functions in a software product via a functioning profile.[Li-Xin Jiang,2012] describes an operational profile is a set of operations and their probabilities of occurrence where an operation is a task that the system performs. Operational profile estimation is done collectively by testers, system engineers, architects, and customers. Once the operational summary has been assessed, testers regulate the total number of test cases that can be written and accomplished on the basis of pecuniary and resource constraints.

Then, the number of test cases written for each operation is a relative proportion of the total number of test cases based upon the use percentage for that operation in the operational profile. The Software Reliability Engineered Testing (SRET) approach is an important consideration for test prioritization, utilization of an operational profile allows for the allocation of tests based on the proportion of usage. This prevents the allocation of excessive testing resources to seldom used operations. Software Reliability Engineered Testing (SRET) is also the only known operational profile based approach at the system test level. Once the tests have been chronicled, they are executed in a random order to enable tracking of reliability growth via system test progress.

2.8.3 Model-Based Test Prioritization

Model-based test prioritization has been proposed as an alternative to traditional code-based test prioritization methods. One advantage of a model-based
approach is the low cost of executing the model as opposed to the actual system. Dependence on testers/developers to provide accurate information is a drawback of the model based approach. In the model-based test prioritization system, models are used for prioritizing test cases. Several modeling languages such as State Charts, Extended Finite State Machine (EFSM), and Specification Description Language (SDL) have been developed to model state-based software systems [Braun, P. and Eckhaus, R. 2008]. Model-based test prioritization may be difficult to implement in state-less systems, the PORT approach can be applied to both a stateful and a stateless system. At the crux, model-based test prioritization utilizes the notion that changes in specifications leads to changes in the model. Therefore, the differences in the models can be utilized to prioritize the test suite. At the onset, the modified model is executed for the complete test suite and information is garnered on any modifications, prioritization is then made based on the information collected.

The end goal of such an approach is to detect faults in the modified system. In the work in the researcher utilize Extended Finite State Machine (EFSM) for model-based test prioritization. In this approach, two models are created to represent the system – one for the original system and one for the modified system. The difference between the models is identified by two sets – one constituting the additions and the other constituting the deletions made to transitions in the original model to obtain the modified model. One prioritization approach may be a selective prioritization, whereby a test is given a higher priority if it invokes a newly added transition. Neural network based models have also been used to predict the software effort. The inputs of these neural network models include software size in UCP and other quality attributes such as complexity [Nassif A.B. et al. 2013].
Table 2.1 Comparative Study of Agile and Test case prioritization technique

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Author Name</th>
<th>Issue (s) and Approach</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Konstantin Beznosov and Philippe Kruchten 2004.</td>
<td>The Software Testing can be described as the process of validating and verifying the product. Testing also ensures whether the product meeting business and technical requirements</td>
<td>Testing is done manually or by using automated tools. Agile testing is the software testing practice, which involves all members of the cross-functional agile team to ensure the business values desired by the customer.</td>
</tr>
<tr>
<td>2.</td>
<td>Md. Junaid Arafeen and Hyunsook -2010</td>
<td>The software testing process contains numerous Test case prioritization techniques are available, that provide a way to schedule and run test cases, which have the highest priority on the basis of detect faults</td>
<td>Test case prioritization techniques schedule test cases to run more important test cases earlier so that faults can be detected earlier or earlier feedback can be provided to testers. Most of these techniques depend primarily on software code information, including code coverage or code dependency relations.</td>
</tr>
<tr>
<td>3.</td>
<td>Siripong Roongruangsuwan and Jirapun Daengdej 2010.</td>
<td>Agile methodology is adopted mainly to increase software development, reduce documentation, and satisfy customers through continuous and unfailing delivery of the product</td>
<td>Most of the software companies accept only agile based projects because the time duration is smaller than the waterfall model. Waterfall model takes considerable time and changes affect the entire work- from the beginning to the end.</td>
</tr>
<tr>
<td>4.</td>
<td>Alamelu Mangayarkarasi and Srinath, M.V December 2016</td>
<td>Describe Coverage-based TCP techniques involving ranking test cases based on the statement coverage they provide. Test cases are ranked on the basis of the number of statements executed/covered by the test case such that the more lines of code the test executes, the higher the test coverage</td>
<td>For branch and function coverage techniques, tests are prioritized on the basis of the program branches or program functions covered, respectively. Research has shown that prioritization of system configurable software is driven not only by fault detection but also by the cost of configuration, and setup time improves early fault detection rate.</td>
</tr>
<tr>
<td>S.No.</td>
<td>Author Name</td>
<td>Issue (s) and Approach</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>5.</td>
<td>Hema Srikanth&amp; Sean Banerjee 2012</td>
<td>Software testing is defined as the process of verifying and validating the correctness of the developed software product. Software testing is the last phase before delivering the product to the customer.</td>
<td>Some of the important properties of the agile testing are they are unstructured, faster implementation, testers, and developers cooperate with each other, and testing is initiated at the end of every logic release. Multiple software development industries have started using the agile methods for software development. When compared to the traditional software development methods.</td>
</tr>
<tr>
<td>6.</td>
<td>Alamelu Mangayarkarasi, V and Srinath M. V. 2016</td>
<td>Explain that the Software development for mobile platforms comes with unique features and constraints that apply to most of the lifecycle stages.</td>
<td>The merits of using the test case prioritization are as follows: Economy in resource utilization Minimal time consumption Increased fault detection rate Enhanced system reliability Increased function test coverage.</td>
</tr>
<tr>
<td>7.</td>
<td>Andrei Cristian Spataru 2010</td>
<td>The sporadic task model is a well-known model to represent real-time systems based on a finite number of independent recurrent processes or tasks, each of which may generate an unbounded sequence of jobs.</td>
<td>Determining how multiple recurrent tasks can be scheduled on a shared unit- or multiprocessor platform is one of the traditional subjects of study in real-time scheduling theory.</td>
</tr>
<tr>
<td>8.</td>
<td>Vincenzo Bonifaci, Alberto Marchetti-Spaccamelay, Sebastian Stillerz, and Andreas Wiese 2010</td>
<td>Explain an over roughly the past decade, iterative methods of software development have gained acceptance, largely displacing older methodologies such as the waterfall or V models of software development.</td>
<td>This paper discusses how the distinguishing aspects of embedded systems software development affect the application of agile methods to embedded systems. Although the net result of using agile methods in embedded system software development is an improvement, careful consideration is required to achieve potential benefits.</td>
</tr>
<tr>
<td>S.No.</td>
<td>Author Name</td>
<td>Issue (s) and Approach</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>9.</td>
<td>Doug Dahlby 2004</td>
<td>Describe as there are many reasons to practice test-driven development (TDD). It encourages communication between customers and developers, increases programmer confidence, improves software quality, and (arguably) decreases bug density without decreasing productivity.</td>
<td>As most published approaches to TDD test the software layer just underneath the GUI, test-driven development of graphical user interfaces (GUIs) remains an unsolved problem. The reason for this is twofold. First, GUIs are very likely see repeated changes over the course of development. This indicates the need for tests to be updated and repaired frequently—which is a nontrivial task. Second, the easiest method of creating GUI tests—using a capture-replay tool (CRT)—requires that a GUI exists before tests can be defined.</td>
</tr>
<tr>
<td>10.</td>
<td>Theodore D. Hellmann, Ali Hosseini-Khayat and Frank Maurer 2010</td>
<td>Cloud computing is an emerging paradigm for providing services and solving large-scale problems in science, engineering, and commerce. The initial challenges of cloud computing—in the areas of providing a service, managing multiple virtual machines on different systems—have been resolved to the first degree.</td>
<td>The use of cloud resource management is far from ubiquitous. This is due to the fact that scheduling and mapping decisions have to take into account the myriad standards, procedures, and devices in a highly dynamic environment.</td>
</tr>
<tr>
<td>11.</td>
<td>Dzmitry Kliazovich, Johnatan E. Pecero and Andrei Tchernykh 2013</td>
<td>Regarding the software development industry, the culture of 'outsourcing' has become popular taking advantage of economic factors and skills. In terms of this culture, companies have been decentralizing development or testing units through contracts or outsourcing processes into different zones.</td>
<td>Proper execution and estimation important in the software development life cycle. Such an example in agile scrum software development, the team needs to estimate the test. Normally its team does it manually through open consultation. Later the scrum master needs to measure test effort by a standard process and find out the estimated effort from the team really feasible or not. If not, SM requests the team to re-estimate the test again.</td>
</tr>
<tr>
<td>S.No.</td>
<td>Author Name</td>
<td>Issue (s) and Approach</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>12.</td>
<td>Abu Wahid Md. Masud Parvez 2013</td>
<td>Vulnerable Web applications constitute a fair game to malicious users who conduct cyber-attacks. Secure coding still needs to be put into practice by programmers themselves even if the latest secure Web application framework is availed.</td>
<td>There is a need for comprehensive and automated security testing to find and cope with unavoidable human errors. The existing security testing methods can be classified into two approaches: static and dynamic. Static analysis directly checks application source code to detect adverse coding styles that cause vulnerabilities.</td>
</tr>
<tr>
<td>13.</td>
<td>Seiji Munetoh and Nobukazu Yoshioka 2013</td>
<td>New advances in mobile computer technology are emerging speedily in the software engineering field. The number of mobile devices with computation capabilities incorporated (such as third-generation mobile phones and personal digital assistants) is growing all over the world.</td>
<td>Usage of mobile commerce applications and services. Examples of such commercial applications include mobile information services, advertisement, content-based services, location-based services, and mobile payment applications.</td>
</tr>
<tr>
<td>14.</td>
<td>Vahid Rahimian and Raman Ramsin 2006.</td>
<td>Due to slowing down of the rate of increase of clock frequencies, most processor chip manufacturers have recently moved to increasing performance of processors by increasing the number of cores on each chip.</td>
<td>The rapid evolution of multi-core processor technology, real-time system software and programming models have failed to keep pace. In particular, most classic results in real time scheduling concentrate on sequential tasks running on multiple processors or cores.</td>
</tr>
<tr>
<td>15.</td>
<td>Abusayeed Saifullah, David Ferry, Kunal Agrawal, Chenyang Lu, and Christopher Gill - 2011</td>
<td>Embedded systems are pervasive in the world we live in today, the extent of which is illustrated by Egger Mont, who notes that the average home today has over 50 embedded systems. Organizations involved in the lifecycle (design, development, and sustainment) of these systems, have to manage the growing operational and environmental complexity that these systems face.</td>
<td>Determine the state of the practice in embedded software engineering one of the key findings in the study was that systems engineering decisions are largely being driven by hardware constraints, which then impact software efforts two stages later in the lifecycle when software requirements at the component level are developed.</td>
</tr>
</tbody>
</table>
2.9 INTRODUCTION TO MVC CONTROLLER

Agile testing is software testing method that provides a continuous iteration of development and testing throughout the software development life cycle. Some of the important properties of the agile testing are they are unstructured, faster in implementation [Hema Srikanth. and Sean Banerjee. 2012]. Testers, and developers cooperate with each other with and testing initiated at the end of every logic release. Multiple software development industries have started using the agile methods for software development. When compared to the traditional software development methods, the agile methods have the following advantages: Accelerate time to market, Increase quality, Increase productivity, Enhance Information Technology (IT), Enhance flexibility.

The existing techniques used for the agile testing are Model View Controller (MVC) approach and Just In Time (JIT) approach and Directed Acyclic Graph (DAG) model. These techniques consider the methods through which the test cases are generated for agile testing but they do not execute the test cases in a sequential manner. Thus, to address this issue, a Directed Acyclic Graph (DAG) based model is proposed. Generally, the lack of time and resources in the companies prevent their ability to perform the testing. Thus, prioritizing the execution order of the test cases can enhance the software testing efficiency.

Test case prioritization is the process of scheduling the order of test case execution to enable the execution of higher priority test cases first. The merits of using the test case prioritization are as follows: Limited resource utilization, Minimal time consumption, increased fault detection rate, Enhanced system reliability and increased function test coverage.
The Software development for mobile platforms comes with unique features and constraints that apply to most of the lifecycle stages. The development environment and the technologies that support the software are different from the “traditional” settings. The most important distinguishing characteristics are identified. Environment particularities include a high level of competitiveness; necessarily short time-to-delivery; and added difficulty in identifying stakeholders and their requirements. Users' positive attitudes toward computers have been found to be a likely indicator of software products acceptance and there is strong support for a dependency between attitudes and satisfaction [Mahmood, M. A. et al. 2000]. Development teams need to face the challenge of a dynamic environment, with regular adjustments with customer needs and beliefs. Technological constraints apply to mobile platforms in the form of limited physical resources and rapidly changing specifications [Carton, A. et al. 2007][Kunz, T. and Black, J. 1999]. There is also a great variety of devices, each with particular hardware characteristics, firmware and operating systems.

The writer references two types of restraints, viz., evolving and integral. Evolving constraints, such as bandwidth, coverage, and security, currently apply to the mobile technology, but are likely to be addressed and possibly resolved in the near future. On the additional hand, inherent constraints such as limited display real land, reduced data entry ability (due to a limited keypad), memory capacity, processing power and limited power reserve, are long-lasting, at least relative to desktop surroundings. Various approaches must be used for mitigating the impact of inherent constraints.
Due to major differences in the atmosphere and in platform specifications, mobile application development needs a suitable development approach. By taking into account the main features of a mobile application development scenario, a matching development paradigm can be identified. To develop the learning-oriented and semi-automated approach, first identified the particular characteristics and quantitative metrics of globally distributed software projects that are important to early-stage cost estimation and available at the start of the project presented in the software is released in an uncertain and dynamic environment with high levels of competition[Ramasubbu N. and Balan, R.K, 2012]. Teams that develop mobile applications are usually small to medium-sized, co-located, and generally use object-oriented tools and practices. The applications are small-sized, are not safe, and do not ensure satisfaction of interoperability or reliability constraints. They are delivered with rapid releases in order to meet market demands and are targeted at a large number of end-users [Alamelu Mangayarkarasi V and Srinath, M. V. 2016] [Kim, H. et al. 2009]. The author suggests agile methods as a suitable approach to development, by comparing the above features to agile “home ground” characteristics: small-scale, application-level software, developed in a highly dynamic environment by a small to the medium-sized team using object-oriented approaches, in relatively short development cycles. The following section provides a short overview of agile methods, focusing on their suitability for mobile application development.

2.10.1 Recent Agile Mobile Application Growth

Agile techniques represent a comparatively new methodology to software development, becoming wide-spread in the last years. The thoughts behind these
approaches originate from the principles of Lean Manufacturing and Agile Manufacturing, which highlight the flexibility of enterprises to an active environment [Amanquah, N. and Eporwei, O.T. 2009].

The unique features of agile methods are derived from the list of principles found in the “Agile Manifesto”: individuals and interactions are more important than processes and tools, working software is more valuable than comprehensive documentation, customer collaboration is preferred over contract negotiation, and adaptability is valued higher than creating and following a plan. In Boehm & Turner, the authors identify fundamental concepts to agile development: simple design principles, a large number of releases in a short time frame, extensive use of refactoring, pair programming, and test-driven development.

2.11 FEASIBILITY ANALYSIS IN THE SPORADIC DAG TASK MODEL

The sporadic task model is a well-known model to represent real-time systems based on a finite number of independent recurrent processes or tasks, each of which may generate an unbounded sequence of jobs. Determining how multiple recurrent tasks can be scheduled on a shared unit- or multiprocessor platform is one of the traditional subjects of study in real-time scheduling theory [Andrei Cristian Spataru, 2010]. Different formal models have been proposed for representing such recurrent tasks; these models differ from one another in the restrictions they place on the jobs that may be generated by a single task it is well-known that the technological evolution of processor manufacturing is moving away from increasing clock frequencies to increasing the number of cores per processor.
In the sporadic Directed Acyclic Graph (DAG) model, a task is represented as a Directed Acyclic Graph (DAG) $G = (V; E)$; the task repeatedly emits a dag-job, which is a set of precedence-constrained sequential jobs. More precisely, in each vertex $v \in V$ of the Directed Acyclic Graph (DAG) corresponds to a sequential job and is characterized by a worst-case execution time (WCET) $iv$. Each (directed) edge of the Directed Acyclic Graph (DAG) represents a precedence constraint: if $(v; w) \in E$ is a (directed) edge in the Directed Acyclic Graph (DAG), then the job corresponding to vertex $v$ must complete execution before the job corresponding to vertex $w$ starts execution. Any groups of jobs that are not constrained (directly or indirectly) by precedence constraints among each other may execute in parallel, whenever enough processors are available for them. This implies that jobs of subsequent dag-jobs of the same task can be scheduled in parallel.

### Table 2.2 Comparative of test case and DAG Algorithm

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Author Name</th>
<th>Issue (s) and Approach</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Jayakanth Srinivasan, Radu Dobrinand Kristina Lundqvist 2007.</td>
<td>Firewalls, which act as the most important defense mechanism of network security, have to be tested for validation of their work as specified. The firewall specification is mainly composed of intended security policy and allowed network protocols, which are usually the main focus of an attacker</td>
<td>The novelty of this approach is the use of DAG model for firewall testing. This paper proposes a modeling of firewall rules and generating test cases using DAGs. Since event sequence graphs (ESG) are directed graphs, we applied its test case generation algorithm to the DAG representation of firewall rules</td>
</tr>
<tr>
<td>2.</td>
<td>Krishnamoorthi, R. Mary, S. S. A 2009</td>
<td>At the time of the formal testing, phase software testers develop the test suites. These test suites often keep saving for their reuse from future perspective. Rerunning such tests is expensive in terms of time and effort.</td>
<td>Uses Agglomerative Hierarchical Clustering technique to group the test cases and code coverage, fault detection ratio, and code complexity metric separately to re-order the test cases in each phase</td>
</tr>
<tr>
<td>S.No.</td>
<td>Author Name</td>
<td>Issue (s) and Approach</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>test suites can cost one-half of the total cost requires spending in the maintenance phase and take unjustifiable excessive time, For example, an industry reported that executing all the test cases at the maintenance phase required seven weeks for a product having 20000 lines of code</td>
<td>The approach proposed as relies upon requirement based clustering and used K-means for clustering and nested block depth, Lines of code and McCabe’s complexity metric to assign priority to test cases.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 3. | Geetanjali Chaurasia, Sonali Agarwal and Swarnima Singh Gautam 2015 | A classification of test cases into two groups using $k$-mean clustering. The idea is to focus on effected portions of the program and the related effected test cases and eliminate the need for executing non-effected test cases. The purpose of cluster analysis is to group a set of objects such that the objects clustered in the same group or cluster is more similar to each other than to those in the other groups or clusters. | It shows the test case clustering, when applied to regression testing, having the ability to reduce the cost of regression testing substantially. The contributions of this paper are as follows:  
- Introducing a test case clustering-based approach for reducing the cost of regression testing.  
- Adopting a $k$-means clustering algorithm on the basis of Hamming distances to effectively cluster test cases, i.e. a binary classification. |
| 4. | Thillaikarasi Muthusamy and Seetharaman.K 2013 | Software testing is a process done with an intention to locate the defects in the existing software. It is also the process of evaluation of a software item to detect differences between the given input and the expected output and also to assess the features of a software item. | Testing assesses the quality of the product. Software testing is a process that should be done during as part of development process. In other words, software testing is a verification and validation process of a computer program or application/product to meet the requirements that guided its design and development, works as expected, can be implemented with the same characteristics, and satisfies the needs of stakeholders. |
2.12 APPLYING AGILE METHODS TO EMBEDDED SYSTEMS DEVELOPMENT

During the past decade, that iterative method of software development have gained acceptance, largely displacing older methodologies such as the waterfall or V models of software development. The past few years have seen agile methods, in particular, gain a widespread following. Agile methods indeed symbolize an advancement in the best performs of software development, but the welfares of agile methods differ across the dissimilar systems of software development. Embedded systems is an example of a software development regime in which application of agile methods can be challenging, and the benefits of using agile methods may not be as pronounced as in other regimes of software development[Vincenzo Bonifacil 2010].

In Embedded system Advanced home care technologies and home care systems have not been taken up in people’s homes as eagerly as that might have anticipated. Yet with an increasing ageing population and an increased drive to keep people out of hospital and support people living independently in their own homes, there is a continuing need for well designed, acceptable home care technologies [McGee-Lennon, M. R. 2008].
2.12.1 Brief Description of Software Development Issues

This paper discusses how the distinctive aspects of embedded systems software development disturb the application of agile methods to embedded systems. Though the net outcome of using agile methods in embedded system software development is an enhancement, careful concern is required to achieve the potential benefits.

2.12.1.1 Software Life Cycle

The method of creating, setting out, and supporting software has several properly distinct stages: system specifications, system architecture/design, component specification, design, coding, unit testing, integration, and maintenance. Different software development use different methodologies to address the stages. Some focus on or neglect certain stages. Some proceed serially through the stages, while others allow for overlapping stages or cyclic iteration across stages. Nevertheless, each stage needs to be giving a lecture at least in passing in any software development approach.

2.12.1.2 Software Coding Trade-offs

There are many metrics that can be used for evaluation of the "quality" of software. Many code quality metrics are correlated, but some see the absence of correlation. In particular, performance aspects such as cycles and memory usage are often anti-correlated with code clarity aspects such as readability, testability, modularity, and maintainability. To get good performance, the machine code must be well matched to the particular chip architecture. The source code should be intelligible for human reviewers. In theory, a compiler should have the ability to translate good-clarity source code into good-performance machine code, but in
practice, this translation is too complex to be completely feasible. Ideally, a compiler would generate a highly tuned machine code from a natural language or simple graphical specification, but compilers will not have sufficient artificial intelligence to do this within the foreseeable future.

2.12.2 Brief History of Software Development Methodologies

Moore’s law of exponential improvements in processor capability suggests that the performance/clarity trade-off should be weighted towards clarity since machines continually improve their capacity to run the machine code but humans do not appreciably improve their capacity to read the source code. The optimal performance/clarity balance depends on the software regime. A typical software project team consists of a project manager, analysts, developers, and quality assurance personnel. Often it includes users or their representatives. In the case of commercial off-the-shelf (COTS) software, marketers such as sales representatives and account managers tend to substitute for users and customers [Hofmann, H. F., and Lehner, F. 2001] [Ihme, T. and Abrahamsson, P. 2005].

2.12.2.1 CHAOS

The “chaos” model of software development jumps into coding and neglects requirements, design, and incremental testing. This model was used in the early days of computer programming but works only for very small and simple systems. All subsequent growth models have required improvement of the chaos model through application of the decomposition to make software development of a great system practical. Practical software development relies on decomposing the system into largely independent pieces and building the system implementation by gradually
accumulating working pieces. Thus, applied software development focuses at a high level on the entire system and at a complete level on minor components of the system. This is in contrast to the chaos method’s unsalable focus at a detailed level on the whole system.

2.12.2.2 Functional and Waterfall

Functional programming focuses on the intended behaviour of the software. The gross behaviour of the software is decomposed into functionally cohesive routines. The waterfall model of software development creates a particular serial pass through the life-cycle phases, building all parts of the method at once and accumulating them at the completion. This model is illustrated in the figure 2.1 below.

![Figure 2.1 Basic Waterfall Model](image)

Some versions of the waterfall model allow for iteration between adjacent phases of the life cycle, as is illustrated in the below figure 2.2.
In practice, it is virtually certain that some problems discovered in large system during later stages do not lend to addressing by quick fixes, but rather will require re-architecting or even revisiting requirements. Thus, whether intentionally or as accidental the waterfall model in practice usually ends up being large-scale iterations that involve progressively more of the life-cycle phases. The potential resulting in complexity is seen from the figure 2.3 below.

These paradigms were developed partially in reaction to the problems encountered with the chaos model and were used extensively through the ‘70s and ‘80s.
2.12.3 Object-Oriented and Iterative

Object-oriented programming focuses on the characteristics of components of the software. The iterative model of software development results in repeated serial passing through the life-cycle phases, constructing and incorporating small pieces of the system. Additional definition of agile methods is delivered in an agile development method is incremental (multiple publications), supportive (a strong cooperation between buyer and customer), direct (easy to recognize and modify) and adaptive (allowing for frequent changes).

The use of agile methods in software development has received both supporting and opposing arguments. The main disagreement against agile methods is the declared lack of scientific authentication for connected activities and practices, as well as the effort of integrating plan-based practices with agile ones. Indeed, some projects present a mix of plan-based and agile home ground characteristics, in which case a balance must be achieved in the use of both methods [Boehm, 2002]. There is also some uncertainty in unique agile methods from ad-hoc programming, as stated in agile methods do provide an organized development method.

When trying to match mobile application features to those of an agile method, the struggle comes partially from the fact that limitations of agile methodologies where recognition is not patent or obvious. A comprehensive overview of research in the field is presented the author's partition studies into four categories: introduction and adaptation, human and social factors, the perception of agile methods, and comparative studies. Outcomes indicate that the outline of agile methods to software projects produces welfares, especially if agile practices do not completely replace traditional ones, but work in combination with them, according to the authors,
studies in the field are mostly focused on Extreme Programming (XP), are limited in number and are of doubtful quality.

This researcher provides a direct comparison between agile method characteristics and mobile application features. It focuses on environment volatility, the amount of documentation produced, the amount of planning involved, and the size of the development team, scale of the application in-development, customer identification, and object orientation. Except for client documentation, all other agile characteristics reduce the methods appropriate for mobile application growth. The client may be recognized as the software supplier, particularly in the case of mobile applications, the client identification problem is far more complex.

A new development methodology, specifically tailored for mobile application development, called Mobile-D, is presented in the method is on the basis of agile practices, drawing elements from well-established agile methods such as Extreme Programming and Crystal Methodologies, but also from the “heavier” Rational Unified Process. Additional information on XP is available in [Beck and Andres, 2004], while Crystal Methodologies are thoroughly described in [Cockburn 2004]. The Rational Unified Process is explained from a practical point of view in Practices associated to Mobile-D include test-driven development, pair programming, continuous integration, refactoring, as well as software process improvement tasks.

2.12.4 Embedded System

Embedded systems can be roughly defined as “a system that is not primarily a computer but contains a processor”. But rather than focusing on a definition, it is useful to consider aspects that most embedded systems share, at least to some degree.
Embedded systems are frequently price and size sensitive. The increased up-front software development costs and periodic maintenance costs are amortized by the high-volume sales and outweighed by the continuous hardware cost savings of cheaper components.

Many other embedded systems, though not highly price-sensitive, have physical constraints on the form factor or weight in using the smallest components possible. Embedded systems often have power limitations.

Embedded systems are frequently real-time. The real-time constraints again favour performance aspects (particularly cycle’s usage) over maintainability aspects. There are generally both hard real-time constraints, which require an event to be handled by a fixed time, and soft real-time constraints, which set limits both on the average event response time and the permissible magnitude of outliers.

Real-time operating systems use pre-emptive prioritized scheduling to help ensure meeting real-time deadlines, but careful thought is required in dividing processing into execution contexts (threads), set the relative priorities of the execution contexts, and manage control/data flow between the contexts. Embedded systems frequently use custom hardware. Even though the components may be standard, the custom of mixing and matching require a high degree of cohesion between the hardware and the software -- a significant portion of the software for an embedded system is operating system and device driver software. Often the functionality of an embedded system is distributed among multiple peer processors and/or a hierarchy of master/slave processors.

Embedded systems have monolithic functionality in many cases. They can be decomposed into components, and potentially the components could have low cross-
cohesion and cross-coupling. That is, each component could serve a distinct purpose, while the interactions between components could be restricted to a few well-defined points. Nevertheless, the system as a whole will not function unless most or all of the components are operational. A system that requires all components to function before the system as a whole achieves useful functionality is a "monolithic system".

The space probe will be useless when any of these vital components is missing, even if all other components are completely functional. Another example is a cell phone, in which all the sub-features such as the user interface, the cellular base station selection, the vocoder, and the communications protocols are all vital aspects of the over-arching goal to transfer bi-directional audio information between the user and specific remote nodes. Though the software components of an embedded system are combined into a monolithic functionality, the components themselves are often very distinct. Each of these specialized components requires a distinct developer skill set.

Embedded systems have limited development tools in many instances. Considering of the limited choices of commercial tools for embedded systems software development, many embedded systems projects create their own tools for use in debugging and testing or at least augment commercial tools with in-house tools.

Embedded systems have stringent robustness requirements in many cases. Therefore, requirements for reliability, correct exception handling, and mean time between failures are typically more stringent for embedded systems than for many other types of software. This is translated into rigorous development processes and testing requirements. In turn, this increases the overhead needed to make a release of the software. Furthermore, for several types of embedded systems, it is difficult or even impossible to upgrade firmware, which emphasizes the need to “get it right” in the system’s initial commercial release.
2.13 TEST-DRIVEN DEVELOPMENT OF GRAPHICAL USER INTERFACES

Test-Driven Development (TDD) encourages communication between customers and developers, increases programmer confidence, increases software quality, and (arguably) decreases bug density without decreasing productivity [Doug Dahlby. 2004] [Johnson, M. J.et al.. 2007]. As maximum published approaches to test the software layer just beneath the Graphical User Interface (GUI), test-driven development of Graphical User Interface (GUI), rests an unsolved problem. The reason for this is twofold. First, GUIs are very likely to change repeatedly over the course of development. This means that tests will need to be updated and repaired frequently-which is a nontrivial task. Second, the easiest method of creating Graphical User Interface (GUI), tests – using a capture-replay tool (CRT) – requires that a Graphical User Interface (GUI) exists before tests can be defined. While there are further issues that create testing Graphical User Interface (GUI)s difficult, these two extant the greatest tasks to user interface test-driven development (UITDD). The approach presented in this paper combines a low fidelity prototyping tool with a Capture-Replay Tools (CRT). First, details of user stories are collected. Second, these stories are used to create a low-fidelity prototype of the system. Hard, usability evaluations of the prototype are conducted to identify and fix flaws that would require changes to the interface. These steps are repeated until the prototype is sufficiently stable.

Then, the prototype can be augmented with additional information about the expected behavior of the Graphical User Interface (GUI). This allows for complex acceptance tests to be recorded using a Capture-Replay Tools (CRT). The resulting
tests can then be run on the Graphical User Interface (GUI), while it is being developed to ensure that it matches the user expectations expressed in the prototype.

In short, uses of an adequately complete prototype for agile interaction design begets gather two items of welfare. First, usability apprehensions get early exposure in development, meaning the last Graphical User Interface (GUI), will be less likely to require changes. Second, if the prototype is adorned with automation data – statistics that can be used for classifying and making declarations about widgets and if this data is maintained in the actual application, then tests can be documented from the prototype and replayed on the actual Graphical User Interface (GUI), as it is implemented.

2.13.1 Capture-Replay Tools

Capture-Replay Tools (CRT) work by copy communications with a Graphical User Interface (GUI) and keeping them as a sequence of actions that can be repeated on that Graphical User Interface (GUI). The fundamental difficulty with this is that methods in widgets can’t be accessed easily by test code. Capture-Replay Tools (CRT) evaded this problem entirely by basically recording keyboard input and the screen position of mouse clicks. A test script based on this would simply replay these actions. This sort of testing was useful for detecting crashes, but verifying correct system behavior was another matter. Relying on screen coordinates also has the distinct disadvantage of being very sensitive to non-semantic changes to the Graphical User Interface (GUI) under test. Rearranging widgets, for example, would cause test failures even though the application was functioning properly.
The generation that followed Capture-Replay Tools (CRT) uses a technique called testing with object maps, which works by keeping as much data as possible about a widget so that an uncertain match can be finished when the test is run. This makes tests more robust against changes, and allows access to widgets by the test to enable testing of their behavior. While testing with object maps is more robust and useful than testing with direct input, it is still difficult to code due to the amount of information that must be known about a widget in order to correctly locate it. Keyword-based testing is a Graphical User Interface (GUI) testing technique that has been developed relatively recently. Rather than storing substantial data about a widget, this system simply assigns a unique keyword to each widget, this means that only a keyword is required to locate and interact with a specific widget from within a test. Keyword-based testing is a robust, easy way to write Graphical User Interface (GUI) tests, and is now possible through most Capture-Replay Tools (CRT).

2.13.2 User Interface Test Driven Development

In recent years, several tools have been developed in support of user interface test-driven development (UITDD). The support uses from their simplification of manual Graphical User Interface (GUI) test authoring by providing framework support that makes identification of and interaction with widgets simpler and more robust. Some provide added robustness by storing tests in an intermediate form along with an intermediate representation of widgets used in testing, which helps test maintenance. While these tools can reduce the effort involved in user interface test-driven development (UITDD) there is still an imperative need for manual coding. Writing Graphical User Interface (GUI) test scripts by an indicator can be a dull,
error-prone task, while an agile team using this method in the past found that tests inscribed for user interface test-driven development (UITDD) tend to need alteration before they can even pass for the major time after the corresponding Graphical User Interface (GUI) code is printed. This team found it faster to rerecord tests using a capture-replay tool than to attempt to repair the initial target Graphical User Interface (GUI) test.

2.13.3 Support for Test Script Maintenance

Tool support for test maintenance has also been a subject of recent research. Work takes a compiler-inspired approach by attempting replacement of events in a broken test automatically in an attempt to create a legal sequence of test steps without the need for human interaction.

Information to Graphical User Interface (GUI) test scripts, simplifying manual maintenance. REST, on the other hand, makes a connection between widgets in an application’s code and their use in tests and is able to make suggestions as to where and why a test script is likely to fail. Actionable Knowledge Models store tests in an intermediate model, which allows the root cause of a test failure to be addressed in a single location rather than propagated between individual test scripts manually.

2.14 COMMUNICATION-AWARE DIRECTED ACYCLIC GRAPHS

Cloud computing is a rising theory for providing services and solving significant problems in skill, work, and business. The main challenges of cloud computing is to make the service available and to progress multiple virtual machines on various systems which for getting resolved to the first degree [Theodore D. Hellmann, et al 2010]. Consequently, scholars can now deal with the problems that
are well-ordered by the use of the resources. The usage of cloud resource management is distant from all over the place. This is appropriate to the point that preparation and mapping decisions have to make way for the myriads of principles, processes, and strategies in an extremely dynamic situation. As a consequence, resource management proceedings should have the ability to adjust to changes in data communication requirements to come across their desired Quality Of Services (QoS) constraints as traditional methods to resource optimization that turn out to be insufficient. Planning of jobs on multiprocessors is usually well agreed and has been done for years. Many results of inquiries do exist; some of them deliver theoretical perceptions while others provide suggestions for practical implementation. The communication-aware scheduling problems that require an ease of use of communication resources are infrequently addressed. The communication properties are either completely ignored or highly generalized and weakly captured by current task models and scheduling approaches, resulting in ineffective cloud infrastructure and communication media application.

In traditional scheduling, the communication system demonstrated itself either as a homogeneous wholly connected network, which takes up constant communication postponements, or heterogeneous, where postponements inside a cluster of processors are lesser than that between clusters. Moreover, communication delays may be totally neglected as, for example, when a predecessor task and a successor task are executed on the same processor. This is known as the locality assumption. The essential property of such models is that task duplication can avoid communication delays. In detail, there are only very few scheduling solutions accessible that take into interpretation large communication delays. The most widely
used approaches to balance communication delays and processing times are task clustering, using critical path analysis or decomposition of the precedence task graph. Cloud applications and services can be represented with workflows, defined as a composition of tasks with precedence constraints, and modelled by Directed Acyclic Graph (DAG)s.

The vertices of a Directed Acyclic Graph (DAG) represent the volume of computing job that requires processing for successful execution of a task, while the edges define precedence constraints. Such a workflow model works well for HPC application but fails in the cloud where announcement processes often become a restricted access. Several researchers have got used to the standard Directed Acyclic Graph (DAG) model by either permitting vertices to symbolize both computing and communication requirements of a task or by correlating edges with the interactions completed by the tasks.

The primary model fails to make a difference between the computing and communication jobs of a mission preventing their proper scheduling on fundamentally different resources: processors and communication network. The future method demonstrating communication work with edges does not allow a particular communication process drive before two computing tasks as a single edge cannot lead to two dissimilar vertices in a Directed Acyclic Graph (DAG).

It defines the communication-aware model of cloud applications, called Communication Aware Directed Acyclic Graph (CA-DAG). It allows making separate resource allocation decisions, assigning processors to handle computing jobs and network resources for information transmissions, such as application database requests. It is based on Directed Acyclic Graph (DAG)s that in addition to computing
vertices include separate vertices to represent communications. The future communication-aware model produces space for optimization of many current results to resource allocation as well as emerging completely new scheduling schemes of developed efficiency.

2.15 RISK FACTOR BASED USER CASE MODEL

In the software development industry, the culture of 'outsourcing' has become highly popular availing economic factors and skills. In terms of this culture, companies have been decentralizing then developing or testing units through contracts or outsourcing processes into different zones [Dzmitry Kliazovich, et al 2013]. Test estimation is really important for software project planning and software bidding to win a pro, etc., in the software industry.

Proper execution and estimation are important in software development life cycle. With an example in agile scrum software development, the team needs to estimate the test. Normally team does it manually by an open consultation. And then the scrum master will need to measure test effort by a standard process and find out the estimated effort from the team really feasible or not. At present software, outsourcing is a really popular approach in software development industry. So when stakeholder decides to build any software product then normally ask the industrial companies to bid for the pro, etc.,

Besides there is multi-level outsourcing. Such an example, one American stakeholder outsource pro etc to a renowned company at Europe and then that company outsources the pro etc to a medium size company in Southeast Asia. So for the multi levels outsourcing the bidding become more and more important. In
numerous software establishments, people make use of no standardization but
conservative estimation methods to make stuffs effort.

The set of stakeholders in the knowledge acquisition process and the set of
stakeholders in the use of a knowledge-based system, are not necessarily identical
[Sharp, H. Finkelstein, and A. Galal, G. 1999]; they are likely to vary in membership,
and for those members in common, the type and level of stake they have is likely to
vary At present software companies mainly follow the following test estimation
models estimate the test:

- FIA (finger in the air) or best guess
- Use case point estimation method
- Functional point estimation method
- Ad-hoc method
- Delphi technique
- Experience Based - Analogies and experts
- Three-point estimation (successive calculation)
- Function points f Test point Analysis
- Percentage of development effort method
- Percentage distribution

2.16 AN AGILE SECURITY TESTING FRAMEWORK FOR WEB-APPLICATION
DEVELOPMENT

Vulnerable Web applications as are fair games to for malicious users who
conduct cyber-attacks. Secure coding still needs to be put into practice by
programmers themselves even if the latest secure Web application framework is
availed. Coding bugs during implementation lead to serious vulnerabilities. In
addition, security problems arise in both the implementation and design of applications. Incomplete or improper placement of security features is a design error [Abu Wahid Md and. Masud Parvez 2013].

There is a need for comprehensive and automated security testing to find and cope with unavoidable human errors. The existing security testing methods can be classified into two approaches: static and dynamic. Static analysis directly checks the application source code to detect adverse coding styles that cause vulnerabilities. There has been little research and few tools for Web applications written in a scripting language. Security scanners for Web applications are dynamic checking tools that communicate with the target application and identify potential vulnerabilities. These security testing approaches have a tendency to become specialized tools. Consequently, security testing becomes isolated in the development process.

Moreover, both approaches cannot address the problem of design errors, such as implementing improper access control. Sun. Demonstrated a new method that generates a sitemap par role from the source code of Web application to detect access-control vulnerabilities.

Model-driven development seeks to improve the security and security testing practices of Web applications; such top-down approaches are not yet well used within the existing Web application frameworks because of the limited capability of code-generation from the model [Abrahamsson, P. et al. 2004]. Also, it is incompatible with the code-centric development practice. The verification is done in the following two stages. First, security features are comprehensively verified with the navigation model. Second, typical test-cases are generated from the model to verify individual security features. In conjunction with code-centric web application development, our framework can play a key role in coordinated security management.
2.16.1 Security Assessment Framework

A tool called “Railroad Map” that generates a behaviour model from the application source code in Ruby- Rails (shortened to Rails, http://rubyonrails.org), which is a popular Model-View-Controller (MVC) Web application framework. This contains the operational flow of the proposed framework, which consists of four steps: Step 1) parse the application code and create a navigation model, Step 2) assess the security design and requirements with the navigation model, Step 3) generate an abused model by adding the behaviours of the security features, and attack vectors to the navigation model, and Step 4) generate the minimum amounts of test cases.

Generation of navigation and abuse model using modelling Web applications as state machines have been proposed in earlier works. The “navigation model” is referred to as a state machine that expresses Web application behaviour of intended and legitimate uses. Therefore, it simply abstracts the main application code that is maintained by the programmer. The “abuse model” extends the navigation model by adding behaviours against improper or malicious access. Thus, the abuse model covers the exhaustive application behaviours, which includes all of the security, security design, and implementation required by a secure coding guide.

2.17 DESIGNING AN AGILE METHODOLOGY FOR MOBILE SOFTWARE DEVELOPMENT

The new advances in mobile computing technologies, that is too fast in emergence for the software engineering field to keep up with. There is a universal growth in number of mobile devices with computation capabilities incorporated (such as third-generation mobile phones and personal digital assistants), and the ever-
increasing demand for specialized software for these devices has caused new concerns for software developers, as this type of software has its own unique characteristics and requirements [Seiji Munetoh and Nobukazu Yoshioka 2013][ Nascimento, L. et al. 2008]. Although commercial mobile systems have not been as successful as originally predicted, mobile operators and mobile value-added software providers expect that the deployment of 3G technologies would have a dramatic effect on the mobile applications industry [ Abrahamsson, P. 2007].

There are numerous challenges that the designer of a software system for mobile environments has to come up with. These challenges mainly stem from

- **Wireless communication issues** (considerations such as availability and disconnection, bandwidth variability, heterogeneous networks, and security risks)
- **Mobility issues** (concerns such as address migration, and management of location-dependent information)
- **Portability issues**
- **Various standards, protocols and network technologies**
- **Limited capabilities of terminal devices** (factors pertaining to low power, risks to data integrity, small sized user interfaces, and low storage capacities)
- **Special privacy and customizability needs**
- **Strict time-to-market requirements.**

Some of these issues are due to deficiencies in current technology, but most of them are intrinsic to mobility. The design of mobile software systems is therefore much more complicated than usually seen in software development projects, thus forcing developers to reconsider the use of traditional software development methodologies [Harrold, M.J. and Rothermel, G. 2007]. Despite the above-mentioned
problems, research endeavours aim at ameliorating the status quo through enhancing/devising methodologies for mobile-software systems development have been relatively few and far in between. Most of the work performed in this field has been focused on low-level (implementation-oriented) aspects of software development, while high-level (methodology oriented) issues still remain to be properly addressed.

To identifying mobile computing challenges, adaptation of mobile software, the notion of mobile agents, resource sharing in mobile environments, abstract architectures with special quality features, and design patterns for developing mobile software are examples of the more popular, lower-level research so far performed on mobile software development, whereas research on specialized development methodologies is fairly limited [Jones, L. G. and Northrop, L. M. 2010].

New Product Development (NPD) is a complete process for bringing a new product or service to the market. There are two parallel paths followed in the New Product Development (NPD) process: one involves idea generation, product design, and detailed engineering; the other involves market research and analysis. They also analyzed the activities performed in two commonly-used methodologies and have compared them with two New Product Development (NPD) process models. They have concluded that current process models should contain more market-oriented activities in order to be efficiently incorporated into the development of mobile software products.[ Staples, M. Hill, D. 2004] describes along the years companies have placed a great deal of effort to produce software artifacts that can be reused in other projects.
Another major work in the field of mobile software development is that in
which a methodology called Mobile-D is proposed as an agile approach to mobile
application development. [Zhang, W. 2005] illustrate the approach is based on
development practices borrowed from XP (eXtreme Programming), enjoys method
scalability inspired by the Crystal family of methodologies, and provides life-cycle
coverage as prescribed by RUP (Rational Unified Process) [Oinas-Kukkonen, H. and
Kurkela, V. 2003].

The framework presented in [Unhelkar and Murugesan, 2010] could provide a
useful architectural model for the development team. The information could be taken
into consideration in the Explore phase, Project establishment stage, when performing
the task called Architecture Line Definition. [Zhang, W. and Hansen, K. M. 2007]
describes according to Mobile-D specifications, the aim of the task is to gain
confidence in the architectural approach, in order for the project to be successfully
carried out. [Varshney, U. and Vetter, R. 2001] describes Mobile-D, through its
phases and disciplines, tries to merge the classic software development process (i.e.
traditional plan, design, implement, test, and release activities, as mapped to Mobile-
D disciplines of Phasing, Architecture Line, Test-Driven Development, Continuous
Integration, Pair Programming, and Off-Site Customer) with the necessary umbrella
management/support processes (i.e. project management, software configuration
management, and software process improvement, as mapped to Mobile-D disciplines
of Metrics, Agile Software Process Improvement, and User-Centred Focus). Although
the work of mobile software development seems to be very promising, the description
that they provide of their Mobile-D approach is cursory and incomplete.
2.17.1 Characteristics of an Ideal Mobile Software Development Methodology

The traits that believe a development method should have in order to be efficiently employed for mobile software development. Based on the properties, identified a new methodology using the Hybrid Methodology Design approach.

2.17.1.1 Agility

As stated in the previous section, agile methods seem to be a good starting point for constructing a mobile software development method. Agile methodologies are believed to enhance software development flexibility and productivity, by providing means to adapt to changes in requirements and the environment, and also to learn from development experiences. To support early and quick delivery of working software, these methodologies use iterative-incremental development engines to produce artifacts tangible to the customer.

Agile characteristics of highest importance in the context of mobile software development include iterative and incremental process (which leads to enhanced risk management capabilities), test-driven development, adaptive process, continuous customer involvement, highly skilled developers, enhanced quality assurance, and continuous process-wide reviews [Abrahamsson, P. 2005]. Furthermore, considering the competitive market for mobile software, shorter time-to market is a precious advantage; this could be achieved via early releases of operational software, which is itself an important feature of agile methods. Prioritization of requirements is another agile practice that can prove essential in mobile software development, since it sets the stage for and governs risk management activities, and helps ensure that features of higher value to the customer take precedence.
2.17.1.2 Market Consciousness

As the current market for mobile software is biased towards relatively fine-grained software products, a general mobile development process should be oriented mainly towards product development, rather than project development. Consequently, such processes should focus on establishing the business case, thereby striving to identify the potential market.

The use of New Product Development (NPD) practices for market analysis can enhance mobile-software development: New Product Development (NPD) process activities utilize market information for mitigating uncertainties and risks. In a market-oriented process for mobile software development, market and customer needs should be carefully analyzed, and a strict release schedule, which meets time-to-market requirements, should be established and maintained during development. In contrast to the typical modus operandi, in which a process’s main focus is on technical activities, a mobile software development process should maintain a balance between market-oriented and technical activities.

2.17.1.3 Software Product Line Support

A software product line is “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”. The special benefits of applying the product line approach to mobile software development are mainly due to the fast pace of advances in mobile technology. This makes the life-cycle of mobile software products even shorter. Consequently, software companies tend to develop a family of mobile software
products in a bid to reduce development costs. As a result, we suggest that a mobile software development process should provide means for supporting product line engineering. Component-based development, use of reconfigurable architectures, and product scoping can help in achieving this capability. The process should also provide adequate guidance on performing quality product line design.

2.17.1.4 Architecture-Based Development

The efficiency of the software product line approach depends on the development firm’s ability to invest in the development of a common platform. This necessitates the development of a general architecture for such products.

2.17.1.5 Support for Reusability

The need for using functionally equivalent components (such as message boxes in a specific language), and functionally similar components (such as security operation classes), along with tight chronological constraints on the development of mobile software, requires the developer to use reusable components extensively. The intense need to develop these components from scratch each time results in increase in the cost of mobile software development, delayed product delivery, makes the software error prone. Support for component based and layer-based development approaches is, therefore, essential in a mobile software development methodology.

2.17.1.6 Inclusion of Review and Learning Sessions

One of the most effective factors in a production company’s success is its ability to abstract the knowledge obtained during product development. As mentioned above, today’s mobile software development industry tends to be product-oriented;
the methodology should, therefore, incorporate review sessions throughout the process to ensure product analysis, and include “lessons-learnt” sessions after delivering a working product to the market to ensure that experiences are analysed and logged.

**2.17.1.7 Early Specification of Physical Architecture**

Constraints seen in Mobile terminals’ should be looked into even from very early stages of software design. In fact, a high degree of technical risk can be traced to mobile hosts’ limitations and their differences in the implementation of basic features. Consequently, the physical architecture should be elaborated in the early stages of software development. A prototype may also be required in order to mitigate technical risk.

**2.18 REAL-TIME SCHEDULING OF PARALLEL TASKS UNDER DAG MODEL**

The rate of enhancement of clock frequencies are most processor chip manufacturers contain lately stimulated to increase performance of processors by raising the number of cores on each chip. With the quick development of multi-core processor skillfulness, real-time system software and programming models are ineffective in keeping pace. In scrupulous, most classic results in real time, scheduling focuses on chronological tasks running on numerous processors or cores [Vahid Rahimian and Raman Ramsin, 2006].

A program that uses fork-join constructs produces a non-synchronous task, usually represented as a Directed Acyclic Graph (DAG), where each thread (series of instructions) is a node and edges represent dependency between threads. For the unit-
node Directed Acyclic Graph (DAG) model, the scheduler first converts every task to a synchronous task and next applies the investigation followed by a synchronous model.

The nodes of a general Directed Acyclic Graph (DAG) are represented as sequential tasks. This breakdown does not transfer non-synchronous responsibilities to synchronous responsibilities and, therefore, contrasting that in, it does not require splitting threads into shorter threads. Hence, our proposed decomposition allows preemptive scheduling, where threads (nodes of the Directed Acyclic Graph (DAG)) are never pre-empted. Prove that parallel tasks in the general Directed Acyclic Graph (DAG) model, upon decomposition, can be listed using pre-emptive global Earliest Deadline First (EDF) with a resource intensification bound of 4. This bound is as good as the most excellent known bound for more restricted models and, to our information, is the first for a common Directed Acyclic Graph (DAG) model.

2.18.1 Parallel Task Model

This model considers n sporadic parallel tasks to be listed on a multi-core stage consisting of m the same cores. A node may have 0 or more parents or children. A node can start execution only after all of its parents have finished execution. Figure 2.4 shows a task _i with ni = 10 nodes.2 The similar node in succeeding figures will have the same shade W I 1Wi4Wi10Wi2Wi7Wi8 Wi9Wi3Wi5Wi6.
Fig. 2.4 A parallel task represented as a Directed Acyclic Graph (DAG)

The total implementation requirement of _i is the sum of the implementation needs of all of its nodes and is denote by Ci (time units). The era of task _i is denoted by Ti. The aim Di of each task _i is measured implicit.

2.19 AGILE METHODS FOR EMBEDDED SYSTEMS

Embedded systems are pervasive in the world we live in today, the extent of which is illustrated by Egger Mont, who notes that the average home today has over 50 embedded systems [Saifullah Abusayeed. et al. 2012]. Organizations involved in the lifecycle (design, development, and sustainment) of these systems, have to manage the growing operational and environmental complexity that these systems face. Increased competitive pressures are driving compressed schedules to enable the organization to meet time-to-market goals, while, at the same time, teams face daunting budgetary challenges. Surviving, in this context, is no longer sufficient, and organizations must fundamentally change the way they do systems developed to thrive and grow. Determination of the state of the practice in embedded software engineering one of the key findings in the study was that systems engineering decisions are largely being driven by hardware constraints, which then impact software efforts two stages later in the lifecycle when software requirements at the
component level are developed [Khambati, A. et al. 2008]. This development partitioning into two independent streams of hardware and software development occurring in parallel, resulted in the integration becoming the critical point of failure, the nexus in the classical ‘V’ model of software development. It is important to note that, in the context of embedded systems development, there are multiple instances of software development that need to be synchronized with hardware development, at the latest at integration testing.

The designer has to a trade-off between flexibility and efficiency in selecting the right hardware platform. When the choice of a platform is made at the system architecture level, the impacts on software are not really known, and, more importantly, may lead to significant changes in requirements when the developed software does not meet the performance requirements. The development of software for embedded systems is difficult, because these systems are part of a physical environment whose complex dynamics timing and requirements have to be satisfied. Furthermore, these projects often lack an effective software development methodology.

Agile development has three elements of the ecosystem that constitute agile methods: viz., barely sufficient methodology, collaborative values, and chaotic perspective. Going further to discuss the class of problems that are best suited for agile methods, he notes that the more volatile the requirements, and the more experimental the technology, the more agile approaches improve the odds of success. Considering that most embedded systems development projects (including those with a co-design flavour) have those characteristics, it begs the question of why agile methods are not the development approach of choice.
2.19.1 Effective Systems Development

The development of embedded systems as hardware related’ software development identifies four problem areas that need to be addressed, viz., being: meeting hard real-time requirements; supporting safe experimentation; generating sufficient documentation; and supporting test driven development. Analysing the discussion topics at the workshop on agile development for embedded software, see emergent themes are seen as the areas of domain characteristics, tool support, verification and validation, and project management. The perspective of developing mobile software shows the Mobile-D approach as effective. In discussion of the challenges seen in creation of a hybrid approach for print engine development notes that development effort really did not increase, the lack of improvement mainly to learning curve effects, and also found that using a visual language improved understand ability at peer-review sessions.

Report on their year-long experience at Atomic Object in developing control software for autonomous warehouse vehicles. The two case examples of implementing speed control capability and monitoring battery charge level, demonstrate not only the challenges that they faced but also provide lessons learned. The major pain points in the first project, besides manually generating mock objects were the lack of transparency into compiler assumptions; poor simulator capability; and lack of automation for the system and regression testing.

Here addressed some of the pain points by automating system testing in the second project by creating a more effective test rig, and equally important, a trustable, more powerful tool chain (including support for automatically generating mocks). Similarly, using component-based development (for example) or service-oriented
architecture does not preclude the use of agile methods – for instance, instead of stories; the development unit can be effectively measured as components or services.

2.19.2 Analysis of the Literature

Manhart and Schneider’s use of agile methods at DaimlerChrysler was motivated by the fact that 58% rejection of all functions at least once, with six potential rejection paths leading to rework. Their challenges revolved around automated testing and changing the developer mindset from develop-document-test to test-develop-document. In an orthogonal view, Schatz shows approaches such as model-based development (which is becoming increasingly more popular for embedded systems development) are orthogonal to the software development process used, be it agile or plan-based.

2.19.2.1 Requirements Management

Agile approaches are best suited for environments in which requirements are in the process of evolving. When software development is outsourced (which is often the case when software is a part of a larger system such as an aircraft or an automobile), there is tension between innovation-driven value created by the designer (on the client side), and the value added through implementation (carried out by the supplier). When the client is not well versed in software specification, novel requirements gathering approaches are needed to ensure accurate capture of the voice of the customer. The use of goal-oriented brainstorming and workshops as potential strategies support agile requirements gathering. On visual specifications to support human computer.
Interaction adds to provide interesting insights into using hi-fidelity prototypes as partial replacements for solely textual specifications, leading to a greater emphasis on requirements evaluation rather than just generation. Whether the organization decides to use the actual construct of stories or not, it is important to note that the right abstraction mechanism makes requirements gathering and analysis easier.

2.19.2.2 Test-Driven Development

Testing of embedded software is bound up with the testing of hardware that crosses professional and organizational boundaries. On reflecting on three years of experience in the project, they note that only 50 bugs made it undetected to integration testing, and at any given point in time, the list of open bugs never exceeded 2. They attribute their success to having multiple testing strategies that ranged from unit testing to domain testing.

An important enabler has the ability to use approaches such as mock objects, and testing frameworks. An approach for scenario-based testing that is compatible with formal analysis techniques such as model checking, and also for reusing test script templates. For embedded applications that have safety critical implications, such approaches can provide the necessary additional assurance needed.

The challenge for any organization that is attempting use of agile methods is finding the right tools and techniques to support test-driven development without compromising on the efficiencies already gained with home grown tool suites. Erickson’s guidance provides the first step towards creating more generalizable guidance.
2.19.2.3 Process Tailoring

Boehm lays out the planning spectrum ranging from ‘undisciplined hacking’, at one end, to the heavyweight ‘inch-pebble ironbound contract’ at the other, with milestone risk-driven approaches such as spiral development in the middle. In analyzing the risk exposure as a function of the probability of loss and the size of the loss, he notes that the sweet spot is a function of the ‘home ground’ of the development approach. From the perspective of adopting agile, this essentially translates into the organization assessing the needs of the project, and matching the project needs with the organizational culture to find the best tailoring.

This notion of requisite tailoring is implicitly discussed in the study of agile adoption in a large telecom company analyse the family of agile methods notes that while universal solutions dominate the literature, more work is needed to support situation appropriate adoption.

2.19.2.4 Knowledge Sharing and Transfer

Agile development is about people, as Cockburn and Highsmith note. The approach is based on the idea that the development team is more effective when responding to change if it can reduce the cost of moving information between people, and if the time elapsed between making a decision and seeing the consequences of the decision is minimized. The uses of collocation, brainstorming facilities are present in the development of embedded systems through the notion of integrated product teams. This construct of creating organizational environments is referred to by Nonaka, Toyama, and Konno. The shared team mentality (since work is partitioned into hardware and software and often executed independently – this also has the negative
impact of the teams not know what their customers want) is missing, as is the construct of working incrementally. Agile development relies heavily on socialization through communication and collaboration to access and share tacit knowledge within the team.

Embedded software project knowledge contexts for both inputs and outputs, to either share knowledge using planning templates or through the use of a sharing chart. While creating communities of practice, such as those in Nokia, are a promising approach, we recommend that organizations build on their current collaboration and knowledge management strategies.

2.19.2.5 Developing Supporting Infrastructure

As has been noted earlier, transitioning to agile methods is not easy. It is critical to train both the people involved in the process, and also the senior leadership that manages the process. The concepts and practices of agile methods are simple to grasp but very difficult to institutionalize without a proper support structure. As Lundvall noted the greatest challenges associated with agile adoption are integrating them with the existing environment.

In addition to supporting tailoring, the supporting infrastructure has to support cross-team communications, harmonize existing organizational monuments such as a change control board with practices such as refactoring, and put in an effective governance system. The governance structure doesn’t necessarily mean ‘burning your Gantt charts’, but rather about finding a mechanism that best supports both the project team and the project manager.
2.19.2.6 Managing Culture Change

One of the elements critical to successful iterative development is the active involvement of the customer. As Hirsch pointed out, a commitment issue may lead to project failure. From the stakeholder perspective, it is essential to note that the software team is both a customer of (since they provide feature requests etc to) and a supplier to (providing working software) the hardware team. When organizations work in a distributed fashion, the problem of managing commitments and risks get exacerbated.

2.20 MODEL-BASED VERIFICATION AND VALIDATION DIRECTED ACYCLIC GRAPH MODELING

Firewalls act as the most important defense mechanism of network security, have to be tested to validate their work as specified. The firewall specification is mainly composed of intended security policy and allowed network protocols, which are usually the main focus of an attacker [Jayakanth Srinivasan, et al 2007]. The intended security policy consists of firewall rules, which configure the firewall behavior, and allowed network protocols. These constitute an important part of firewall’s internal infrastructure which can be described as packet capture, decision making on the packet under consideration and packet release.

Decision-making operation is carried out with respect to firewall policy and network protocols. Security policy is external to the firewall like a configuration file, whereas packet checking with respect to network protocols is implemented in the firewall software. The firewall policy is considered as a specification and can be represented by a formal model, the researcher proposes a model-based testing approach for firewalls [Balagtas-Fernandez, and F. Hussmann, H. 2008].
The novelty of this approach is using Directed Acyclic Graph (DAG) model for firewall testing. This paper proposes a modeling of firewall rules and generating test cases using Directed Acyclic Graph (DAG) s. Since Event Sequence Graphs (ESG) are directed graphs, we applied its test case generation algorithm to the Directed Acyclic Graph (DAG) representation of firewall rules.

2.20.1 Directed Acyclic Graphs

Directed Acyclic Graph (DAG) is a directed graph with no directed circuits. For any vertex v, there is no nonempty directed path that starts and ends on v. The simplest example of a DAG can be given as a directed tree. The vertices of an n-vertex acyclic directed graph G can be labelled with integers from the set \{1, 2, n\} such that the presence of the edge (i,j) in G implies that i<j where the edge (i,j) is directed from vertex I to vertex j. A partial order is formed by the reach ability relation in a Directed Acyclic Graph (DAG). Directed Acyclic Graph (DAG)s are mainly used to model processes where the flow of information moves in a consistent direction. In the researcher’s case, Directed Acyclic Graph (DAG) is used for representing the firewall policy rules as a rule (evaluation) sequence graph, from which test cases are generated using test case generation algorithm developed for event sequence graphs.

2.20.2 Event Sequence Graphs

The testing process consists of the execution of the System under Test (SUT) with the produced test inputs and the comparison of the real outputs with the expected ones. If the outputs are in compliance with the expected ones, the test is said to have succeeded, else it fails. Even the sequence graph is an event-based formal model,
where the inputs and events are merged and assigned to the vertices of an event transition diagram. The arcs visualize the sequence relation of the events. An Event Sequence Graphs (ESG) is a simple albeit powerful formalism for capturing the behavior of interactive systems. The complete set of interactions is captured in terms of a set of Event Sequence Graphs (ESG) s, where each Event Sequence Graphs (ESG) represents a possibly infinite set of event sequences. An event can be a user stimulus or a system response, punctuating different stages of the system activity. As stated in, each edge in the Event Sequence Graphs (ESG) is marked as a legal event pair (EP).

A complete event sequence (CES) represents a walk through the Event Sequence Graphs (ESG) by starting at the entry node and ending at the exit node of the Event Sequence Graphs (ESG). Entry and exit nodes are not events; they represent only entry and exit points of an Event Sequence Graphs (ESG). Faulty (or illegal) event pairs (FEP) are introduced as the edges of the corresponding Event Sequence Graphs (ESG). Moreover, an Event Pair (EP) of the Event Sequence Graphs (ESG) can be extended to a faulty, or an illegal, event triple by adding a subsequent Faulty (or illegal) Event Pairs (FEP) to this Event Pair (EP).

2.20.3 Test Case Generation

The method described which uses an Event Sequence Graphs (ESG) and its complement as input and generates a test set is used. It is complete with respect to model-based coverage criterion. There are mainly two objectives for the test case generation procedure. One is the generation of complete event sequence (CES) s and the other is the generation of FCESs from the complement of ESG that model the
system behavior by considering both the desirable and undesirable parts. With the input of an FCES, the Function under Test (FUT) is expected to go to a faulty state and raise a related exception handling mechanism. Hence, complete event sequences (CES) are used for testing the correct behavior, where the complete event sequences (CES) are used for checking the exception handling mechanism.

Given an Event Sequence Graphs (ESG) and the corresponding complete event sequence (CES), the test case generation algorithm generates tests that cover both all event pairs in Event Sequence Graphs (ESG) and all faulty event pairs of the complete event sequence (CES). A point to note is that the sum of the lengths of the generated complete event sequence (CES) s should be minimal to avoid the long chain of events.

2.20.4 Firewall Testing Approach

The firewall testing approach proposed here is composed of five phases: (1) generating test cases from firewall policy rules, (2) constructing network test packets from generated test cases, (3) sending constructed test packets to Function Under Test (FUT), (4) capturing packets that go through the Function Under Test (FUT), and (5) comparing sent and captured packets to determine the test result.

The firewall policy parser algorithm explained here is used for conversion of rules to a Directed Acyclic Graph (DAG) and then each rule sequence in the Directed Acyclic Graph (DAG) is considered as an event sequence, to enable utilization of the test case generation algorithm for Event Sequence Graphs (ESG). At the test case generation step, the algorithm creates a test case for each complete event (rule) sequence, which is derived from the Directed Acyclic Graph (DAG).
The test case generation algorithm works as follows: for each complete event sequence, a test case is generated from the first rule of that complete event sequence (CES) and this test case is modified by the proceeding events (rules) in the complete event sequence (CES) until the “deny all” rule, which is always the last rule for all firewall policies that follows default deny principle. Once concrete test cases are ready, construction of network test packets as well as sending and their capture require appropriate network architecture and some network programming. The firewall testing tool proposed by the researcher contains the necessary network programming code, to have the ability to analyze and evaluate the behavior of the firewall under test with respect to test cases.

The test packets are released from packet injection point (PIP), which is the computer that hosts the firewall testing tool. All the traffic entering and leaving the firewall are recorded and the collected data is analyzed to obtain test outputs, which is compared with expected outputs to determine a test result. The allowed packets are expected to be seen at the packet leaving point but not the denied ones.

2.21 CLUSTERING BASED NOVEL TEST CASE PRIORITIZATION TECHNIQUE

Formal testing phase software testers developing the test suites, these test suites often keep saving for reuse them future’s perspective. [Krishnamurthy, R and Mary, S.S.A 2009]. Rerunning such test suites can cost one-half of the total cost required in the maintenance phase and take unjustifiable excessive mount of time, For example, an industry reported that executing all the test cases at the maintenance phase required seven weeks for a product having 20000 lines of code. In these
situations, testers may re-arrange the test cases to assign the priority to each test case to enable their execution execute them in a specific order with the objective of achieving their desired performance goals. Test case prioritization is different from test case selection and minimization in the sense that it overcomes the drawbacks of these two by not discarding the test cases realizing that sometimes discarding of the test cases is not acceptable.

These techniques do not bother about the execution time taken by the test cases. Requirement based clustering incorporates term extraction and some manual process that is time-consuming. The clustering techniques used in these approaches have their own disadvantages. A clustering based test case prioritization approach that first clusters the test cases on the basis of their common feature of using Fuzzy C-means and then perform Intra and inter-cluster prioritization based on coverage, complexity metric, execution time and test execution history [Jeffrey, D. and Gupta, N. 2006].

Test Case Prioritization Method with Weight Factors in Regression Testing on the basis of Measurement Metrics. In the software development life cycle, regression testing is a necessary component that seeks to uncover software errors after effecting changes to the program. A set of test cases should be written appropriately for effective testing of the software [Malishevsky, A.G. et al. 2006]. When the software is altered, a new set of test cases are added to the test suite for the changed requirements. It is unfeasible and ineffective to re-execute each test for every program function when a modification occurs. This tends to increase the size of the test suite, cost and time constraints. The problem of regression test case selection is solved by prioritizing test cases. Test case prioritization techniques can organize the test cases for increasing the effectiveness of testing.
A test suite containing $m$ test cases and a program containing $n$ statements, total statement coverage prioritization can be accomplished in time $O(\ldots m + n + m \log m)$ [Gregg Rothermel. et al., 2010][ Carlson, R. 2011] . The first term denotes the time required counting the statements covered by each test case, and the second term denotes the time required to sort the test cases according to coverage. Typically, $n$ is greater than $m$, in which case the cost of this prioritization is $O(mn)$. It also avoids the test suite minimization problems. Recently, the experimental studies fault detection capability saw enhancement through selective retaining of test cases. Several techniques for prioritizing test cases have been proposed these are detailed in the section following the related work. The main objective is to run test cases on the basis of practical weight factors to increase the likelihood of fault detection and detect the severe faults in early stages of testing life cycle [Kim, J.M. Porter, A.2012] [Elbaum, S. et al. 2009].

Test case prioritization is an important kind of regression testing technique. Test case prioritization approaches typical existing test cases for regression testing according to attain performance goals [Qu, B. 2007][Agrawal, H. 2013]. The metric of Average Percentage of Fault Detected (APFD) is extensively used for analysing test case prioritization technique. Researchers have used various prioritization techniques to measure Average Percentage of Fault Detected (APFD) values and found it producing statistically significant results. Average Percentage of Fault Detected (APFD) is a measure that the average number of faults identifies in a given test suite. The Average Percentage of Fault Detected (APFD) values range from 0 to 100 and the area under the curve by plotting percentage of fault detected against the percentage of test cases executed [Rothermel, G. et al. 2012].
The prioritization techniques involve coverage of test cases. Ranking of test cases is given on basis of the statement, function, branches covered by each test case [Arafeen, M. J. Do,H. 2013]. There are three different groups, namely, comparator group, statement level group, functional level group. Each containing fine granularity techniques such as random ordering, optimal ordering, total statement (total-st), total branch (total-bar), total function (total-fn), additional statement (add-st), additional branch (add-be), additional function (add-fn). Total statement (total-st) TCP techniques measure the coverage of statements in a program. Prioritization of test cases is done by the total number of statements they cover and sorted out in the order of coverage achieved. Total branch and total function are similar to the total statement (total-st) except that they use the function and the branch coverage instead of the statement coverage information. Additional statement (add-st) TCP techniques select a test case that covers the maximum number of statements not covered in each round. When two test cases cover the same number of additional statements in a round, each randomly picks one, with the additional function (add-fn) and additional branch (add-be) being similar to the additional statement (add-st) except that they use the branch coverage and the function coverage information respectively. The code coverage strategies are measured using a weighted average percentage of fault detected (APFD), average percentage of branch covered (APBC), the average percentage of decision covered (APDC) and the average percentage of statement covered (APSC). Average Percentage of Fault Detected (APFD) is the measure of the rate of fault detection made by a test suite. The potential goal of the above technique is to increase the rate of fault detection at earlier stages of the testing process. Kim and Porter have introduced a technique that uses the historical execution of data for test case
prioritization in a regression testing. In this technique, Test case prioritization problem is considered as a probabilistic approach so that selection probability of each test case in a test suite is based on the execution history. This is referred to in the next test session. Test cases are prioritized according to the execution history, faculty detection effectiveness and the coverage of the program respectively.

2.21.1 Requirement Factors

In the technique proposed by the researcher, four requirement factors for the test cases are considered. Considering their substantial impact on the new software. Each of the four factors is discussed.

- **Customer Assigned priority (CP)**
  
  It is a measure of the importance of customer requirement. So, the requirement with highest customer importance should be tested early to improve customer satisfaction. The customer assigns values for each requirement ranging from 1 to 10 where 10 denote highest customer priority.

- **Implementation complexity (IC)**
  
  It is a measure of the complexity in implementation of the requirement by the development team. Requirements with high implementation complexity have a high number of faults. Each Requirement is assigned values ranging from 1 to 10 where 10 indicate high implementation complexity.

- **Requirement Change (RC)**
  
  It is based on the total number of times a requirement has been changed in their development cycle. 50% of faults in the projects is identified in the requirement
phases due to changes in the requirement stages. The volatility changes for all the requirements are normalized to a 10 point scale.

- **Requirement Coverage (ReqCov)**

  It is the subjective measure of the total number of requirements covered by each test case in a test suite. A weight in 10 point scale can be assigned where 1 is the minimum value and 10 is the maximum value. Literature review shows that requirement coverage can help to validation of implementation all requirements in the system.

- **Complexity Factor**

  The complexity factor determines the total effort required for executing the test cases. Several studies show the complexity of test cases as one of the most important factors for regression test case prioritization. The following complexity factors are considered.

- **Test Case Complexity (Tcplx)**

  It is the measure of the difficulty and complexity in running the test case during the testing process. It is also referred to as the effort needed to execute the test cases. Weightage is assigned by true or false conditions where true denotes 10 points and false denotes 0 points for representing the complexity of test cases.

- **Test Impact (TI)**

  It is based on the impact seen on test cases during the testing of software. This factor helps to assessment of the importance of test cases to determine if test cases are not executed. Weightage can be given in three states such as high, medium, low
where high denote 8 to 10 points, medium denotes 4 to 7 and low denote 1 to 3 points in scale assignment.

2.22 IDENTIFYING EFFECTIVE TEST CASES THROUGH K-MEANS CLUSTERING FOR ENHANCING REGRESSION TESTING

A classification of test cases into two groups using k-mean clustering, the idea is to focus on effected portions of the program and the related effected test cases and eliminate the need for executing non-effected test cases. Regression testing is one of the largest contributors to the overall cost of software [Harrold, M. J. and Orso, A. 2008] [Harrold, M. J et al. 2013]. Improve the cost effectiveness of regression testing techniques, many researchers have proposed and empirically studied various regression testing techniques, such as regression test selection test suite minimization and test case prioritization [Yu-ching Hung. 2012] [Harrold, M.J et al. 2012] [Geetanjali Chaurasia, and Sonali Agarwa, .2016]. The purpose of cluster analysis is to group a set of objects such that the objects clustered in the same group or clusters are more similar to each other than to those in the other groups or clusters. The K-means clustering is a common method of cluster analysis which aims at partitioning n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Results show a substantial reduction in the cost of regression by the test case clustering that the test case clustering, when applied to regression testing.

The contributions of this paper are as follows: Introduction of a test case clustering-based approach for reducing the cost of regression testing; Adopting a k-means clustering algorithm based on Hamming distances to effectively cluster test cases, i.e. a binary classification. Evaluating the performance of the proposed
technique when a number of factors related to source code coverage and distances are
controlled for the first reviews some of the regression testing techniques and describes
their possible limitations. It then reviews the techniques and algorithms that are
referred to in this paper including $k$-means clustering and the Hamming distances
[Kayes, M. L.2011].

2.22.1 Regression Testing

There exist numerous regression testing techniques including prioritization-based
test case execution, test case selection, and test suite minimization. Various
techniques have been proposed for prioritizing test cases for regression testing; a
technique similar to the researcher’s proposed approach in this paper is introduced.
The technique is based on utilizing the Hamming and string distances to differentiate
text within test cases for their prioritization [Rothermel G and Harrold, M. J.2014].
There are some other approaches also such as use of topic model and linguistic data to
prioritize test cases [Zheng Li, et.al.2007].

A typical test case prioritization approach follows two basic key ideas: 1) the
utilization of greedy search algorithms with its aim at ranking test cases in a
descending order with the hope of detecting newly introduced faults in the earlier
stages of regression testing, and 2) the heuristic-based search approaches for exposing
remaining as well as the newly introduced faults with the minimum number of test
cases chosen to be exercised. Regardless of which search strategy is used, the existing
search-based approaches fall short in determining adequacy criteria for regression
testing. More precisely, the tester may not be aware when a regression testing process
has been reached.
Regression testing becomes essential in software testing process. This leads to regression testing in which all the tests in the accessible programmes or suite should be re-executed [Muthusamy, T and Seetharaman, K. 2014]. Thus incurring excess cost, time and resources. Test case prioritization is an important technique adopted in regression testing. Prioritize the test cases depending on business impact, importance & frequently used functionalities. Selection of test cases based on priority will significantly reduce the regression test suite.

Regression testing goal and the testing are adequate enough. Therefore, it is likely that the test practitioner re-runs the entire test suite for the newly released version of the software system. The selection-based regression testing techniques intends to determine a subset of the test suit from the previously released software by highlighting the affected portions of the code which have the potential to induce errors. A typical selection technique consists of two major activities: 1) highlight the changed portions of the code, and 2) select test cases that are likely to detect bugs caused by the affected and changed parts. The ability of use of this technique to reduce the cost of regression testing significantly has been observed.

Several surveys and studies show that very few software industries deploying systematic test selection strategies in their testing activities. As a complementary to test case prioritization, test suit minimization techniques focus on identification of redundant test cases and eliminating them with the objective of reducing the test suite size and thus reducing the time and effort needed to perform the regression test. There exists some research grounded on the guess that the validation of a particular requirement can be fulfilled by a particular test case. In practice, this assumption does not hold good primarily because of the complex characteristic of a software system,
especially for some functional requirements. K-means Clustering is one of the simplest but most popular unsupervised learning algorithms that have been extensively used to cluster data points. The algorithm is easy to implement and apply even on large data sets and therefore the k-mean clustering technique has been successfully applied in the various areas, ranging from statistics, data mining to general information technology. A typical k-means clustering algorithm consists of the following basic steps:

- **Initializing**: Place K points into a multidimensional space to represent the centroid of each group;
- **Clustering**: Assign an object to the group that has the closest centroid;
- **Updating**: Update the centroid of the group into which a new object is added;
- **Repeating**: Repeat clustering and updating steps until termination conditions and adequacy criteria are satisfied.

### 2.22.2 Code Coverage Differences Based on Hamming Distances

Code coverage is a measure used in adequacy testing and devising test cases on the basis of the internal logic and structure of software systems. In other words, code coverage describes the degree to which the source code of a program has been tested. The code coverage based on program statements is the simplest form of this adequacy criterion, which aims at checking whether each executable statement in the program has been exercised. Hamming distance is widely used in information theory to measure the differences between two strings [Offutt, J. et al. 2010].

The metric based on Hamming distances counts the minimum number of substitutions required to change one string into another, or the minimum number of
errors that could have transformed one string into the other one. For example, the Hamming distance between “ABC def” and “abb defog” is 1, whereas, the Hamming distance between “00110011” and “00110000” is 2. Our approach employs the Hamming distance metric together with the code coverage information to measure the similarities/dissimilarities between two test cases. More specifically, based on two coverage reports generated by test cases A and B for the same program, the Hamming distance of the two test cases A and B is measured by counting the number of statements that have been exercised by only one of the two tests cases.

In order to have a better insight into the use of Hamming distances to measure the similarities/dissimilarities of two test cases when applied to the coverage information. The Java code snippet implements a class for computing the sum of the abstract values of two values. Since the input numbers could be either double or an integer, an override method is implemented which can take different input types. The code is composed of 13 lines with one class Abs Sum along with two methods. A fault is injected to line 11. Four test cases are devised for the purpose of testing the class and its methods. Using the binary numeric values 1 and 0 to represent covered/not covered statements, the representation of statement coverage can be transformed by each test case to a binary string.

The binary representation string for test case tc1 is, therefore “111111000000”, and consequently the binary representation string for tc2 are “1000000111111”. Thus, the Hamming distance value between tc1 and tc2 is 12. If the Hamming distance between the binary string representations of two test cases is zero, the two test cases appear similar. Similarly, if the Hamming distance calculated is some non-zero value, the conclusion may be that the two underlying test cases are
different. It is important to note that the magnitude of Hamming distances may reflect the significance differences among test cases and their coverage. A large Hamming distance indicates the significance in the difference between two test cases.

2.23 CLUSTERING APPROACH TO TEST CASE PRIORITIZATION USING CODE COVERAGE METRIC

Software testing as a process done with the intention of locating the defect in the existing software [Thillaikarasi Muthusamy and Seetharaman.K, 2013]. It is also the process of evaluation of a software item to detect differences between the given input and the expected output and also to assess the features of a software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words, software testing is a verification and validation process of a computer program or application/product to meet the requirements that guides its design and development, works as expected, can be implemented with the same characteristics, and satisfies the needs of stakeholders.

2.23.1 Test Case Prioritization

Test case prioritization is that of increasing a test suite’s rate of fault detection a measure of how quickly a test suite detects faults during the testing process [Elbaum, 2008]. An improved rate of fault detection can provide earlier feedback on the system under test, enable earlier debugging, and increase the likelihood that, if testing is prematurely halted, those test cases that offer the greatest fault detection ability in the available testing time will have been executed.

Test case prioritization is a mechanism needed for arranging a test case in an appropriate order to increase their effectiveness at meeting some performance goals.
and the rate of fault detection. Test case prioritization is a method to prioritize and schedule test cases in appropriate order. To run test cases of higher priority before than the lower priority test case in order to minimize time, cost and effort during software testing phase [Jones, J. and Harrold, M. 2013] [Elbaum,S. et al. 2012].

Various performance goals are like the rate of fault detection which is a measure of the speed of the fault detection so that during testing faster feedback can provide about the system under testing and allow the software tester to correct the software at earlier phase as possible [Rothermel, G et al. 2014]. Regression testing is a process of retesting the modified software and ensures that new error is not introduced into the previously tested source code due to these modifications. Most regression testing research has treated software systems as if they possessed a single homogeneous configuration [Qu, X. et al. 2008]. A primary focus has been on techniques for reducing test suite size or on ordering test cases. Regression testing is a very expensive testing process. The software tester may prioritize the test case for economizing on the cost of regression testing so that the test case which is more important are run earlier during regression testing process. A test plan is constructed on the basis of the identification of changes in the program and classification of test cases. [Yoo, S. et al. 2009][Yoo, S. and Harman, M. 2010]. Although the definition of a test plan remains informal, it provides a basis for the subsequent literature. It is of particular importance to regression test case selection techniques, since these techniques essentially concern the problem of identifying retestable test cases. Similarly, test suite minimisation techniques concern the identification of obsolete test cases. Test case prioritisation also can be thought of as a more sophisticated approach to the construction of a test plan.
Four different test case prioritization techniques that usually use the three types of information written are considered below: Code Coverage, Complexity Metric and Fault History Information. The purpose is to define factors for testing on the basis of the specific clustering that can be done. This variable is also a numeral as it includes the calls to a routine, procedure, method, function or subprogram. It is the sum of both built-in functions as well as user defined functions. No. of Function calls (Fc) = Local functions + nested functions + private functions + overloaded methods after the test suite is executed the first time for the software then it yields a result seen that when it is not – prioritized. Now with the ability to keep an account for the re-test that is regression test that what is the statement coverage for each test case and to examine the number of function calls occurring in the particular statement coverage for the particular test case. As described above the product of the statement coverage and the no. of function calls will be computed.

Test case prioritization technique based on genetic algorithm is a risk-based regression testing, its purpose is to use the least number of test cases that test the most potential bugs [Jun, W. et al.2011]. It can save computational resources and time. This becomes the software metric or the basis of the ordered set of test cases. As each test case has a value now, they will be ordered as the one with the greatest value as the highest prioritized test case and the number of followed in the descending order. An algorithm is proposed for this with the assumption that the statement coverage and number of function calls is known, such that this is given as the input to the algorithm and the output is the prioritized (in some order (either ascending or descending)) test suite.
2.24 FAULT DETECTION BASED TEST CASE PRIORITIZATION

Software testing is a thorough arrangement of activities carried out with the expectation of discovering mistakes in software. It is an activity in the development procedure of software focused for assessing a software item, for example, system, and subsystem and features (e.g. functionality, performance & security) against a given arrangement of framework requirements [Yulei Pang, et al. 2013]. Software testing is the process of validating and verification of the proper functioning of a program. Many researchers have demonstrated that software testing is a standout amongst the most fundamentally essential periods of the software development life cycle consumes significant resources in terms of effort, time and cost. Test case prioritization (TCP) improves the probability that if the test cases are utilized for regression testing in the given direction, they will more firmly meet some target than they would if they were executed in some other direction.

2.25 NON-FUNCTIONAL REQUIREMENTS IN AGILE PROCESSES

Agile software development methodologies, such as Scrum and Extreme Programming (XP), have been gaining traction and adoption on a global scale and are expected to grow in the next 10 to 15 years. Software engineers face challenges constantly in the matter of delivery of quality software under tight schedules, hence, pressuring developers for a quick start of coding with “ad hoc short cuts” to accelerate the development phase. Therefore, such ad-hoc methods suffer from lack of a well-defined structure and tool support [Medhun Hashini. 2014]. Agile software development methodologies have not been adequately identified, modelled, and linked Non-Functional Requirements (NFRs) and their potential solutions (operationalization’s) with Functional Requirements (FRs) during early requirements
analysis phases. Researchers agree that Non-Functional Requirements (NFRs) have been traditionally ill-defined during conventional requirements engineering phases and especially ignored in agile methodologies [Bach, J. 2008].

Academic research and industrial case studies suggest agile software development methodologies, such as Scrum and Extreme Programming (XP), gaining tremendous popularity and proven to be successful in implementing and quickly delivering quality product [Weam M. Et al. 2012]. Non-functional requirements (and often crosscutting concerns such as security, performance, and scalability) have been traditionally ignored or at best ill-defined in agile environments. This research study is aimed at introducing the Non-functional Requirements [Srikanth H. and Williams, L. 2005].

Modeling for Agile Automatic processes—is a Java simulation tool that supports the more general Non-functional Requirements Modeling for Agile Processes (NORMAL) Methodology. The fundamental objective of this process is to develop a simple non-functional requirements Modeling tool that specifically supports agile software development processes like Scrum. These implement three building blocks of the more general Non-functional Requirements Modeling for Agile Processes (NORMAL) framework: Agile Use Case (AUC), Agile Loose Case (ALC), and Agile Choose Case (ACC). It also provides support for planning agile releases and iterations (sprints) and visualizing such plans in a tree-like structure.

2.25.1 Normatic: An Agile Modeling Tool for NFRS

The objective of NORMATIC are integrate Functional Requirements (FRs) and Non-Functional Requirements (NFRs) modeling under one agile tool. Parse user stories automatically to improve agility. Identify and classify Non-Functional
Requirements (NFRs) to improve agility. Support a newly proposed and enhanced W8 Story Card Model that captures functional or non-functional agile requirements. Visually classify Non-Functional Requirements (NFRs) and their potential solutions into three types: source code, architecture and design, and organizational policies. Improve visualization through a downscaled version of Chung’s Non-Functional Requirements (NFRs) Framework with new color-coding of clouds that is suitable for agile processes to model Non-Functional Requirements (NFRs) soft goals and their potential solutions. Provide extensibility through configurable project management and requirements quality metrics. Implement a risk-driven requirements implementation sequence algorithm used in agile planning. Visualize proposed agile plans based upon business value as well as two newly proposed priority schemes. First, the W8 User Story Card Model is proposed as an enhancement of the agile index card technique used to gather high-level agile requirements in eXtreme Programming (XP) and Scrum. The story card is a physical 3x5 index card used to capture 3-5 sentences of high-level requirements from the stakeholders’ point of view. Despite the recommendation of that requirements should be expressed in one sentence and avoid complex sentences (which may contain more than one requirement), capturing a user requirement in 1-2 sentences will almost certainly not suffice.

On the other hand, the traditional use case model is heavyweight and stifles agile processes which value working software over documentation. A good balance between the two approaches is to capture the most important elements of a user requirement, along with essential project management and requirement of quality metrics and automate as much of the Non-functional Requirements Modeling for Agile Processes (NORMAL) framework as possible in an easy to use visual
environment. Furthermore, the tool supports risk calculations of Non-Functional Requirements (NFRs) and their potential solutions as well as visually reasoning about the requirements implementation sequence, hence, aiding in release and sprint planning. An Agile Choose Case—which is adopted from Chung’s framework—models one or more potential solutions to the non-functional requirement. Our novel scheme suggests that potential solutions can be source code (e.g. an Aspect) which is color coded as a dark green cloud, architectural or design constraint which is color-coded as a dark red cloud, or organizational policy which is color-coded as a dark blue cloud.

2.26 SUMMARY

This chapter presents literature review of various related methods of agile software testing, DAG based approaches, and test case prioritization methods such as user requirement based test case prioritization, code coverage based test case prioritization and fault detection based test case prioritization etc.,. An Extent review of literature about various types of agile methods, principle of agile method and characteristics of agile methods are also presented. It form the foundation and background knowledge to identify the research problem for the present study.