AUSTIN COMMUNITY COLLEGE

A Concise Review of Challenges involved in the Testing of Modern Software Applications

ITSE 1392 Special Topics In Computer Programming

Shruti Ratna Charugulla

12/15/2017

Table of Contents

Abstract

In the context of development of modern software applications, Agile SDLC (software development life cycle) has been hogging the limelight in the past couple of decades. Literature shows that organizations are experiencing tremendous success in meeting with the rapid-paced change of customer needs by adopting agile processes that promote incremental and iterative approaches in the design and development of software. Software testing is extremely challenging to pursue in such organizations. Cloud computing has emerged as a new computing paradigm that facilitates the development and utilization of highly flexible, elastic services on-demand, and over broad-band network access. Testing of cloud-based software especially in the context of agile has very unique challenges associated with it. Modern consumer products are complex and their design and development involves multiphysics simulations. Organizations that develop multiphysics software are now increasingly adopting agile processes to deliver value to the market more quickly than ever. Software testing in such a context brings yet another set of unique challenges. This paper provides a general overview of agile SDLC while discussing few of the major challenges involved in testing of modern and complex software applications being developed in that context. Further, the paper also discusses the unique challenges involved in the testing of cloud-based and multiphysics simulation software. Finally, the paper reviews some of the solutions being adopted by various software development organizations to mitigate their testing challenges.

Introduction

Agile software development methodology is becoming increasingly popular in modern software development organizations such as those related to telecommunications, finance, science, engineering, and healthcare only to name a few. The agile approach encourages teams to split the lengthy software requirements, build, and test phases into smaller increments and deliver working software in stages. See Figure 1 for an illustration of the process. In fact, there is even a manifesto (Agile Manifesto, 2017) for agile software development that is like a formal proclamation of four key values and twelve principles guiding such a process. The key values discussed in the manifesto emphasize: 1) individuals and interactions over processes and tools, 2) working software over comprehensive documentation, 3) customer collaboration over contract negotiation, and 4) responding to change over following a plan. The twelve principles are not included here but are available to refer to in (Agile Manifesto, 2017). The manifesto was created as an alternative to the document-driven, heavy weight software development processes such as the waterfall approach (Dyba, 2008).

Software testing by itself is a challenging pursuit and working as a tester in an agile team can be even more challenging. Some of the challenges (Agile testing challenges, 2017) (Challenges in agile testing, 2017) involved in agile testing are discussed below, not necessarily in any particular sequence or priority.

Challenges in Agile Testing

Changing requirements

Last minute changes in requirements are a very common scenario in an agile project. However, if requirements change towards the end of a sprint when there is not much time left, it becomes an overhead.

Frequent Builds

Since code is changed and built frequently, the likelihood of it breaking regression is much higher.

Lack of communication

Effective communication is very critical in an agile project. However, due to the pace at which products are released, communication gaps frequently arise between development and testing teams. This is in spite of the fact that individuals and interactions are emphasized over processes and tools in the agile manifesto. As a result, critical defects may not be identified early.

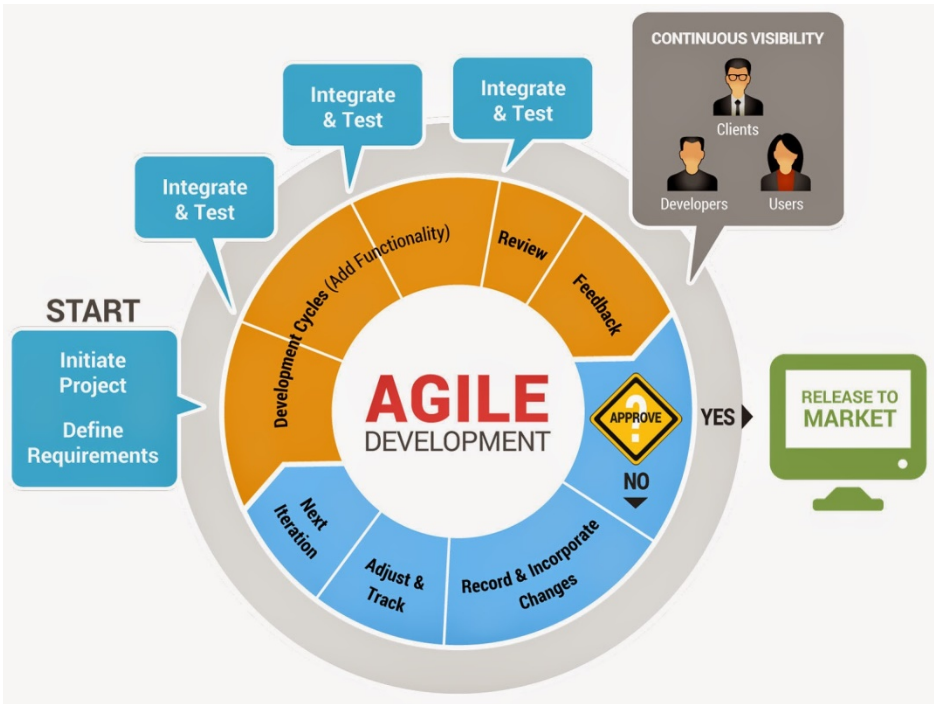


Figure 1. Agile SDLC illustration (courtesy: web)

Lack of focused testing

While it is very beneficial and also required for a tester in an agile team to participate in various story-planning meetings, sometimes it can be very overwhelming. Planning for future stories while still discussing and executing stories of the current sprint can lead to confusion and lack of focus on testing.

Miscalculation of story estimates

Due to increased pressure to release valuable software very frequently albeit in incremental steps, user stories may often be underestimated. As a result, testers may focus on the test coverage but not necessarily on quality. The huge list of testing tasks that include the development and execution of test cases along with reporting of test results to the team and the management could easily overwhelm the tester.

Testing with artifacts

Developers working in agile teams frequently use personal branches to make changes related to the specific user stories they own. It is a common practice to build artifacts out of such branches and provide to testers to seek early feedback before the changes are merged into the main development branch. Testers must then switch from one feature's artifact to another and test appropriately. This process could lead to confusion and errors. Furthermore, testers are charged with the responsibility to retest the changes once they are in the build that results from the main development branch (i.e. the non-personal one).

Continuous learning

Agile teams are often tool-rich and testers are required to continuously learn and constantly update their skills. For example, testers should be technically competent to help developers with integration and API testing. They should also be very comfortable with scripting to automate regression tests. If the testers possess a purely manual or exploratory background, they will find it difficult to keep-up with the pace of delivery. This often makes the job of a tester demanding and of course, exciting as well.

Distributed virtual agile teams

Agile software development teams are now routinely distributed across the world in different time zones and interact virtually. If effective communication is to be facilitated for such teams, their work infrastructure (i.e. network connectivity, effective chat and file sharing media such as Skype, other hardware and software infrastructure) must be sound enough to help them engage in a collaborative testing operation. Moreover, the need to have highly visible and transparent project management is very important since team members are not collocated in a distributed environment.

Increased regression risk

In agile life cycles, code that may have worked in previous sprints/iterations has the potential to get churned by new features in each subsequent sprint increasing the risk of regression.

Many meetings

Given that agile focuses on working software over comprehensive documentation, it can result in many more meetings than usual between leads, managers, testers, developers, and other stakeholders. That may sometimes reduce effectiveness and efficiency.

These are some of the generic testing challenges that agile-based software development organizations frequently encounter. The rest of the paper is organized as follows: section II first provides an overview of cloud computing and cloud software and presents some of the unique challenges encountered in agile-based cloud software testing. Section III provides an overview of multiphysics and presents the unique challenges encountered in the agile-based testing of simulation software of that kind. Section IV provides an overview of the solutions that are being discussed in literature to mitigate some of the testing challenges discussed in previous sections. Finally, section V makes few concluding remarks.

Cloud-Computing and Challenges involved in Cloud Software Testing

Introduction to Cloud Computing Services

Cloud computing is the delivery of computing services—servers, storage, databases, networking, software, analytics, and more—over the internet (Microsoft Azure, 2017). Basically, using cloud computing, one can create new applications and services, host websites and blogs, stream audio and video, deliver software on demand, and analyze data to make predictions. Companies offering these computing services are called cloud providers who typically charge for cloud computing services based on usage. Some of the important characteristics that define cloud computing include (Cloud Computing Basics, 2017): 1) on-demand self-service, 2) broad network access, 3) resource pooling (multi-tenancy), 4) rapid elasticity, and 5) measured service.

Few of these terms were formally defined in a report published in 2011 by the U.S. Department of Commerce's National Institute of Standards and Technology (NIST) (Mell, 2011). The definitions are given below for the sake of completeness and clarity.

On-demand self-service
A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

Broad network access
Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling
The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

Rapid elasticity
Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured service
Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Cloud providers offer their services via three types of computing service models. They include *Software as a service* (SaaS), *Platform as a service* (PaaS), and *Infrastructure as a service* (IaaS). The idea is that, when a user wants to make use of a software application but has nothing else in hand, then, SaaS is the go-to solution. On the other hand, if a user has a software application and likes to deploy and run on a publicly available platform, then, PaaS services must be utilized. Finally, when both software and platform are ready but not the hardware to run them on, then, IaaS services are to be leveraged. Figure 2 illustrates the above-mentioned service models pictorially.

Overall, cloud computing has emerged as a new computing paradigm that facilitates the development and utilization of highly flexible, elastic services on-demand, and over broad-band network access. These attributes are driving many organizations to move their businesses to the cloud platform.

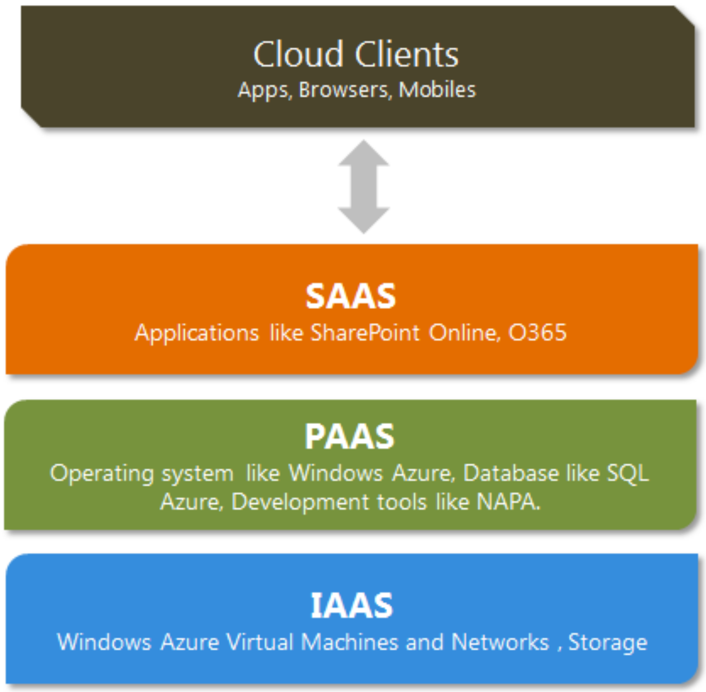

Cloud Clients
Apps, Browsers, Mobiles
SAAS
Applications like SharePoint Online, O365
PAAS
Operating system like Windows Azure, Database like SQL Azure, Development tools like NAPA.
IAAS
Windows Azure Virtual Machines and Networks , Storage

Figure 2. Cloud Computing Service Models
(courtesy: (Cloud Service Models (IaaS, SaaS, PaaS) + How Microsoft Office 365, Azure Fit In?, 2013))

Cloud testing (Gao, 2011) is now becoming a hot research topic in cloud computing and software engineering community. The term *Cloud Testing* is being used in literature in two different contexts (Cloud Testing, 2017) *viz.,* testing on-premises applications in QA (quality assurance) cloud and testing cloud-based (i.e. on-demand) applications themselves in a QA (quality assurance) cloud. *On-premises* software is installed and runs on computers on the premises (in the building) of the person or organization using the software, rather than at a remote facility such as a server farm or cloud (Wikipedia, 2017). *On-demand software* is a type of software delivery model that is deployed and managed on a vendor's cloud computing infrastructure and accessed by users over the Internet as and when required (Technopedia, 2017).

A new paradigm in cloud computing called *TaaS* (*Testing as a Service)* has emerged in the recent past. In that, organizations turn to cloud based QA environments to overcome the limitations of their existing testing infrastructure. Both traditional on-premises as well as on-demand applications can be tested by deploying them on the cloud by leveraging on-demand testing services. The cloud provides all the resources required for testing the application. On-premises applications generally need to be tested for functional, performance, and security requirements. On the other hand, the focus of testing on-demand applications must be on ensuring all of the attributes such as on-demand self-service, broad network access, resource pooling or multi-tenancy, rapid elasticity, and measured service that are already discussed above. The testing methodology has to evolve in all these scenarios and would need to take into account the virtualized infrastructure, network, application business logic, data and end-user experience. Testing cloud applications has its own peculiarities that demand for novel testing methods and tools (Incki, 2012). An excellent overview of cloud testing is given in (Jain, 2014). Advantages of cloud testing include reduced costs from a shared resources and large-scale test environments (Jain, 2014).

The unique attributes of cloud applications that are described above warrant extra consideration while testing. There are many challenges that are encountered in cloud testing in general and they become even more challenging when the testing occurs in an agile context. In fact, some of the most significant challenges are encountered before testing can even begin. For example, getting access to hardware and various software configurations to test against, deploying an environment for the application under test, and deploying a test environment for load and functional test tools all involve challenges to the testers. Few of the specific and unique challenges involved while testing cloud software are listed and briefly described next.

Challenges involved in Cloud Testing

Security

Since data is transferred over the internet, cloud testing is frequently prone to security attacks. Testers must take extreme caution to ensure the protection of privileged data.

Lack of universal standards

It seems as if there are no standard ways of accessing public cloud services. Service providers have their own architecture, operating models, and pricing mechanisms. Therefore, it is a challenge to test applications, which have the potential to switch from one vendor to another.

Lack of knowledge and expertise

Traditional software testers must gain additional knowledge and expertise to extrapolate their existing skillsets to apply them in the context of cloud testing. Given the complex layering of the cloud infrastructure and the rapidly evolving technologies, that becomes a big challenge to software testers especially in an agile environment.

Remote application dependency

Since there is an inherent dependence on applications that are remotely installed in the cloud, it is sometimes difficult to set up the desired type of environment for testing.

Short notice period and frequent releases

With frequent upgrades from cloud application providers, it becomes necessary that customers test the application for validity and security. The service provider usually allows one- or two-weeks’ time to the client to do this. Ensuring that an existing application is running successfully and maintains security within such a short period is a challenging task in an agile software development setting.

Performance Testing

It is important to identify the most used parts of the applications and test them for performance. Simulating the real-life scenario using a massive number of users, users from different locations, and other varying factors in which the applications will run is a difficult task.

Integration and Migration

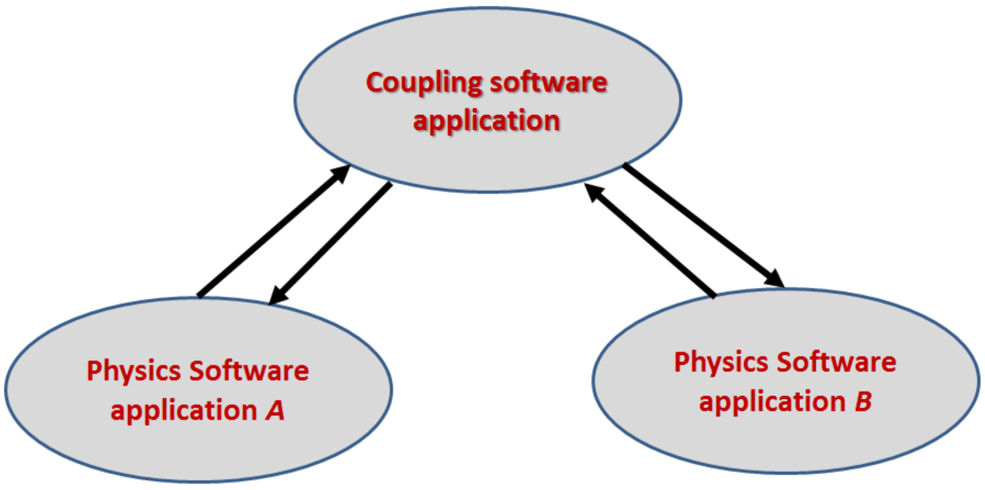
Integrating the SaaS application with other clients' applications requires data integration testing for both incoming and outgoing data. This is a challenging task to test for data validation and at the same time for maintaining data security and privacy. Data migration between *SaaS* applications or from other applications requires a lot of time to understand the requirements and test the integration results.

Multiphysics Simulation Software and Challenges Involved In Testing

Brief Introduction to Multiphysics Software

Multiphysics simulations are becoming increasingly popular in modern industrial Computer-Aided Engineering (CAE) applications such as those in aerospace, biomedical, civil, and nuclear industries. Complex problems in such industries involve physics from multiple disciplines such as electrical, electromagnetics, fluid-mechanical, and bio-chemical (Chimakurthi, 2017). Products such as automobiles, aircrafts, submarines, computer chips, biomedical devices such as stents are being designed and developed in industry using multiphysics simulation tools. Such software tools are being developed both in industry and academia. Few examples include proprietary applications from companies such as ANSYS Inc (ANSYS Multiphysics, 2017), Comsol Inc (Comsol Multiphysics, 2017), Adina R&D Inc (Adina Multiphysics, 2017), and Dassault Systemes Inc (Dassault Systemes Multiphysics, 2017). There are also several open source software packages such as Oomph-lib (Oomph-lib, 2017), and Open FOAM (OpenFOAM, 2017). A survey of such open source packages is given in (Babur, 2015).

Multiphysics software applications such as these frequently involve coupling of multiple individual software packages wherein each package corresponds to a different physics solution. Such an approach to multiphysics simulation software development is known as *partitioned multiphysics* (OpenMultiphysics, 2017). See Figure 3 for an illustration of this. It shows that there are two different software applications A and B (each of which is associated with a different physics) which are both coupled via another piece of software called the *Coupling software application* as indicated in the figure. In a typical simulation, the three software packages could run concurrently while exchanging data dynamically either via sockets or data files.


Coupling software application
Physics Software application *A*
Physics Software application *B*

Coupling software could communicate with A and B either via sockets or files to exchange simulation data.

Figure 3. Example of a Partitioned Multiphysics Simulation Software

Challenges Involved in Multiphysics Simulation Software Testing

Agile testing of multiphysics software of the type described above entails a number of challenges. Some of them are discussed below.

Familiarity with multiple physics software

Given the nature of the simulations, it becomes necessary for testers to become familiar with multiple software applications. As an example, consider a tester involved with the testing of the coupling software application shown in Figure 3. The tester in that case must become familiar (at least to some extent) with software components A and B as well tests the product efficiently.

Complex debugging situations

Testers must be able to coax many reluctant software packages simultaneously while trying to debug a complex situation. This is especially the case when the tester encounters a Class 1 (Fatal error) defect. It is almost always not clear which of the individual components involved in the simulation is the root cause of the problem.

Cross-team dependencies

Companies that are involved with the development of partitioned software typically have developers and testers of individual products (A and B in the example above) communicate with the personnel responsible for the development of the coupling software component. In that situation, testers are involved in heavy cross-team communication which is usually very daunting especially during a busy agile sprint.

Software complexity
Even though the example shown in Figure 3 shows only two software applications, in reality, many more such applications could become involved in the mix. As a result, the entire package becomes extremely complex and its testing naturally becomes a huge challenge.

Test planning and scheduling problems
They often occur when there is no separate test plan, but rather highly incomplete and superficial summaries in other planning documents. Test plans are often ignored once they are written, and test case descriptions are often mistaken for overall test plans. The schedule of testing is often inadequate for the amount of testing that should be performed, especially when testing is primarily manual. Significant testing is often postponed until too late in the development process, especially on projects using traditional sequential development cycles.

Stakeholder involvement and commitment problems
They include having the wrong testing mindset (that the purpose of testing is to show that the software works instead of finding defects), having unrealistic testing expectations (that testing will find all of the significant defects), and having stakeholders who are inadequate committed to and supporting of the testing effort.

Some Solutions to Prevent/Mitigate Agile Software Testing Challenges
Preventing and/or mitigating challenges in agile software testing is very essential to bringing a product quickly and efficiently to the market. Given how competitive the industry is in the present day world, this is more important than ever before. Below are few solutions that are being frequently discussed in literature in the context of agile software testing challenges and their prevention/mitigation. While the solutions presented below are very generic, they are especially useful in the development of modern cloud-based and multiphysics simulation-based software applications.

All hands project/feature planning meetings
Involvement of testers in all project planning meetings along with other stakeholders such as developers, team leaders, and documentation specialists across all teams expected to be involved in a project is very critical to ensure successful completion of the project. By doing that, testers will proactively participate in coming up with acceptance criteria of a feature story and be better placed to create acceptance tests more effectively and provide feedback to developers. On the other hand, care must be taken to ensure that the number of meetings do not become excessive. That is possible if all stakeholders of the project come well prepared for the meeting and possibly coming up with their own version of acceptance criteria for the requirements at hand.

Use of test management tools
Test management refers to the activity of managing the software testing process (Test management tools, 2017). And, a test management tool is a software used to manage tests(automated or manual) that have been previously specified by a test procedure. As testing gets more complex and timelines get crunched, use of test management software can make a huge difference in overcoming challenges.

Tools such as qTest, PractiTest, Zephyr (Zephyr test management solution, 2017), Test Collab, TestFLO, XQual, TestCaseLab, and QAComplete can be used to store information about how testing is to be done, plan testing activities, and report their status. While each tool may have a unique set of features, they all can generally be used to maintain and plan manual testing, run or gather execution data from automated tests and to enter information about defects found.

Automation testing (script- and script-less)

Test automation can help address the continuous testing challenge to some extent by alleviating repeated testing efforts. Automation ensures that the application is and continues to be in good shape with each new sprint/iteration. Running automated regression tests in a timely fashion will make sure that any new code that was added neither breaks nor destabilizes previously working features. Notable commercial test automation tools include HP Quick Test Professional, IBM Rational Functional Tester, Selenium, SOAtest, TestingComplete, Visual Studio Test Professional, and Oracle Application Testing Suite. Apart from these, in-house testing tools are also routinely developed by various organizations.

In recent years, a new paradigm in automation testing called the *scriptless automation testing* has evolved. Such a testing serves to reduce the time required to create automated tests by considerably reducing (not eliminating) the amount of scripting needed. Scriptless automation does not use record and playback, but instead builds a model of

the application and then enables the tester to create test cases by simply editing in test parameters and conditions (Hooda, 2012). This requires no scripting skills, but has all the power and flexibility of a scripted approach. With scriptless test automation, making changes in the test suites is easy when a change in the software under test occurs (Scriptless test automation, 2017). More importantly, it can make automation testing tool independent. An organization no longer needs specific professionals for Selenium or QTP or SoapUI.

Effective communication

Changing requirements or dropping stories mid-sprint is not very uncommon in agile projects. Such changes can affect the scope of testing efforts and can be very frustrating. However, both the testers as well as the testing processes must be robust enough to respond to them. They can do so by effectively communicating with all other stakeholders in the project. Testers can provide as much information as possible about which tests have been run and which part of the application has not been tested well so that the team as a whole can make an informed decision by way of risk assessment. In general, irrespective of how good a process is and tools are, if there is lack of continuous and effective communication among team members, the possibility of critical- and stop-ship defects creeping in will be very high. Usage of tools such as skype, google hangouts, webex, and google docs is now becoming very common in industry to communicate effectively.

Continuous professional development and training

Participation of testers in training and seminars focusing on state-of-the-art testing methodologies and tools will not only help in their professional development but also in leveraging them in testing projects. It ensures that the testers in the organization are competent in their profession.

Project timeline and estimation

Extreme care must be taken by the team to come up with a good estimate for any new feature that is scheduled to be developed. Story point estimates must be made by all team members and a consensus vote must determine the final number. That said, accurate time estimation is a challenging endeavor and usually improves as the team and management matures and handles many projects of similar nature.

Exploratory testing

Testers must be encouraged to conduct exploratory/unstructured testing to identify potential problems in the software before they are even identified by the users. Such problems must be addressed appropriately. If they are due to features that were already introduced, defects must be submitted. On the other hand, if the problems are due to any inadvertently unfulfilled acceptance criteria, then, they must be recorded in a backlog and be picked up in future sprints based on priority.

Expanded hardening sprints

If deemed necessary, the duration of hardening (i.e. testing a code that is frozen with no commits from developers permitted) sprints must be increased so as to allow testers to ensure quality of the released software.

Concluding Remarks

Agile SDLC (software development life cycle) has been hogging the limelight in the past couple of decades. The agile methodology is lauded for high quality software delivery and flexibility in adapting to new requirements. However, testing in the context of agile is fraught with a number of challenges and the ability to overcome them will determine success in a project. Cloud-based and multiphysics simulation-based applications are frequently developed in the context of agile and have a very unique set of software testing challenges. All-hands project planning meetings, use of test management tools, script-based and script-less automation testing, effective communication between project stakeholders, and tester professional development and training are all proven and effective solutions useful to overcome some of the challenges involved in testing an application in an agile environment.

Bibliography

(2017). Retrieved December 10, 2017, from Agile testing challenges: https://blog.smartbear.com/software-quality/agile-testing-challenges-inadequate-test-coverage/

(2017). Retrieved December 10, 2017, from Challenges in agile testing: https://www.axelerant.com/resources/team-blog/agile-testing-challenges-and-how-to-overcome-them

(2017). Retrieved December 2017, 8, from Cloud Testing: http://www.infosysblogs.com/testing-services/2011/11/how_is_testing_cloud_based_app.html

(2017). Retrieved December 10, 2017, from ANSYS Multiphysics: http://www.ansys.com/solutions/solutions-by-application/multiphysics

(2017). Retrieved December 10, 2017, from Adina Multiphysics: http://www.adina.com/multiphysics.shtml

(2017). Retrieved December 10, 2017, from Dassault Systemes Multiphysics: https://www.3ds.com/products-services/simulia/products/abaqus/multiphysics/

(2017). Retrieved December 10, 2017, from Oomph-lib: http://oomph-lib.maths.man.ac.uk/doc/html/index.html

(2017). Retrieved December 10, 2017, from OpenFOAM: https://en.wikipedia.org/wiki/OpenFOAM

(2017). Retrieved December 10, 2017, from OpenMultiphysics: https://sourceforge.net/p/openmultiphys/wiki/Partitioned%20Multiphysics/

(2017). Retrieved December 12, 2017, from Test management tools: https://en.wikipedia.org/wiki/Test_management

(2017). Retrieved December 12, 2017, from Zephyr test management solution: https://www.getzephyr.com/

(2017). Retrieved December 12, 2017, from Scriptless test automation: https://blog.qualitiasoft.com/resources/the-biggest-benefits-of-scriptless-test-automation

Agile Manifesto. (2017). Retrieved December 7, 2017, from Agile Manifesto: http://agilemanifesto.org/

Babur, O. S. (2015). A Survey of Open Source Multiphysics Frameworks in Engineering. *Procedia Computer Science , 51*, 1088-1097.

Chimakurthi, S. R. (2017). ANSYS Workbench System Coupling: a state-of-the-art computational framework for analyzing multiphysics problems. *Engineering with Computers* .

Cloud Computing Basics. (2017). Retrieved December 7, 2017, from https://www.ibm.com/blogs/cloud-computing/2014/02/cloud-computing-basics/

Cloud Service Models (IaaS, SaaS, PaaS) + How Microsoft Office 365, Azure Fit In? (2013). Retrieved December 8, 2017, from Cloud Service Models (IaaS, SaaS, PaaS) + How Microsoft Office 365, Azure Fit In?: https://www.cmswire.com/cms/information-management/cloud-service-models-iaas-saas-paas-how-microsoft-office-365-azure-fit-in-021672.php

Comsol Multiphysics. (2017). Retrieved December 10, 2017, from https://www.comsol.com/

Dyba, T. a. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology* , 1-27.

Gao, J. B. (2011). Cloud Testing- Issues, Challenges, Needs and Practice. *Software Engineering : An International Journal (SEIJ) , 1* (1), 9-23.

Hooda, R. (2012). An Automation of Software Testing: A Foundation for the Future. *International Journal of Latest Research in Science and Technology , 1* (2), 152-154.

Incki, K. A. (2012). A Survey of Software Testing in the Cloud. *IEEE Sixth International Conference on Software Security and Reliability Companion* (pp. 18-23). IEEE Computer Society.

Jain, S. a. (2014). Testing as a Service (TaaS) on Cloud: Needs and Challenges. *International Journal of Advanced Research in Computer Science & Technology , 2* (2), 335-340.

Mell, P. a. (2011). *The NIST Definition of Cloud Computing.* National Institute of Standards and Technology (U.S. Dept of Commerce).

Microsoft Azure. (2017). Retrieved December 7, 2017, from What is Cloud Computing?: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

Smith, J. (2010). *How to do Software Testing.* New York: Random House.

Technopedia. (2017). Retrieved December 9, 2017, from https://www.techopedia.com/definition/26710/on-demand-software

Unknown. (n.d.). *How to Do Software Testing*. Retrieved June 3, 2014, from http://www.howtotest.com

Wikipedia. (2017). Retrieved December 9, 2017, from https://en.wikipedia.org/wiki/On-premises_software