Modul I: Iniţiere în Software Testing
Partea 1: Introduction in Software Development and Testing

Contents

1. The history of quality

The quality movement can trace its roots back to medieval Europe, where craftsmen began organizing into unions called guilds in the late 13th century.

Until the early 19th century, manufacturing in the industrialized world tended to follow this craftsmanship model. The factory system, with its emphasis on product inspection, started in Great Britain in the mid-1750s and grew into the Industrial Revolution in the early 1800s.

In the early 20th century, manufacturers began to include quality processes in quality practices.

After the United States entered World War II, quality became a critical component of the war effort: Bullets manufactured in one state, for example, had to work consistently in rifles made in another. The armed forces initially inspected virtually every unit of product; then to simplify and speed up this process without compromising safety, the military began to use sampling techniques for inspection, aided by the publication of military-specification standards and training courses in Walter Shewhart's statistical process control techniques.

The birth of total quality in the United States came as a direct response to the quality revolution in Japan following World War II. The Japanese welcomed the input of Americans Joseph M. Juran and W. Edwards Deming and rather than concentrating on inspection, focused on improving all organizational processes through the people who used them.

By the 1970s, U.S. industrial sectors such as automobiles and electronics had been broadsided by Japan's high-quality competition. The U.S. response, emphasizing not only statistics but approaches that embraced the entire organization, became known as total quality management (TQM).

By the last decade of the 20th century, TQM was considered a fad by many business leaders. But while the use of the term TQM has faded somewhat, particularly in the United States, its practices continue.

In the few years since the turn of the century, the quality movement seems to have matured beyond Total Quality. New quality systems have evolved from the foundations of Deming, Juran and the early Japanese practitioners of quality, and quality has moved beyond manufacturing into service, healthcare, education and government sectors.

2. The history of quality in software

The history of software testing can be traced quite a while back, way before the first computer was created. It was initially defined as theory to support software development, software quality and project management practices so that it could be extended in the following years by practical applications.

Below are some key moments that helped the definition and setting the ground for the QA profession.

- *1878 - System flaw termed as bug (Edison)*. According to the Yale Book of Quotations the American inventor Thomas Alva Edison uses the term 'bug' in a letter to Theodore Puskas to describe a flaw in a system. According to other sources the term bug was commonly used to describe systems faults in Edison's time.

- *1939 - Shewhart Cycle*. In his second book Statistical Method from the Viewpoint of Quality Control Walter Andrew Shewhart publishes his Scientific Method of Improvement. Page 45 of this book displays the first version of the Shewhart Cycle; a three step cycle constituting a dynamic scientific process of acquiring knowledge. Walter Edwards Deming later popularizes the method as the Plan Do Check Act method (The Deming Cycle).

  ISO founded - The International Organization for Standardization (ISO) is founded during a conference in London in 1946. ISO is born from the union of two organizations; the ISA (International Federation of the National Standardizing Associations), established in New York in 1926 and the UNSCC (United Nations Standards Coordinating Committee), established in 1944.

- *1951- Juran's Quality Control Handbook (Juran)*. In his book Juran's Quality Control Handbook Joseph M. Juran defines quality as 'fitness for use'. Juran, who is considered to be the founding father of quality management, defines three processes for the management of quality; quality planning, quality control and quality improvement.

  Total Quality Control (Feigenbaum). In his famous book 'Total Quality Control' Armand Vallin Feigenbaum defines quality as a customer determination. Quality depends on the perspective of the customer. The product should satisfy the customer in both actual and expected needs. There is a company-wide responsibility for quality.

- *1958- First software test team (Weinberg).* The first test team is formed by Gerald M. Weinberg, working as manager of Operating Systems Development for the Project Mercury. Project Mercury is the first human spaceflight program of the United States.

- *1959- Work Breakdown Structure*. In their classic paper Program Evaluation and Review Technique (PERT) D.G. Malcolm, J.H. Roseboom, C.E. Clark and W. Fazar introduce their method to analyze the involved tasks in completing a given project. Although not mentioned by that name in the paper they also introduce the Work Breakdown Structure.

  Critical-Path Method (Kelley, Walker). In their paper Critical-Path Planning and Scheduling James E. Kelley and Morgan R. Walker offer a planning and scheduling methodology for complex projects. They introduce the Critical-Path Method in order to create optimal direct cost schedules. The authors use project diagrams of tasks and link these tasks (jobs) based on three questions; what immediately precedes this job, what immediately follows this job and what can be concurrent with this job

- *1967- Evaluation of the Functional Testing of Control Programs*. In the IBM white paper Evaluation of the Functional Testing of Control Programs William Elmendorf calls for a disciplined approach to software testing.

  Software engineering introduced. The term 'software engineering' is coined by a NATO study group that recommends a conference to discuss 'the problems of software'. The report from the ensuing 1968 conference, which was sponsored by the NATO Science Committee and took place in Garmish, Germany, is titled Software Engineering. Brian Randell and Peter Naur point out in the introduction to their edition of the proceedings, "The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be [based] on the types of theoretical foundations and practical disciplines[,] that are traditional in the established branches of engineering.

- ***1968- NATO report mentions Software Quality Assurance***. During the Software Engineering conference sponsored by the NATO Science Committee (7th to 11th October 1968) among other things quality assurance for software production is one of the topics. The report of the conference includes the working paper Checklist for planning software system production by Robert W. Bemer. This paper contains a chapter on quality assurance. One of the questions in the checklist is 'is the product tested to ensure that it is the most useful for the customer in addition to matching functional specifications?'

- ***1969- Testing shows the presence, not the absence of bugs***. Edsger Dijkstra's famous quote was reportedly first spoken on a conference by the NATO Science Committee, Rome, Italy, 27–31 October 1969.

- ***1970- Waterfall model published (Royce)***. Winston Royce describes a waterfall model in the paper Managing the Development of Large Software Systems, presented to IEEE WESCON in 1970. The waterfall model is a sequential software development process. In his paper Royce also mentions the shortcomings of the waterfall and warns against using the method.

- ***2001- Agile Manifesto published.*** A conference in Utah by 17 representatives from different development methodologies leads to the Agile Manifesto in which the Twelve Principles of Agile Software are published.

Source: (http://www.testingreferences.com/testinghistory.php)

3. The software testing profession

The software testing profession expanded along the time and changed its purpose based on how requirements grew in complexity and size over time. As more and more practices and theories appeared, software became more and more complex and so did the goal of software testing. As the profession grew in visibility, numerous testing categories by which different approaches and concepts were applied became available.

3.1 Classification of software testing development over the years

Until 1956 it was the debugging oriented period, where testing was often associated to debugging: there was no clear difference between testing and debugging.

- From 1957-1978 there was the demonstration oriented period where debugging and testing was distinguished now - in this period it was shown, that software satisfies the requirements.
- The time between 1979-1982 is announced as the destruction oriented period, where the goal was to find errors.
- 1983-1987 is classified as the evaluation oriented period: intention here is that during the software lifecycle a product evaluation is provided and measuring quality.
- From 1988 on it was seen as prevention oriented period where tests were to demonstrate that software satisfies its specification, to detect faults and to prevent faults.

https://en.wikiversity.org/wiki/Software_testing/History_of_testing

3.2 Classification of testing by test levels

- Component testing (also called unit testing)
- Integration testing
- System testing
- Acceptance testing

3.3 Classification by test philosophies

- Test Driven Development
- Extreme Programming
- Behavior Driven Development

4. Why do we need software testing?

4.1 The purpose

Software testing is really required to point out the defects and errors that were made during the development phases.

It's essential since it makes sure of the Customer's reliability and their satisfaction in the application. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence.

Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which results in lower maintenance cost and hence results into more accurate, consistent and reliable results.

Testing is required for an effective performance of software application or product. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development as we will further see in this chapter.

It has slowly become, from an optional activity to a mandatory one in order to stay in the business.

4.2 Economic Cost of Software Bugs

Report Date: 2/2002 Price Tag: $60 Billion Annually

WASHINGTON (COMPUTERWORLD) - Software bugs are costing the U.S. economy an estimated $59.5 billion each year, with more than half of the cost borne by end users and the remainder by developers and vendors, according to a new federal study.

Improvements in testing could reduce this cost by about a third, or $22.5 billion, but it won't eliminate all software errors, the study said. Of the total $59.5 billion cost, users incurred 64% of the cost and developers 36%.

4.2.1 Knight Capital's $440 million loss

Incident Date: 8/1/2012 Price Tag: $440 million

(The Register) Knight Capital, a firm that specializes in executing trades for retail brokers, took $440m in cash losses Wednesday due to a faulty test of new trading software.

Unfortunately, the trading algorithm the program was using was a bit eccentric as well. On every stock exchange, there is a "bid" and an "ask" price. The bid price is what you'd like to pay the holder of the stock if you want to buy their shares. The ask price is what they'll pay to buy those same shares from you. There's always a spread between the two prices, with the "ask" being a few cents or more above the "bid". If the stock is thinly traded, then the spread between the ask and the bid is higher than what you'd see for, say, IBM.

Knight Capital's software went out and bought at the "market", meaning it paid ask price and then sold at the bid price--instantly. Over and over and over again. One of the stocks the program was trading, electric utility Exelon, had a bid/ask spread of 15 cents. Knight Capital was trading blocks of Exelon common stock at a rate as high as 40 trades per second--and taking a 15 cent per share loss on each round-trip transaction. As one observer put it: "Do that 40 times a second, 2,400 times a minute, and you now have a system that's very efficient at burning money".

As the program continued its ill-fated test run, Knight's fast buys and sells moved prices up and attracted more action from other trading programs. This only increased the amount of losses resulting from their trades to the point where, at the end of the debacle 45 minutes later, Knight Capital had lost $440m and was teetering on the brink of insolvency.

4.2.2 Microsoft Zune's New Year Crash

Incident Date: 12/31/2008

(Associated Press) Happy New Year from Microsoft Corp.: Your Zune is dead.

Thousands of Microsoft's Zune media players -- the software company's answer to Apple Inc.'s iPod -- unexpectedly conked out Wednesday and showed users an error message, prompting references to 'Y2K for Zunes'. The problems appeared when people tried to start up their devices.

4.2.3 Air-Traffic Control System in LA Airport

Incident Date: 9/14/2004

(IEEE Spectrum) -- It was an air traffic controller's worst nightmare. Without warning, on Tuesday, 14September, at about 5 p.m. Pacific daylight time, air traffic controllers lost voice contact with 400 airplanes they were tracking over the southwestern United States. Planes started to head toward one another, something that occurs routinely under careful control of the air traffic controllers, who keep airplanes safely apart. But now the controllers had no way to redirect the planes' courses.

The controllers lost contact with the planes when the main voice communications system shut down unexpectedly. To make matters worse, a backup system that was supposed to take over in such an event crashed within a minute after it was turned on. The outage disrupted about 800 flights across the country.

Inside the control system unit is a countdown timer that ticks off time in milliseconds. The VCSU uses the timer as a pulse to send out periodic queries to the VSCS. It starts out at the highest possible number that the system's server and its software can handle—232. It's a number just over 4 billion milliseconds. When the counter reaches zero, the system runs out of ticks and can no longer time itself. So it shuts down.

Counting down from 232 to zero in milliseconds takes just under 50 days. The FAA procedure of having a technician reboot the VSCS every 30 days resets the timer to 232 almost three weeks before it runs out of digits.

4.2.4 NASA Mars Climate Orbiter

Incident Date: 9/23/1999 Price Tag: $125 million

WASHINGTON (AP) -- For nine months, the Mars Climate Orbiter was speeding through space and speaking to NASA in metric. But the engineers on the ground were replying in non-metric English.

It was a mathematical mismatch that was not caught until after the $125-million spacecraft, a key part of NASA's Mars exploration program, was sent crashing too low and too fast into the Martian atmosphere. The craft has not been heard from since.

Noel Henners of Lockheed Martin Astronautics, the prime contractor for the Mars craft, said at a news conference it was up to his company's engineers to assure the metric systems used in one computer program were compatible with the English system used in another program. The simple conversion check was not done, he said.

http://www.cse.psu.edu/~gxt29/bug/softwarebug.html

4.2.5 Russian Interference in US Elections

Russian hackers allegedly hacked into the Democratic National Convention (DNC), and ended up manipulating the election in favor of Donald Trump. It is said that the Hackers sent out repeated phishing emails to multiple US institutions. Eventually, John Podesta, the chairman of Hillary Clinton's campaign accidentally clicked on one of the malicious mails, thus sanctioning access to over 60,000 emails of the Clinton campaign. Reports also suggest that the emails were forwarded to WikiLeaks website, which later published those mails in the run-up to the US Elections tainting Clinton's image further.

4.2.6 Hive Thermostat App Sends Users' Homes to 32c (90f)

Even with the abundant smart devices out there, the concept of the Internet of Things would still comprise of significant complications. In this specific context, "Hive", British Gas's smart thermostats, made news in late February, when an error persistently set user's homes to a 32C (90F). Aside from the threat the heat posed to susceptible individuals, many livid customers criticized the state of their upcoming energy bill due to this maddening glitch.

4.2.7 Yahoo Data Theft

Yahoo! informed two major data thefts this year: The first was in September, which ended up affecting over 500 million Yahoo! user accounts, and another one in December, which claimed that about one billion accounts were compromised. Information of all sorts, including email addresses, user names, passwords, dates of birth, security questions and even phone numbers, were all reportedly leaked.

4.2.8 Bangladesh Banks Heist

One of the major financial crimes accomplished online this year took place in February when about $81 million of Bangladesh's money was siphoned off by hackers that remain unknown. According to multiple reports, the money was successfully shipped to parts of Asia, Philippines, and Sri Lanka. A group of hackers penetrated the Bangladesh Bank system successfully in order to steal reserves.

By making a spelling error that tipped off the bank, however, they caused about $870 million in transfers to be canceled. The software bug involved in this funds cancellation entered the system with the $81 million the thieves successfully stole. The anomaly in the system broke the process for automatic printing, and it was many days later that the transfer receipts were even discovered. Needless to say, this was enough time for the hackers to cover their tracks.

4.2.9 Satellite Failure Sends Global Software for a Toss

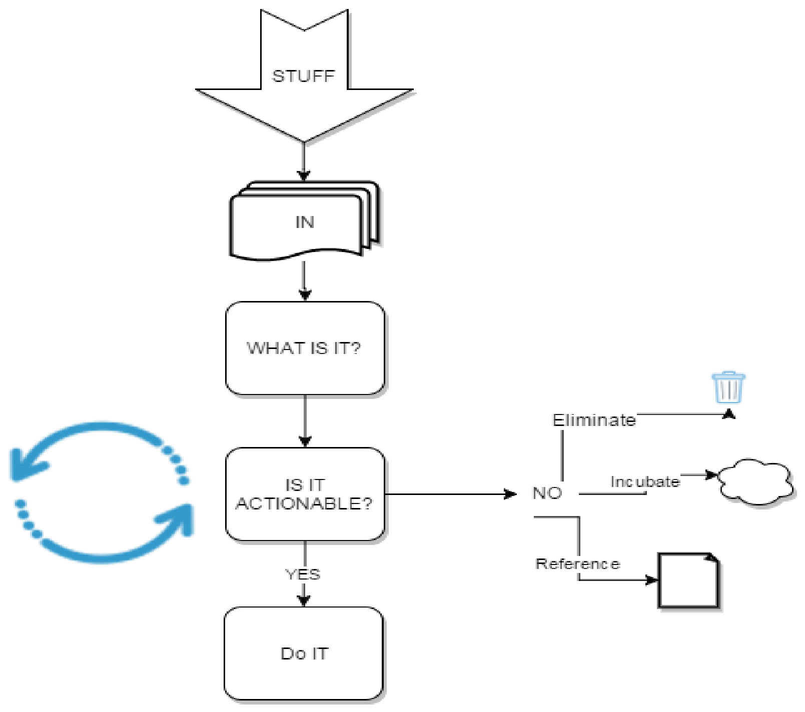
The catastrophe of a 25-year-old satellite which failed this past January sparked a software bug that lasted for a mere 13 microseconds (0.000013 of a second). Even though the error was momentary, it resulted in astonishingly enormous consequences across global positioning systems. According to reports, “The rule of thumb is that for every nanosecond of error, you could be out by as much as a foot…an error of 13 microseconds or 13,000 nanoseconds works out as just under four kilometers.” All over the world, GPS systems were thrown off for several hours before the operations were once again restored to normalcy. Systems such as select radio stations took numerous days to be completely reinstated.

4.3 Conclusion

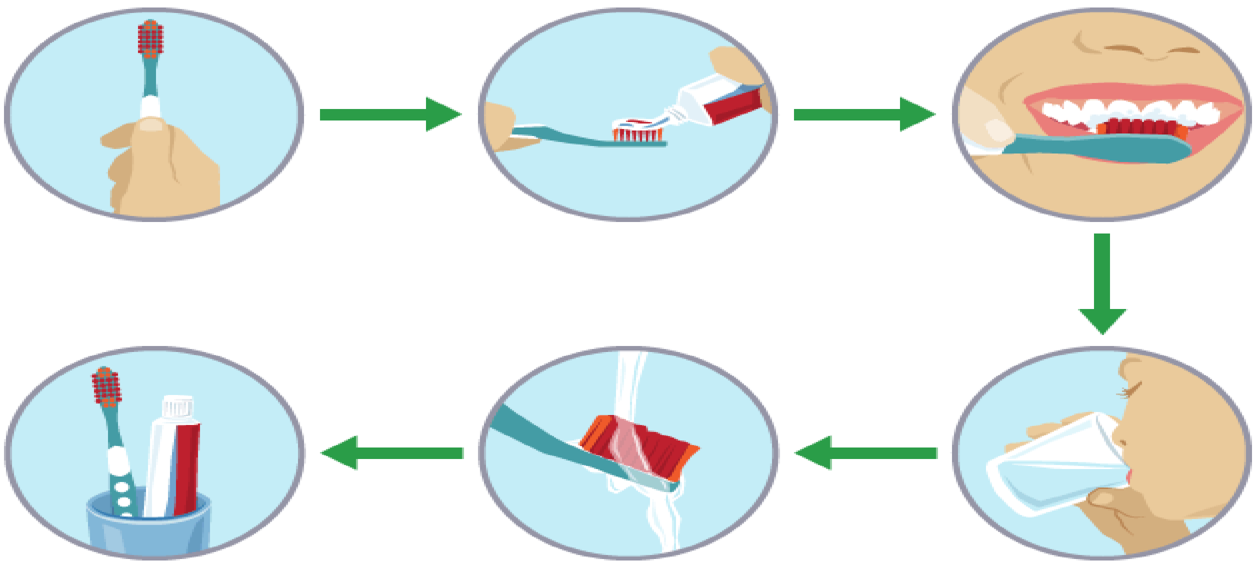
5. Process definition

As per the classic definition, a process is a series of actions or steps taken in order to achieve a particular end (https://en.oxforddictionaries.com/definition/process). This can easily be related to almost everything in one’s life. This becomes even clearer when we understand what composes a process:

- States - Among the most pervasive notions in computing is that of "state". We define the state of a computation as a set of information sufficient to determine the future possible behaviors. In other words, the state tells us how the system can change. It also can tell us something about what has happened in the past, if we know it to have been started in a particular initial state. It doesn't usually tell exactly what has happened, but rather conveys some abstraction of what has happened.
- Transitions - To apply relations to the discussion of states, the set of pairs of states (s, s') such that one move will take the system from s to s' is called the transition relation. Typically we will represent a transition relation by =>. A single pair (s, s') such that s => s’ is called a transition. We sometimes say that s' is a successor of s when there is a transition (s, s') and that s is a predecessor of s'.
  (*Computer Science: Abstraction to Implementation Robert M. Keller Harvey Mudd College keller@cs.hmc.edu September 2001*)



A real life process implementation that we use daily:

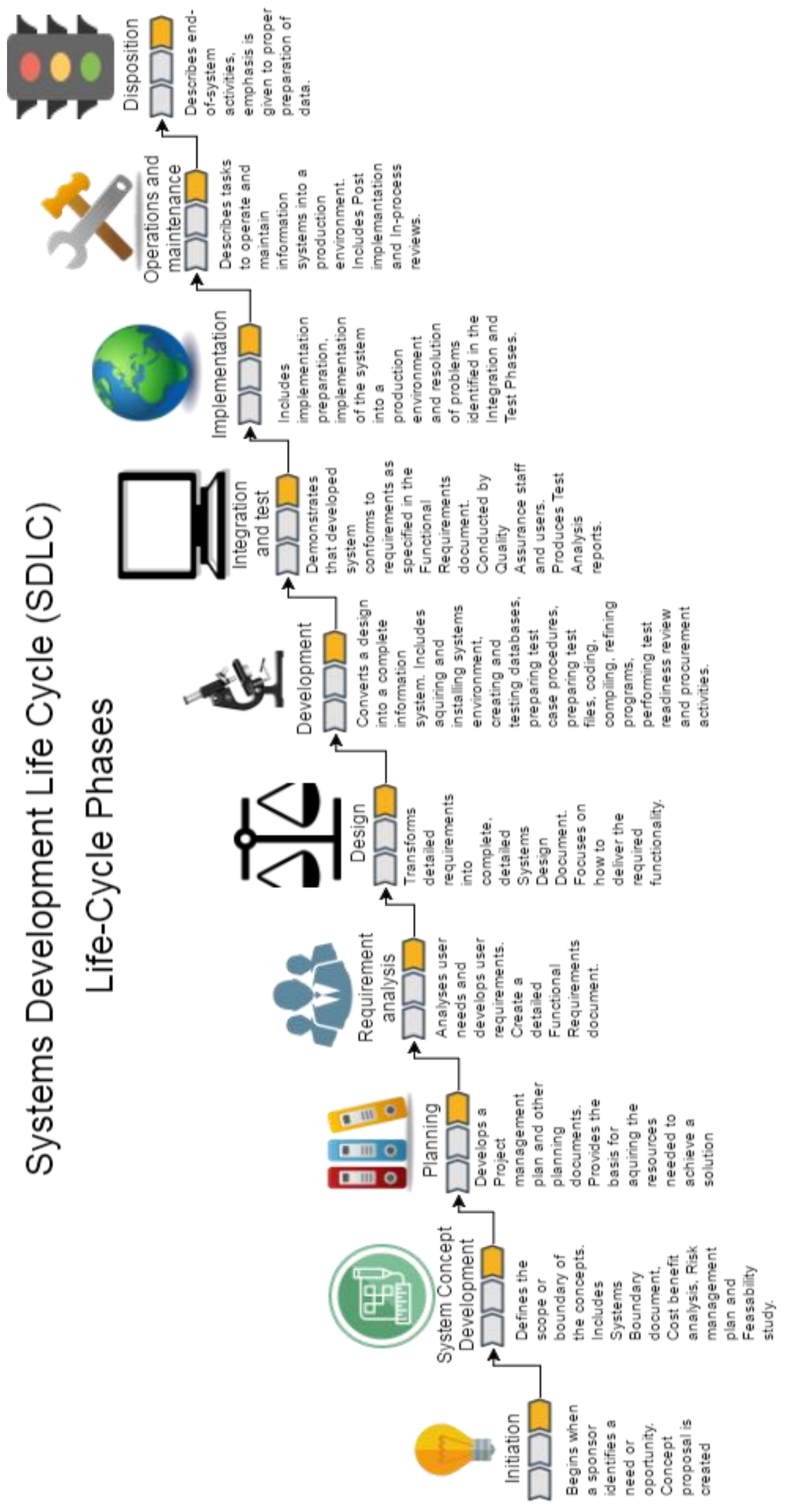


Source (http://www.bbc.co.uk/education/guides/zqh49j6/revision)

6. Overview of the Development Lifecycle (SDLC)

6.1 The SDLC

The System Development Lifecycle, associated with the traditional development process, is a staged collection of states and transitions. To be thorough, the below diagram shows the full development staged process that is employed to develop software from scratch.



Begins when a sponsor identifies a need or opportunity. Concept proposal is created

Defines the scope or boundary of the concepts. Includes Systems Boundary document, Cost benefit analysis, Risk management plan and Feasability study.

Develops a Project management plan and other planning documents. Provides the basis for aquiring the resources needed to achieve a solution

Analyses user needs and develops user requirements. Create a detailed Functional Requirements document.

Transforms detailed requirements into complete, detailed Systems Design Document. Focuses on how to deliver the required functionality.

Converts a design into a complete information system. Includes aquiring and installing systems environment, creating and testing databases, preparing test case procedures, preparing test files, coding, compiling, refining programs, performing test readiness review and procurement activities.

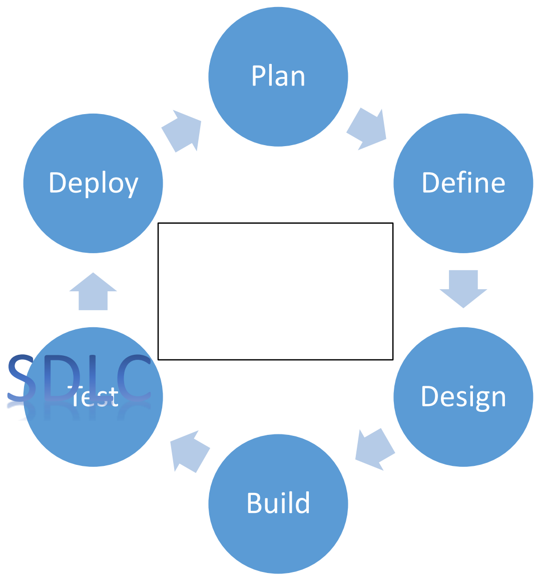
Demonstrates that developed system conforms to requirements as specified in the Functional Requirements document. Conducted by Quality Assurance staff and users. Produces Test Analysis reports.

Includes implementation preparation, implementation of the system into a production environment and resolution of problems identified in the Integration and Test Phases.

Describes tasks to operate and maintain information systems into a production environment. Includes Post implemantation and In-process reviews.

Describes end-of-system activities, emphasis is given to proper preparation of data.

A more simplified approach, defines a circular process consisting of less phases and transitions, summarizes a couple the steps depicted above (Requirement analysis, Design, Development, Integration and test) but can also be easily extended to both Implementation and Operations & Maintenance.



Following are the most important and popular SDLC models followed in the industry:

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

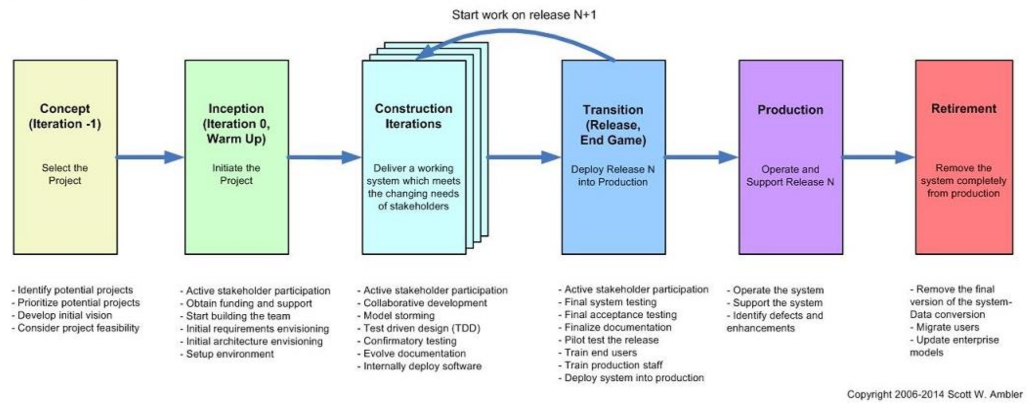
Strength and Weaknesses of SDLC	
Strengths	**Weaknesses**
Control	Increased development time
Monitor large projects	Increased development cost
Detailed steps	Systems must be defined up front
Evaluate costs and completion targets	Rigidity
Documentation	Hard to estimate costs, project overruns
Well defined user input	User input is sometimes limited
Ease of maintenance	
Development and design standards	
Tolerates changes in staffing	

Source (https://en.wikipedia.org/wiki/Systems_development_life_cycle#cite_note-Post.2C_G._2006-14)

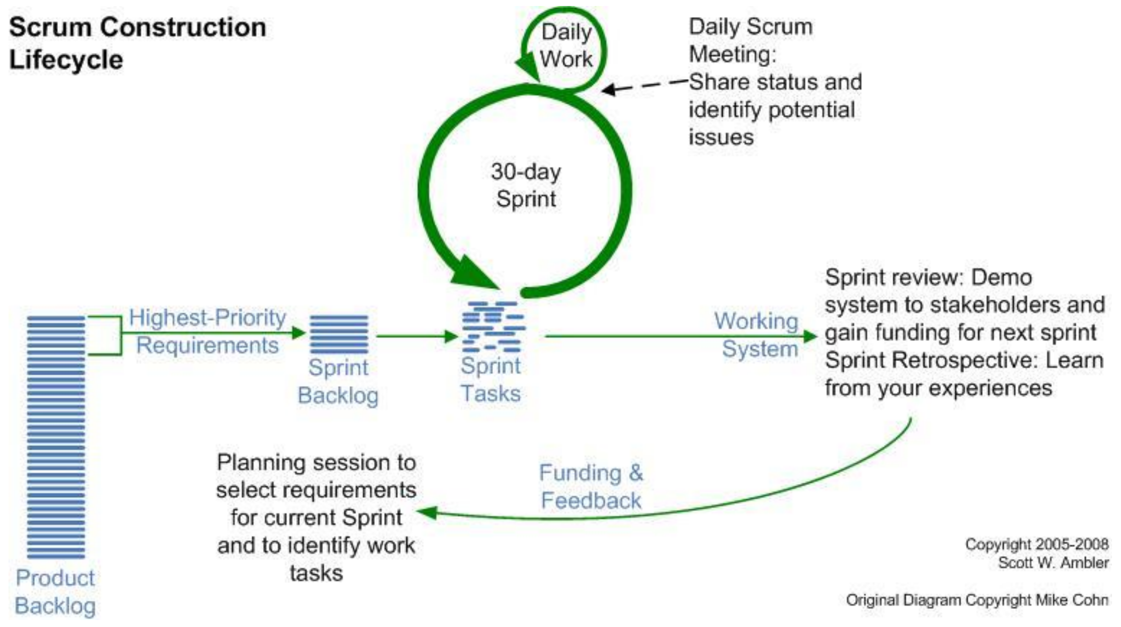
5.2 The ADLC

The AGILE Development Lifecycle is more complex project because a cyclic transition enhances the overall process. Normal linear transitions follow until project retirement.

Figure 5. The Agile SDLC (high-level).


Start work on release N+1
Concept (Iteration -1)
Select the Project
Inception (Iteration 0, Warm Up)
Initiate the Project
Construction Iterations
Deliver a working system which meets the changing needs of stakeholders
Transition (Release, End Game)
Deploy Release N into Production
Production
Operate and Support Release N
Retirement
Remove the system completely from production
- Identify potential projects
- Prioritize potential projects
- Develop initial vision
- Consider project feasibility
- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements envisioning
- Initial architecture envisioning
- Setup environment
- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Confirmatory testing
- Evolve documentation
- Internally deploy software
- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production
- Operate the system
- Support the system
- Identify defects and enhancements
- Remove the final version of the system-Data conversion
- Migrate users
- Update enterprise models
Copyright 2006-2014 Scott W. Ambler

The AGILE Development Lifecycle has a shorter ramp up period, before starting to deliver functional software to the end user. Additional increments are added to the initial delivery until the final product, as envisioned by the end user, is deployed into the production environment.


Scrum Construction Lifecycle
Daily Work
Daily Scrum Meeting: Share status and identify potential issues
30-day Sprint
Highest-Priority Requirements
Sprint Backlog
Sprint Tasks
Working System
Sprint review: Demo system to stakeholders and gain funding for next sprint
Sprint Retrospective: Learn from your experiences
Planning session to select requirements for current Sprint and to identify work tasks
Funding & Feedback
Product Backlog
Copyright 2005-2008 Scott W. Ambler
Original Diagram Copyright Mike Cohn

The Scrum (popular AGILE model) construction life cycle of Figure 1, although attractive proves to be clearly insufficient in practice.

Where does the product backlog come from? Does it get beamed down from the Starship Enterprise? Of course not, it's actually the result of initial requirements envisioning early in the project. You don't only implement requirements during an iteration, you also fix defects (disciplined agile teams, particularly working at scale, may have a parallel testing effort during construction iterations where these defects are found), go on training, support other teams (perhaps as reviewers of their work), and so on. So you really need to expand the product backlog into a full work items list. You also release your system into production, often a complex endeavor.

Source: (http://www.ambysoft.com/essays/agileLifecycle.html#Development)

7. Software tester's purpose in software development

6.1 Role in SDLC

Software Testing is a process that must be planned, specified, designed, implemented and quantified. Testing is a dual purpose process, as it is used to detect bugs as well as to establish confidence in the quality of software. The testing process divided into a well-defined sequence of steps is termed as STLC. STLC involves the testers at early stages of development.

STLC consists of the following phases:

- Test Plan
- Test Design
- Test Execution
- Post Execution/Test Review

Test Plan: - The main goal of this phase is to consider the important issues of testing strategy like resources, schedules, responsibilities, risks and priorities as a roadmap. The output of this phase is test plan document. Test plans are developed every phase of SDLC.

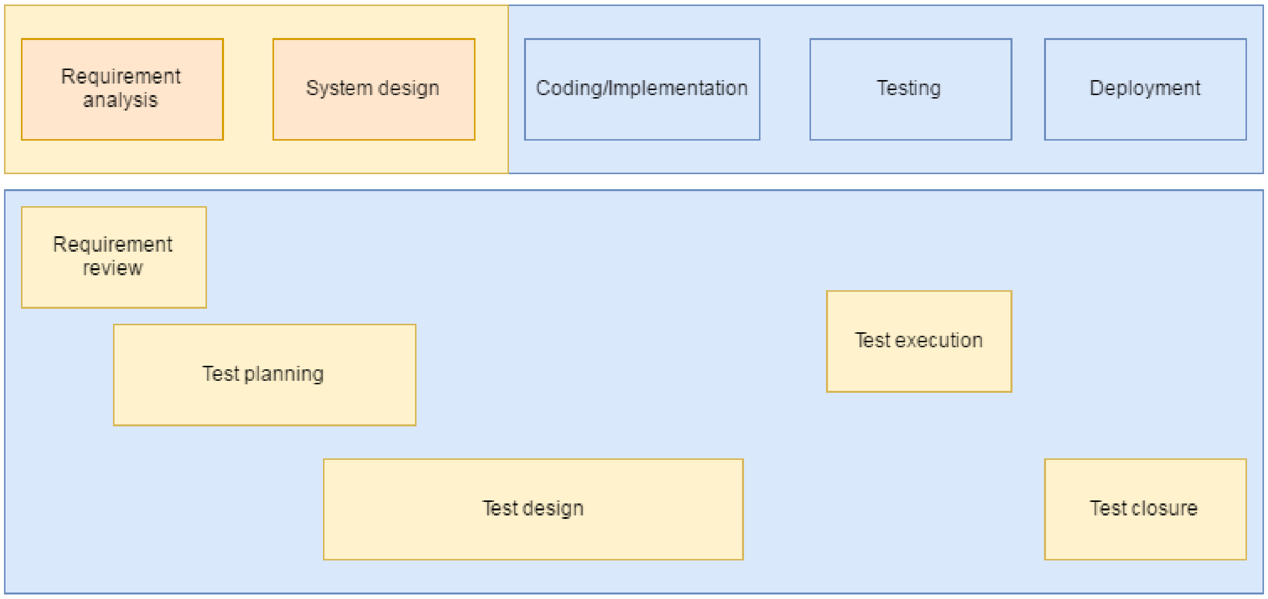
Test Design: - In this phase Test Cases are designed. These include the following activities:

- Determine the test objectives
- Preparation of list of items to be tested
- Identification of test cases.
- Selection of test case design techniques.
- Creating test cases and test data
- Setting up the test environment and supporting tools.
- Creating test procedure specification.

Test execution: - In this phase, all the test cases are executed including verification and validation. Verification test cases are started at the end of each phase of SDLC. Validation test cases are started after the completion of a module. Test results are documented in the test incident reports, test logs, testing status and test summary reports.

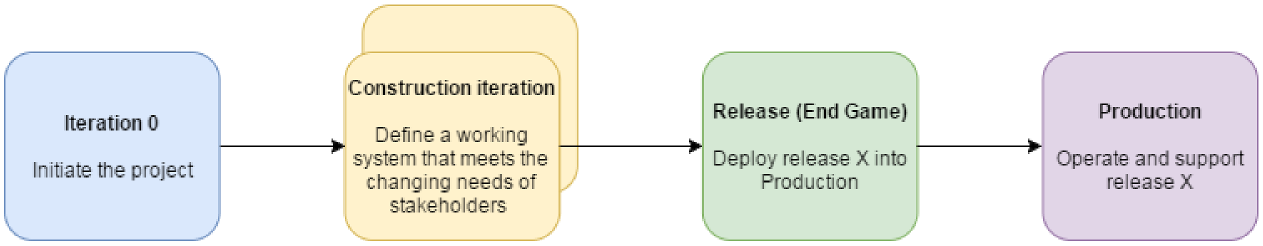
Post Execution/ test review: - After the successful test execution, bugs will be reported to the developers. This phase is to analyze bugs, related issues and get feedback so that maximum number of bugs can be fixed. After fixing the bug, the developer reports to the testing team and the modified portion of software is tested once again. The final evaluation reports are reviewed and analyzed for the overall testing process.

Source: International Journal of Recent Research Aspects (http://ijrra.com/)



6.2 Role in ADLC

The role shifts a little more in the AGILE Development Lifecycle by encouraging closer stakeholder relations as well as also improving the technical coding skills. Activity generally starts after Iteration 0 kicks off but prior to its completion.


Iteration 0
Initiate the project
Construction iteration
Define a working system that meets the changing needs of stakeholders
Release (End Game)
Deploy release X into Production
Production
Operate and support release X

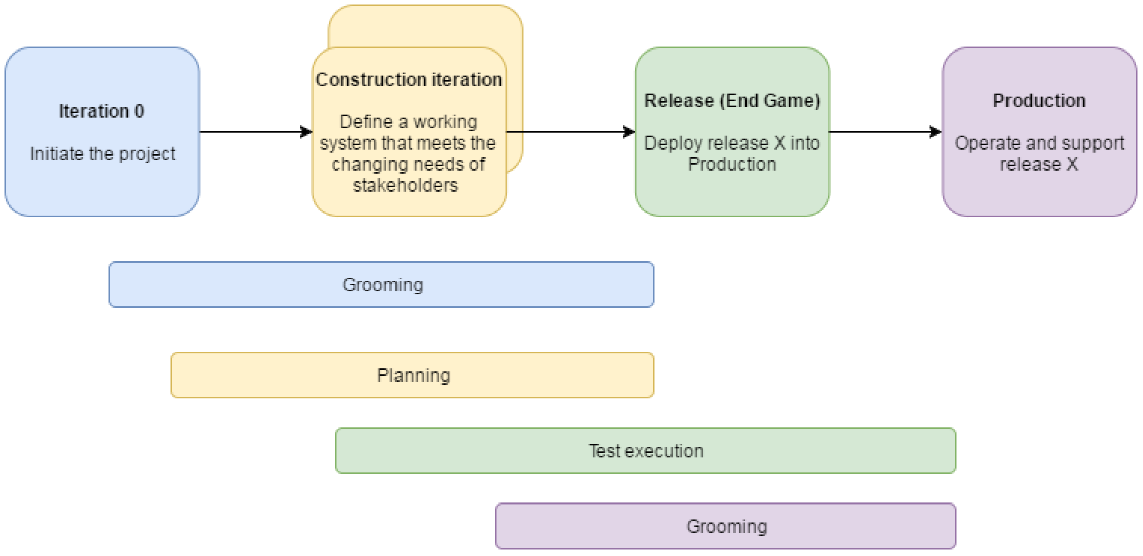
In general, the iteration activities are the following:

Grooming – clarification of requirements. Addition of enough requirement acceptance criteria so that it gains stakeholder satisfaction upon successfully delivery.

Planning – Committing on delivering refined requirements in the next iteration based on team capacity

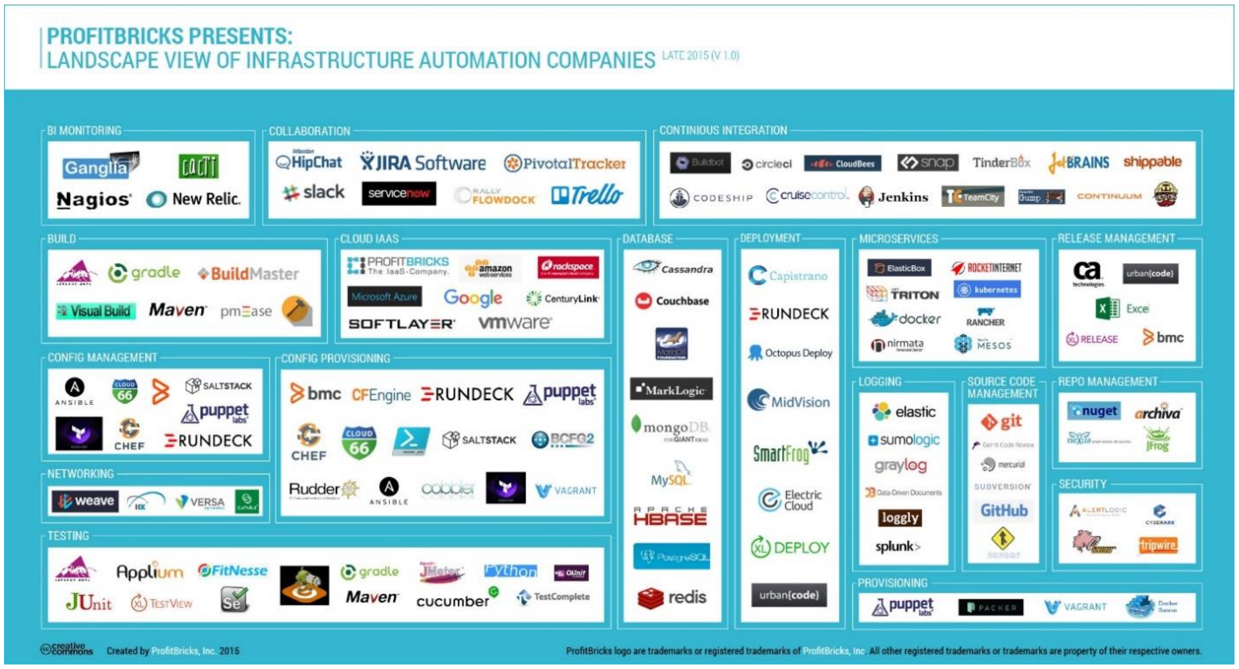
Test execution – Validation of refined requirements against requirement acceptance criteria.

Retrospectives – Closure activity for the iteration. Definition of lessons learned and follow-up on implementing the items identified in the “Improvement” section.


Iteration 0
Initiate the project
Construction iteration
Define a working system that meets the changing needs of stakeholders
Release (End Game)
Deploy release X into Production
Production
Operate and support release X
Grooming
Planning
Test execution
Grooming

8. Tools to support the SDLC

Each profession involved in the SDLC has collated a series of tools meant to enable and support their respective activities. It would be inhuman to know them all. Each tool has been adopted within multiple companies thus enabling specialization in specific SDLC support software.


PROFITBRICKS PRESENTS:
LANDSCAPE VIEW OF INFRASTRUCTURE AUTOMATION COMPANIES LATE 2015 (V 1.0)
BI MONITORING
COLLABORATION
CONTINIOUS INTEGRATION
BUILD
CLOUD IAAS
DATABASE
DEPLOYMENT
MICROSERVICES
RELEASE MANAGEMENT
CONFIG MANAGEMENT
CONFIG PROVISIONING
LOGGING
SOURCE CODE MANAGEMENT
REPO MANAGEMENT
NETWORKING
SECURITY
TESTING
PROVISIONING
Created by ProfitBricks, Inc. 2015
ProfitBricks logo are trademarks or registered trademarks of ProfitBricks, Inc. All other registered trademarks or trademarks are property of their respective owners.

Out of these many, several tools will be used to support the practical exercises as well as excerpts for the course.