GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY

BENEFITS OF GOOD QUALITY SOFTWARE AND HARMFUL CONSEQUENCES OF POOR-QUALITY SOFTWARE FOR ORGANIZATIONS AND INDIVIDUALS: ROLE OF SOFTWARE ENGINEERING PROFEFESSIONALS.

SYNDICATE - 18

<u>DS COMMENT</u>

COVER SHEET

1. Topic - Benefits of good quality software and harmful consequences of poor-quality software for organizations and individuals: role of software engineering professionals.

2. Academic DS - Dr. LP Kalansooriya

3. Military DS - Lt PGDPMK Palliyaguru

4. Syndicate Leader - DANR Dissanayake

5. Syndicate Members -

S/N	**Svc NO**	**Rank**	**Name**		**Stream**
1.	5794	SCQMS	DANR	Dissanayake	MTS
2.	5863	C/CPL	KMM	Lakshan	ENG
3.	5984	O/C	RGN	Chathuranga	SE
4.	5986	O/C	AK	Kavindaya	SE
5.	5985	O/C	RMDKN	Rathnayake	SE
6.	5983	O/C	HHAMI	Hettiarachchi	SE
7.	5982	O/C	VI	Samarasinghe	SE
8.	5825	L/O/C	JMIB	Dissanayake	SS
9.	5872	O/C	WAC	Jeewantha	ENG
10.	5935`	L/O/C	MN	Thishera	MBBS

Supervised by:

Date: ………………………. ………………………………

Dr. LP Kalansooriya
(Academic DS)

Date: ………………………. ……………………………….

Lt PGDPMK Palliyaguru
(Military DS)

<u>DECLARATION</u>

We declare that this does not incorporate, without acknowledgment, any material previously submitted for a degree or a diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published and written by another person or our self except where due to reference is made in the text. We also hereby give consent for our dissertation, if accepted, to be made available for photocopying and interlibrary loans, and the title and for the title summary to be made available to an outside organization.

S/N	**Svc NO**	**Rank**	**Name**		**Sign**
1	5794	SCQMS	DANR	Dissanayake	…………….
2	5863	C/CPL	KMM	Lakshan	…………….
3	5982	O/C	VI	Samarasinghe	…………….
4	5983	O/C	HHAMI	Hettiarachchi	…………….
5	5984	O/C	RGN	Chathuranga	…………….
6	5985	O/C	RMDKN	Rathnayake	…………….
7	5986	O/C	AK	Kavindaya	…………….
8	5825	L/O/C	JMIB	Dissanayake	…………….
9	5872	O/C	WAC	Jeewantha	…………….
10	5935	L/O/C	MN	Thishera	…………….

ACKNOWLEDGEMENT

No one walks alone in the journey of life. We would like to express our deep gratitude to Lt PGDPMK Palliyaguru for his advice and assistance in keeping our progress on schedule. We would also like to thank our academic DS Dr. LP Kalansooriya for his patience, guidance, enthusiastic encouragement, and useful criticisms provided in this case study. We take this opportunity to acknowledge the service of the library and the training support officer who collaborated in producing this work. The assistance, cooperation, and experience of our fellow undergraduates were essential for the completion of our presentation. Finally, we wish to thank our parents, family members, and loved ones for their support and encouragement throughout this study and syndicate presentation.

ABSTRACT

Software industry plays a significant role in the economic, industrial, educational and many other sectors. This script based all the areas on software usages by organizations and individuals deviates according to the quality of them. According to the software engineering international standards ISO/IEC there are some measurements to deviate whether a software is good or bad. As per that deviation there will be benefits as well as harmful consequences by them. The traditional methods which were used by mankind were not sufficient enough to fulfil these requirements of the new technological era. Thus, new automated methods were introduced. These new approaches satisfied the world requirements and also provided employment opportunities to billions of people. In this sheet we have emphasized all aspects and examples for software quality and how they affect organizations and individuals in separated defined subtopics. We have managed our resources well to elaborate the whole spread idea about this content.

AIM

This study aims to explore options available good and bad impacts to organizations and individuals due to quality of software accordingly. Further, the study aims to analyze pure standards of software quality and usages all over the world.

CONTENT

CHAPTER ONE

INTRODUCTION

1. Software is a set of instructions for a computer. The total set of programs, operations, and routines related with the operation of a computer system is referred to as software. The phrase was coined to distinguish these instructions from the physical components of a computer system, which are known as hardware. A program, or software program, is a set of instructions that tells a computer's hardware how to complete a task. System software and application software are the two basic forms of software. System software, which is primarily controlled by an operating system, regulates the internal functioning of a computer as well as peripherals such as monitors, printers, and storage devices. Application software, on the other hand, instructs the computer to carry out user requests and can be defined as any program that processes data for a user. Word processors, spreadsheets, database management, inventory and payroll tools, and a variety of other "applications" are all examples of application software. Network software is a third type of software that organizes communication between computers connected to a network.

2. Software is often saved on a hard drive or magnetic diskette, which is an external long-term memory device. The computer reads the program from the storage device and temporarily stores the instructions in random access memory when it is in use (RAM). Running or executing a program refers to the process of storing and then carrying out instructions. Firmware, or "hard software," refers to software programs and operations that are permanently stored in a computer's memory utilizing read-only (ROM) technology.

3. A collection of instructions and data that tells a computer how to work is known as software. This contrasts with hardware, which is what the system is made of and does the real job. All information handled by computer systems, including programs and data, is referred to as software in computer science and software engineering. Programs, libraries, and related non-executable data, such as online documentation or digital media, are all examples of software. Both software and hardware are interdependent, and neither can be used effectively on its own.

4. Executable code is made up of machine language instructions that are supported by a single processor—typically a central processing unit (CPU) or a graphics

processing unit (GPU) (GPU). Machine language is made up of sets of binary values that represent processor instructions that modify the computer's state from its previous one. An instruction, for example, may change the value stored in a specific storage area in the computer—an effect that is not immediately visible to the user.

5. An instruction can also cause state changes that should be visible to the user by invoking one of numerous inputs or output procedures, such as displaying text on a computer screen. Unless the processor is told to "jump" to a new command or the operating system interrupts it, the processor executes the instructions in the order they are given. As of 2015, most personal computers, smartphone devices, and servers include processors with multiple execution units or several processors working in tandem, making computing much more concurrent than in the past. High-level programming languages are used to write the majority of software. Because they are closer to natural languages than machine languages, they are easier and more efficient for programmers. A compiler, interpreter, or a combination of the two is used to transform high-level languages into machine language. Low-level assembly languages, which have a strong correlation to the computer's machine language instructions and are converted into machine language via an assembler, can also be used to write software.

OBJECTIVES

The objectives of this study are as follows.

a) To identify the importance of Software to whole word.

b) To identify the bad impacts of poor-quality software and good impacts of good quality software

c) How to enhance the productivity of software by using strategies.

CHAPTER TWO

IMPORTANCE OF SOFTWARE DAY TODAY LIFE USAGE CAPACITY OF SOFTWARE AS ORGANIZATIONS AND INDIVIDUALS

1. Back in the olden days before there was a computer under every rock if you wanted to design a circuit to accomplish something you had to do just that; design a circuit to accomplish that thing. When you were finished the circuit would do that one thing and nothing else.

2. Once computers were introduced it became possible to design circuitry that could accomplish many different tasks, all you had to do is write a program. Programs are much easier to create and modify than electronic circuitry and provide a means, in large measure, to separate functionality from physical implementation. Computer software, or just software is a general term used to describe a collection of computer programs, procedures and documentation that perform some tasks on a computer system. Today software is everywhere, your mobile, TV, computer everything runs on software.

3. You order an ambulance, and it stands at your doorstep in like 15 min, how do you think is this possible??Is it even imaginable without software, NO.

4. Software not only help us during leisure but is also important in various ways. In such an appliance-oriented world, how can you live without such simple but important software that are very useful in daily life.

5. The streetlights during your commute were likely programmed for maximum throughput and minimum traffic using software. Maybe you used software when you used GPS to find a fast route to wherever you might have gone to lunch or to the doctor.

6. When in the past people used to use post for send their messages. But now it changed a lot and if we need to send a message, we just need to take our mobiles out. Even though it is not taking time as using post. It takes a time of a tap.

7. In the past our parents used to pay bills by going to the bank. Also, if they needed to do a bank transaction, they needed to be present at the bank. But because of the adaptation of the software, it comes to the fingertip. We can do our all the payments and every other transaction using our devices using the software.

8. As we see in the present, because of the pandemic time the education sector stepped in to the online education. Because of that all the parents begin to use the online devices. It means they need to know how to use this software for the education of their children.

9. In the early days people use to wake up by the roosters crowing and the mechanical alarm clocks. But by the time, in the present we use the digital alarm clocks and even they are installed into the mobile phones too.

CHAPTER THREE

BENEFITS OF GOOD QUALITY SOFTWARE

The software quality

10. It is the extent to which the appropriate software was produced. Quality software is free of bugs and defects, delivered on time and on budget, meets requirements and expectations, and is maintainable. Quality is defined by ISO 8402-1986 as "the totality of characteristics and features of a product or service that bear its ability to implied needs or Satisfy stated." Quality, in the original sense, is difficult to define but can be recognized when it exists.

11. Software quality is defined as a field of study and practice that describes the desirable attributes of software products. There are two main approaches to software quality

a. Defect Management

A software defect can be regarded as any failure to address end-user requirements. Common defects include missed or misunderstood requirements and errors in design, functional logic, data relationships, process timing, validity checking, and coding errors.

The software defect management approach is based on counting and managing defects. Defects are commonly categorized by severity, and the numbers in each category are used for planning. More mature software development organizations use tools, such as defect leakage matrices and control charts, to measure and improve development process capability.

b. Quality Attributes

This approach to software quality is best exemplified by fixed quality models, such as ISO/IEC 25010:2011. This standard describes a hierarchy of eight quality characteristics each composed of sub-characteristics

1. Functional suitability
2. Reliability
3. Operability
4. Performance efficiency
5. Security
6. Compatibility
7. Maintainability
8. Transferability

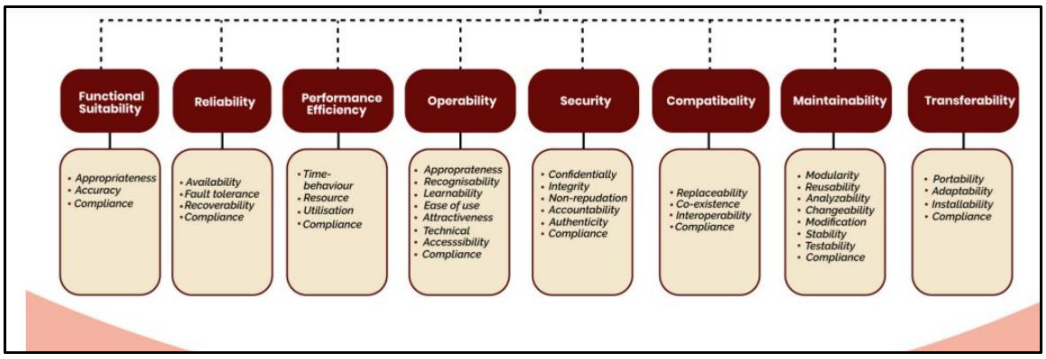

Figure 3.1

Additionally, the standard defines a quality-in-use model composed of five characteristics:

1. Effectiveness
2. Efficiency
3. Satisfaction
4. Safety
5. Usability

There are three widely accepted models when it comes to measuring software quality

1. <u>McCall's Quality Model</u>
   Mc Call's model was first introduced in the US Airforce in the year 1977. The main intention of this model was to maintain harmony between users and developers.

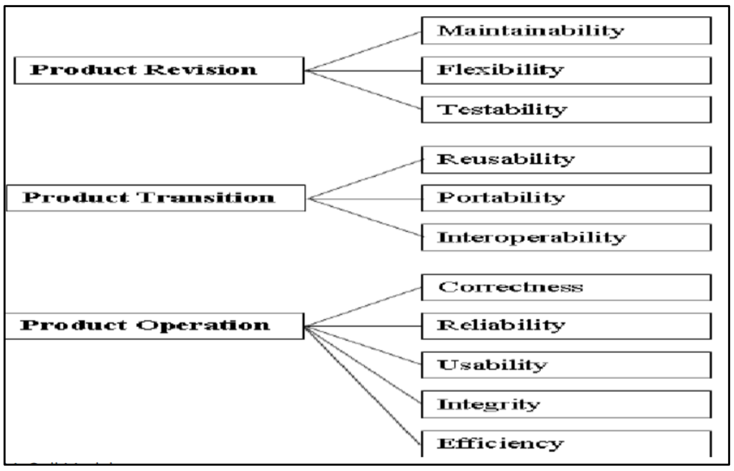

Figure 3.2

2. Boehm quality model

Boehm model was introduced in the year 1978. It was a kind of hierarchical model that's structured around high-level characteristics. Boehm model measures software quality based on certain characteristics.

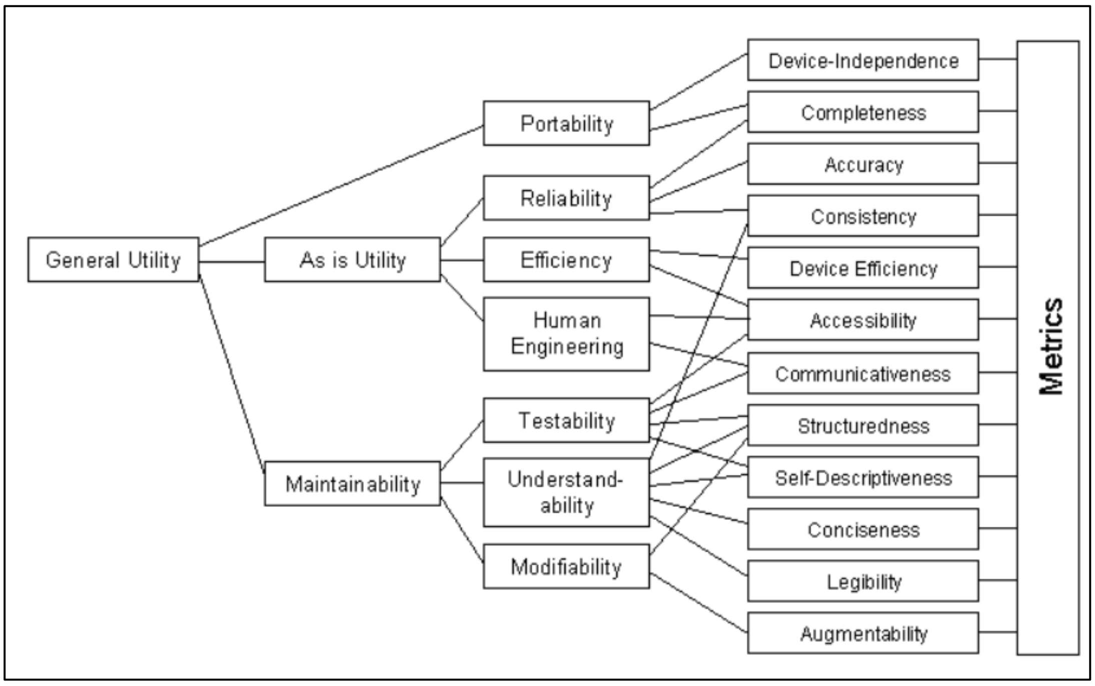

Figure 3.3

3. Dromey's quality model

Dromey's model is mainly focused on the attributes and sub-attributes to connect properties of the software to the quality attributes. There are three principal elements to this model

a) Product properties that affect the quality
b) High-level quality attributes
c) Linking the properties with quality attributes

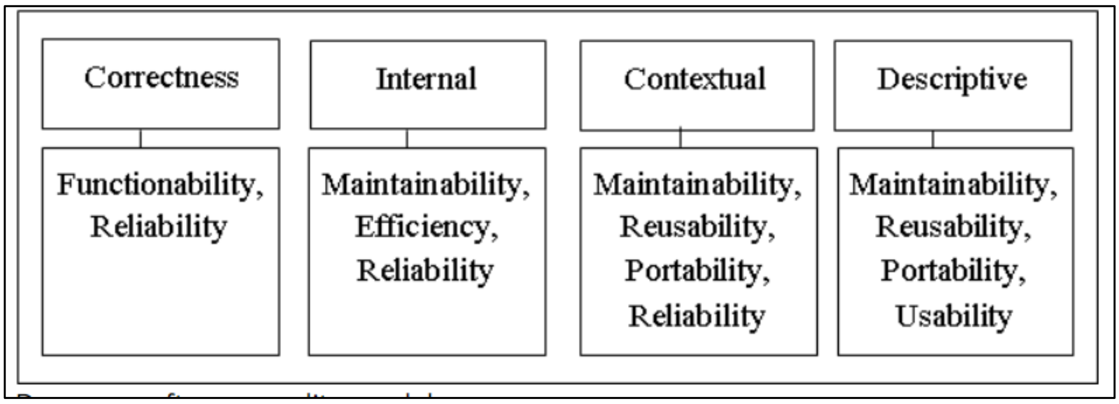
Correctness
Functionability, Reliability
Internal
Maintainability, Efficiency, Reliability
Contextual
Maintainability, Reusability, Portability, Reliability
Descriptive
Maintainability, Reusability, Portability, Usability

Figure 3.4

The ISO/IEC 25000 series of standards

The series of standards ISO/IEC 25000, also known as Square (System and Software Quality Requirements and Evaluation), has the goal of creating a framework for the evaluation of software product quality.

ISO/IEC 25000 is the result of the evolution of several other standards.

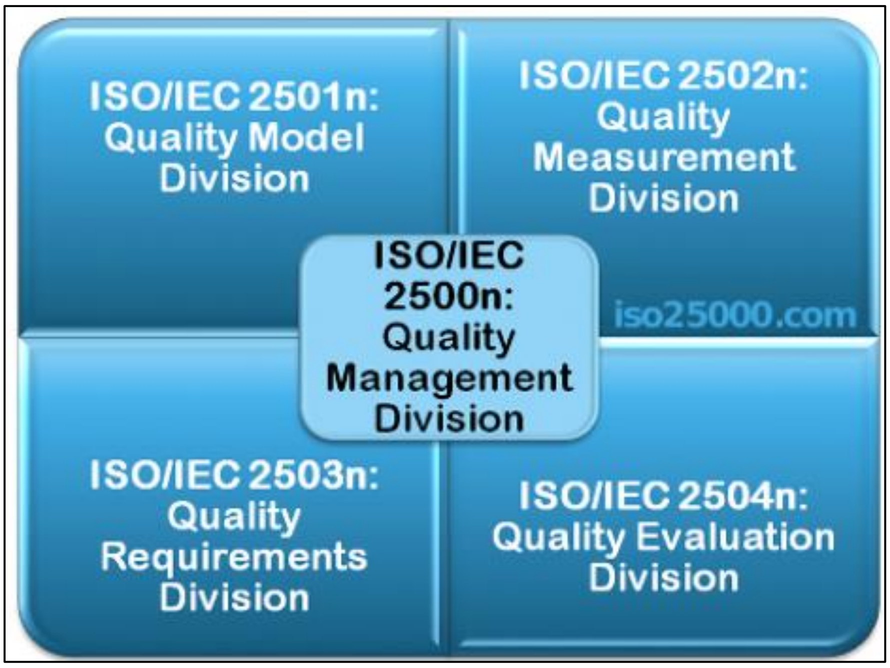
ISO/IEC 2501n: Quality Model Division
ISO/IEC 2502n: Quality Measurement Division
ISO/IEC 2500n: Quality Management Division
iso25000.com
ISO/IEC 2503n: Quality Requirements Division
ISO/IEC 2504n: Quality Evaluation Division

Figure 3.5

12. Quality, in the user's opinion, is fitness for purpose or meeting the user's needs. In the manufacturing industry, quality means adhering to process standards. The following are the most important aspects of quality for the customer:

a) Good design – both in terms of appearance and feel
b) Longevity – lasts as long as
c) Consistency
d) Reliable – a reasonable number of breakdowns/failures
e) Excellent functionality
f) Excellent after-sales service
g) Price-to-value

Process of Software Quality

13. Quality assurance, quality planning, and quality control are the three main components of software quality management activities.

Quality Control

14. Set up an organized and logical set of organizational processes for deciding on software development standards in conjunction with regulatory processes; this gives you a better chance of producing high-quality software.

15. This stage may include the following elements:

a) Identifying any standards that may be used in the software development process.

b) To carry out traditional processes such as quality reviews.

c) Carry out procedures for recording data from in-process tests.

d) Encouraging process documentation standards.

Quality Planning

16. Quality planning is the process of defining the quality attributes associated with the project's output. How were those characteristics evaluated? The software development project has characteristics such as "robustness," "accessibility," and "modularity." The quality plan may also include information about the intended market, critical release dates, quality goals, anticipated risks, and risk management policies.

Quality Assurance

17. The quality assurance team tests and reviews software to ensure that quality assurance processes and standards are met at both the organizational and project levels. When software development firms implement the agile quality approach, the transition from a more formal quality management structure to Agile methods can cause problems if control procedures are not appropriately adapted. Among the activities are:

a) A follow-up review of software to ensure that any required changes outlined in previous testing are addressed.

b) Software release testing with proper documentation of the testing process

c) For evaluation, use software measurement and metrics.

d) Examine software and associated documentation for non-conformance with standards.

18. Benefits of Ensuring Software Quality

a) The development team's productivity has increased.

b) Product quality has improved because test statistics and defect tracking are more precise and up to date.

c) Reduced rework costs as defects are discovered earlier in the software project development life cycle at every stage.

d) Increased trust in current product management and future product development.

e) Increased credibility because the software produced will be of the highest quality.

f) This saves money.

g) Instills trust in the client.

h) Maintains a positive user experience.

i) Increases Profitability.

j) Increases customer satisfaction.

<u>Importance of Software Quality Management</u>

19. Software quality must be a requirement for a successful software development business; it cannot be an exception. Consider the various ways in which software quality can affect business.

<u>Predictability</u>

20. Predictability is driven by software quality. Predictability decreases as the amount of rework and lower quality product increases. If you do it correctly the first time, there will be less variation in productivity, less rework, and overall better performance. Products are shipped on time. It is far more difficult to manage poor quality.

<u>Reputation</u>

21. A significant, solid reputation is difficult to establish and easy to lose, but it is a powerful business driver when the company has it. With a few blunders, fame can vanish, posing significant challenges to sales and, as a result, your bottom line.

<u>Employee Satisfaction</u>

22. Employees who are the happiest and most productive take pride in their work. Allowing employees to create software will boost productivity and morale. Poor quality products, extensive rework, dissatisfied customers, and difficulty meeting deadlines have the opposite effect, resulting in a less productive workforce and high turnover.

<u>Customer Contentment</u>

23. A high-quality product pleases the customer. A happy customer returns to provide positive referrals. Customer loyalty is heavily influenced by the quality of the software produced as well as the service offered. Positive references can spread quickly on social media platforms like Facebook and Twitter. Poor quality and dissatisfaction can also communicate quickly, if not faster than good ones.

<u>In conclusion</u>

24. A successful software business is built on predictable and productive performance, happy employees, a stellar reputation, and satisfied customers. Everything contributes to the bottom line. Quality has an impact on many aspects of software development projects.

25. **<u>Key Software Quality Tools</u>**

a) Selenium (Web Application Testing)
b) Robot Framework (Acceptance Testing)
c) Appium (Mobile Testing)
d) JMeter (Load Testing)
e) Jenkins (Continuous Testing)
f) Postman/Robot framework (API Testing)
g) Firebug / Firepath (Online Debugging)
h) GitLab (Project & Source Code Hosting)
i) Trello, Wekan board (Defect Tracking & Collaboration)
j) UIAutomator (Galen Framework)
k) Pycharm, Eclipse (Source code Editor), etc

<u>How to Improve Software Quality</u>

26. It is critical to reduce testing costs while also improving software quality. These four points will help to improve software quality and testing efficiency.

Testing time

27. By testing earlier, defects can be detected and resolved rather than having to wait until the end of the process. The more software bugs that are discovered, the longer and more expensive it is to fix them. Involved testers in the requirements and design phases so that they could contribute to the creation of a more useful framework. More than 70% of problems in a live environment can be traced back to poor conditions.

Enhance testing organization

28. Implement specific policies to ensure consistency, such as using repeatable industry standard testing processes and training testers within this framework.

Continue to review

29. Just because certain methods have worked in the past does not guarantee that they will continue to work in the future. Evaluation and refactoring processes enable the testing team to maximize efficiency by reviewing what worked well. Implement a Cause Analysis process to determine whether issues were caused by a ‘testing miss,' a ‘development miss,' or a ‘requirements or design miss.' Identify opportunities for improvement throughout the software development process.

Improvement results from innovation.

30. Don't get stuck in the same old routine; instead, try something new.

Best Practices for Software Quality

Determine the Importance of Quality and Develop a Quality Assurance Process.

31. A first step toward preventing future programming errors. The development approach, as well as high-quality design and code, necessitate the involvement of the QA team. To avoid failures in end-product quality, the input and output of the QA process must be well defined, planned, and documented.

Investigate Quality Benchmarks.

32. Gathering requirements is an important part of the development cycle as well as quality assurance. Everything includes the application's features, design, functionalities, scalability, reliability, efficiency, usability, and so on to determine whether the application will meet the quality benchmarks.

Adhere to the 'Test Early, Test Often' principle.

33. For faster quality application releases, we are now using an agile development approach. As a result, it is critical for QA teams to ensure that development efforts are not squandered. The "Test Early, Test Often" principle will assist QA teams in finding bugs early and frequently in order to confirm continuous delivery.

QA and development efforts should be combined with DevOps efforts.

34. To ensure continuous improvement in the end application, it is strongly advised to combine operations, development, and testing processes with DevOps. More information on Performance Automation Testing can be found here.

Participate in Continuous Testing

35. It is the process of performing continuous testing on each (even minor) build to determine whether or not it is successful. If not, we can quickly identify them and make the necessary changes before reintroducing them into the testing process. A process that all QA teams around the world must follow in order to ensure continuous delivery of the end product. Continuous testing is a process that can be automated.

Make use of ready-made test frameworks.

36. It is preferable to use ready-made testing frameworks. It assists QA teams in properly managing end-to-end testing cycles as well as ensuring continuous integrations and deliveries.

The Advantages of Software Integration

Easier decision-making

43. Having a single, comprehensive view by integrating your software systems simplifies decision making. It eliminates the need to switch between applications to access data that may influence your decisions.

Enhanced Productivity

44. Integrating applications that use the same data sources will allow you to increase the productivity of your operations. This is especially important when the same data is entered into multiple software systems. Processing is simpler and faster with a single point of data entry and no need to switch between different software applications.

More dependable data

45. Integrating and unifying your software systems reduces the possibility of using incorrect data. A single point of view will allow your company to operate from a single point of view and eliminate conflicting data values.

Improved analysis

46. When related data is combined in a single application, it becomes more meaningful and powerful. Analysis of multiple data sources is handled better by bringing the data together so that trends and conclusions can be drawn much more quickly.

Enhanced data security

47. Managing the security of your data within a single unified system application is far simpler than managing multiple data systems. Tasks are simplified by integrating management, backup, and administration.

Improved customer service

48. The ability to access customer information quickly and easily is critical for maintaining good relationships; integrating your CTI and CRM software will allow you to assist customers more effectively.

Enhanced sales potential

49. Integrating systems that streamline any aspect of your end-to-end sales process and increase order fulfillment rate will increase your overall sales potential.

CHAPTER FOUR

BAD CONSEQUENCES OF POOR-QUALITY SOFTWARE

50. Today, businesses are defined by their ability to provide quality software to their clients and customers. Building and maintaining quality software isn’t necessarily easy, but poor-quality software can stifle growth and ultimately upend your business.

51. The impact of poor software quality is difficult to define it in strict terms. Software quality is judged in the context of technical and budgetary requirements and many other external factors. And judgment by consumers is the prime measure for rating the quality. Apart from that, features like reliability, stability, and need for maintenance also impact software quality.

Poor business "fit" (mis-functional applications)

52. The biggest complaints about operational business applications are that they just don't do what business users wanted. Consequently, employees implement endless workarounds, managers use hidden spreadsheets, and the business fails to benefit from its application investment

53. The biggest cause of mis-functional applications is missed or inaccurate user requirements. It is easy blame IT for doing a bad job of requirements analysis, but the root cause often lies in immature business processes that vary widely across the business. In many organizations, these processes are often so poorly defined that requirements analysis resembles an archaeological dig.

Outages

54. The most damaging outages are usually those in customer-facing systems such as airline reservation, customer service, or online shopping. The costs of downtime, which frequently hit six digits per hour, can also involve lost revenue. Other related costs may include reactivating the system, recovering transaction fragments, spikes in help desk utilization, and even liquidated damages.

55. The root causes of outages are usually non-functional application problems that are generally invisible to end users until they cause a problem. Typically, developers did not engineer the application defensively to handle the myriad operational challenges

that can beset a system, such as excessive customer load or glitches in other applications with which it interacts.

Security breaches

56. The cost of security breaches can be staggering, especially considering expenses associated with closing the vulnerability, repairing any malicious damage, alerting customers whose records may have been penetrated, and then rectifying any damage caused to them. For instance, I was recently among tens of thousands who received replacement credit cards because hackers penetrated a vendor's transaction records. Security breaches most often result when developers inadvertently allow pathways into the application that skirt authentication procedures or expose the internal structure of the application through user messages. Attackers can use vulnerabilities such as these to inject malicious functions into the application during user interactions.

Business dis-agility

57. As organizations automate more processes, business agility is directly affected by the speed with which applications can be modified or enhanced to meet rapidly changing requirements. The longer it takes to modify or enhance an application, the less agile and competitive the business.

58. When an application becomes needlessly complex and its architecture decays through poorly engineered modifications, the time to release new functions and the number of new defects injected into the application grow proportionately.

59. With each decline in application quality, the business must wait longer to implement adjustments that enhance the company's competitive market position.

Poor performance

60. Although we rarely calculate the cost of lost productivity caused by degraded application performance, costs to the business are alarming when considering impact across a large department such as sales, claims processing, or customer service. Even a five percent reduction in application performance can result in hundreds of thousands of dollars in lost productivity each quarter.

61. The root cause usually involves programs that may be functionally correct, but written with poor coding practices that cause excessive processing as usage or data

volume increases. Performance problems are difficult to detect during development unless testers have the resources needed to simulate high loads the application may experience after deployment.

Data corruption

62. Data corruption is often not detected until users see a bill or report containing wildly inaccurate information. The cost of reconstructing the database and re-releasing invoices, documents, or other corrected materials can be extensive.

63. Frequently, data corruption results when developers do not use approved methods for accessing the database, leading to application data changes executed in an uncontrolled, or poorly coordinated, manner.

Method of Fixing problems

64. Although many of these problems can slip through testing undetected, there are application quality practices that can detect them:

65. Peer reviews of the application's design and code are effective in detecting defects that might otherwise slip past test cases written to verify compliance with functional requirements.

66. Static code analysis is also a good method to detect poor design or coding practices that may cause the impacts described here.

67. Dynamic analysis is another technique useful for uncovering certain classes of problems such as performance degradation.

68. IT executives should match their investment in quality practices such as peer reviews, testing, and static code analysis to the magnitude of the business risks these investments are expected to mitigate.

CHAPTER FIVE
MEASURES TO BE TAKEN AS AN ORGANIZATION OR INDIVIDUAL TO ENHANCE THE SOFTWARE PRODUCTIVITY

Factors Contribute to Increased Software Productivity

69. Following our definition of software productivity as the ratio of the functional value of software produced to the labor and expense of producing it, the next step is to identify ways to improve software productivity. Whereas computer hardware has a reputation for unprecedented performance and cost improvements in the history of technology, software productivity has lagged. This is due in part to the shortcomings in software productivity measurements mentioned above, as well as the fact that faster, more powerful computers can provide performance gains without gaining software productivity.

70. Nonetheless, software development firms are always looking for ways to improve both developer productivity and code quality. The first step toward improving either is to establish the above-mentioned productivity and quality metrics and benchmarks. Following the establishment of benchmarks, areas for improvement can be identified and action plans put in place to improve performance. This could be as simple as reorganizing developer workstations. According to studies, the design of a developer's workspace can have a significant impact on their productivity.

71. Leveraging, or code reuse, is possibly the most effective method for increasing software productivity. The process of reusing or porting application software across multiple business sites is known as leveraging. Sometimes the highest real productivity is the result of figuring out how to reuse previously written programs – possibly for a completely different purpose – or figuring out how to solve problems with previously existing programs by revising them as subprograms.

72. Reducing rework is another great way to boost software productivity. This entails detecting errors and problems as early as possible in the software life cycle. A mistake discovered in one phase can cut the amount of work required in subsequent phases by a factor of three. That is, a good requirements analysis can reduce the design job by three times, a good design can reduce the implementation job by three times, and

a good implementation can reduce the maintenance job by three times. Software reliability can be improved by employing a variety of analysis methods, such as software verification and testing.

73. Of course, the skill and personal behavior of the software developers themselves play a significant role in software productivity. It has been observed that there is a 10 to 1 difference in productivity among programmers, which is due to their varying levels of problem-solving skills and programming knowledge. Individual developer productivity can be increased by providing adequate training in, and insisting on, disciplined processes such as structured analysis, top-down design, modular design, design reviews, code inspections, and quality assurance programs to software developers.

CHAPTER FIVE

CONCLUSION

74. The chapter emphasizes the conclusion about the background that has been created with the software usage as organizations and individuals. This sector It is elaborated the bad and good impacts of a software based on their quality. According to the above discussions, we summarize the areas which we should point out during the research and we have realistic examples which were happened In the world history related to our presentation to prove how our theoretical facts.

REFERENCES

1. IBM. "What is Software Development". IBM. IBM. Retrieved 4 October 2021.

2. Johnson, Dave. "What is Software". Business Insider. Business Insider. Retrieved 4 October 2021

3. "Embedded Software—Technologies and Trends". IEEE Computer Society. May–June 2009. Archived from the original on 28 October 2013. Retrieved 6 November 2013.

4. Guides.ll.georgetown.edu. 2021. Guides: International and Foreign Cyberspace Law Research Guide: Treaties & International Agreements. [online] Available at: <https://guides.ll.georgetown.edu/c.php?g=363530&p=4821478> [Accessed 15 October 2021].

5. Asq.org. 2021. What is Software Quality? | ASQ. [online] Available at: <https://asq.org/quality-resources/software-quality> [Accessed 24 October 2021].

6. dzone.com. 2021. Top 10 Automated Software Testing Tools - DZone DevOps. [online] Available at: <https://dzone.com/articles/top-10-automated-software-testing-tools> [Accessed 24 October 2021].

7. Information and Software Technology, 1990. Software quality assurance. 32(1), p.2.