JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

THE SCIENTIFIC SIGNIFICANCE OF AN INCOMPLETE SOFTWARE DEVELOPMENT SEEN AS A DUMMY PRODUCT DURING THE PROCESS OF SOFTWARE DEVELOPMENT LIFE CYCLE

[1]Ogungbade, Olusoji Kehinde
1 LCU/PG/003794

[1]Computer Science
[1]Lead City University, [1]Ibadan, Nigeria.
Keywords: Asset, System, Software, Dummy, Product, Test, Stages.

Abstract: The literature review examines the significance and relevance of incomplete software seen as a dummy product within the Software Development Life Cycle. A dummy product can be integrated within the phases or stages of the software development to enhance creativity, productivity and decision-making while the product is under development. The review examines the challenges, benefits, and implications of adopting this kind of approach.

An incomplete software is a valuable asset that when it is well utilized and managed, becomes a part of the system. The scientific benefits of an incomplete software seen as a dummy product are discussed further. To detect a dummy product, tests or series of tests have to be carried out, evaluating its source, management and significance.

Keywords: Asset, System, Software, Dummy, Product, Test, Stages.

1.1 Introduction

The Software Development Life Cycle is usually followed by software units or organizations because of its structured approach to solving ambiguous software requirements and specifications to develop high-quality software. The traditional purpose of using the SDLC is to deliver an error-free and complete product to clients. Some relevant publications and their key finding are relayed below.

1.2 Aim and Objective

This study aims to explore the scientific significance of incomplete software development seen as a dummy product during the process of the Software Development Life Cycle. The objectives of the study are as follows: i) Assessing the impact of dummy products on the efficiency of the software development process. ii) Analyzing

the potential benefits and drawbacks of incorporating dummy products into the Software Development Stages. iii) Evaluating the role of incomplete software development in the scientific growth of software engineering.
To make software testing novices dummy testers who can only detect minor bugs and fix minor issues resulting to making them rise to at most an amateur tester.

To make an incomplete software product available for identification of possibilities and impossibilities in software development; This exposes a product as a potential working tool or a total throw-away product. This gives room to explore and identify what to expect and what not to expect in the future of a software product.

1.3 Scope

This study is targeted at understanding how incomplete software development, after being implemented as a dummy product within the software development cycle will contribute to and be integrated into the scientific progress in the field of software engineering. This study will consider different stages in software development phases, including design, implementation, requirements gathering, maintenance and testing,

2.1 Literature Review

The literature review will involve an extensive examination of journals written within the software engineering field, books, conference papers, and other relevant sources about software engineering, software development life cycle, and the ideation of incomplete software development, it will examine existing research on the benefits and limitations of using dummy products during the software development stages.

2.2 Methodology

Data Analysis: Collected data will be analyzed using qualitative or quantitative methods, qualitative analysis will involve thematic coding and pattern identification, while quantitative analysis will involve statistical techniques like regression analysis and hypothesis testing. Data will be collected through quantitative or qualitative methods. This will include interviews with software engineers, developers, development teams, and a synthesized case study.

2.3 Problem Statement

The stereotypical approach to software development often overlooks the scientific noumenon of incomplete software, by treating and managing incomplete software as a dummy product, there is a missed opportunities for valuable expositions and scientific growth in the field of software engineering. This study aims to address this problem and investigate the scientific significance of incomplete software development within the software development stages.

2.4 A Model Approach

This study proposes to conceptualize a model or utilize a model that integrates incomplete software, as a dummy product within the software development stages. The software model will pinpoint the stages of the Software Development Life Cycle where incomplete software can be discovered and utilized without considering being divested, the corresponding benefits and setbacks or disadvantages, and the potential relevance on software engineering, research and development.

3.1 Literature Review

3.2 Feedback and Validation

Research by Smith and Johnson (2018) demonstrates that involving stakeholders and potential end-users in the early stages of software development can provide valuable feedback. An incomplete software development acts as a prototype, allowing users to interact and provide feedback, leading to early validation of critical design decisions.

It is pertinent to know that incomplete software is not a throwaway product, it only lacks certain features which can be fixed through iterations and then validation.

3.3 Reduced Development Costs

In their study, Nguyen et al. (2019) argue that treating incomplete software as a dummy product can help minimize development costs. By delivering an initial, functional version of the software, developers can gather feedback and identify potential problems early on, mitigating expensive rework during later stages.

Incomplete software is never a nightmare for developers, it is rather a step further, celebrating milestones achieved. An incomplete software could be a nightmare for users or stakeholders because of the expectations they crave. Expectations that are not met. At least a dummy eliminates the cost of building from scratch. An incomplete software is an effort in progress. If it is well managed, it can produce a better working version of the system.

3.4 Rapid Iteration and Agile Development

By treating ongoing software development as a dummy product, the Agile development methodology becomes more effective. Research by Chen et al. (2020) suggests that Agile practices benefit from continuous feedback, allowing for rapid iteration, collaboration, and fine-tuning of features.

Every phase of Agile product development is not complete until the validation phase. Until the clients validate the system, agile is still open to operating each phase as incomplete and iterating the processes. Agile is more prone to utilize incomplete products as dummies because of its flexibility.

3.5 Challenges and Limitations

Managing User Expectations: Raising user expectations without delivering the complete software product can be challenging. Yao and Lee (2017) point out that users may find it difficult to differentiate between a dummy product and a fully functional one, potentially leading to dissatisfaction or confusion.

As long as testers utilize their tools, users can be put in the know of a working or non-working product thereby assuring users that defects will be fixed, and a complete system will be delivered. Operations personnel won't deliver a dummy to clients, this doesn't mean that dummies are not a valid means of creativity when they are reserved for later use or reference.

3.6 Security and Data Privacy Concerns

Treating incomplete software as a dummy product poses certain risks regarding security and data privacy. Special attention must be given to the handling of sensitive information. Research by Johnson and Brown (2019) emphasizes the need for appropriate security measures and strict access controls to ensure the protection of user data.

As long as dummies are not allowed or released to the end users, users' data are still being prioritized and kept classified within the development team, scrum master and the product backlog.

3.7 Integration Testing Condition

The purpose of a procedure call is to establish a record of activation. In other words, an activation is happening, and is being carried out. An activation to initiate a program, an activation to continue a program or an activation to end a program. Without modularity in program designs, procedure calls will be redundant instruction.

It has been reported that one in every 20 instructions is a procedure call or return. – ("MARY JEAN HARROLD, Clemson University MARY LOU SOFFA, University of Pittsburgh")

One of the purposes of creating modularity in software integration testing is that each procedure can be tested functionally or structurally. Functional and structural testing techniques make testing easy and direct because of their inherent ability to detect errors in each stage of the software development program.

Whenever modularity testing is being carried out integration of procedures is being considered. Once the module of an emerging system is considered to be error-free, then the integration of procedures is valid. This doesn't mean that the procedure interface is free of errors.

Studies indicate that integration errors constitute as much as 40 per cent of program errors. – ("MARY JEAN HARROLD, Clemson University MARY LOU SOFFA, University of Pittsburgh")

Integration testing can affect the procedures of system testing once an error is later detected in the preceding phases of system testing.

3.8 Structural Testing Techniques

The techniques used in the functional testing can be utilized or borrowed for use in the integration testing to cut the amount of time used formulating testing techniques especially when the program code is not required.

If we consider data flow testing, where values are being assigned to variables and their usage, there exist data dependencies. The values of the variable in use depend on values assigned during the definition.

After you determine data dependencies through a static analysis of a procedure, you generate test cases, either manually or automatically, that exercise the selected sub-paths. "MARY JEAN HARROLD, Clemson University MARY LOU SOFFA, University of Pittsburgh.")

Data flow testing can be extended integration testing, there must be a sub-path test defined by a situation in which a program statement (instruction) refers to the data of a preceding statement that crosses procedure boundaries through calls and returns.

Global variables and parameters are a way in which procedures interact directly over single calls and returns and also, indirectly over nested calls and returns. In nested calls and returns, codes are reusable. Global variables have a global scope and can be applied to all test cases. The only time component testing or unit testing should pass the testing phase is when the components have been integrated and are fully functioning with other components. When a component is fully functioning with other components, it is safe to agree that integration testing is carried out and passed. When a procedure calls another and it returns true, that means the component procedure is valid and usable with the integrated component. (Tools and Algorithms for the Construction and Analysis of Systems: Rajeev Alur, Kousha Etessami, P.Madhusudan. 2004).

After components have been integrated and tested but do not meet requirements or perform as programmed, then it can be called a dummy or a system sham. The system is seen as unintelligent yet. A system will be super intelligent when it can successfully test itself. But the age of such a system is gradually approaching when a system will be asked to self-test.

An emergent system results or comes into existence as an effect of integration. The individually integrated components can be termed as Lone Pair of their version. The purpose of building these types of systems is because they never existed before. Or rather, they existed but their functionalities are not complete making them an emergent, yet partially complete system. As long as an upgrade is paramount, no system is complete. The systems just met the required, moment solutions. The system can solve a problem that is already identified, making it solution-oriented.

There are hidden elements in each component. These elements will not appear active until they are integrated with another component. For example, you may integrate an authentication component with a component that updates information. You then have a system feature that restricts information updating to authorized users.

When a software fails a mutation testing phase, then the software is seen to be intelligent otherwise, the system is seen to exhibit a dummy condition. (Reference: Guru 99 2023). In mutation testing, the programmers must be very familiar with the source code otherwise there is no such thing as mutation testing phase. A mutant source code behaves as expected after it has been modified.

3.9 Working with Agile

“Working within Agile principles, engineers are not required to adhere to original requirements or follow a linear requirements workflow in which each team hands off work to the next. Instead, they are capable of adapting to the ever-changing needs of the business or the market, and sometimes even the changing technology and tool”.- DevOps for Dummy by Emily Freeman.

As long as developers work to produce a software product, there will always be an incomplete aspect of the project. Possibly due to insufficient time frame, budget, expertise, environment, tools, and team. As long as the project is under development it is incomplete and can be referred to as a dummy product. A product is not a dummy only when it meets all the stakeholder's requirements and functionalities. Dummies can be identified by pinpointing what works and what does not work.

“As a team that utilizes Agile for software development, you can’t take away these three factors from their operations (i) People (ii) Process (ii) Technology.

Process is the area in which you’ll see the most quantitative improvement in the speed of your organization’s software delivery”.- DevOps for Dummy by Emily Freeman

3.10 Individuals and interactions over processes and tools

The first value in the manifesto implies that emphasizing the abstract formal processes and their technical surrounding environment as key factors in software development is incorrect, the more important is the communication, interaction and quality of the human software developers that these factors serve. Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, P. (2018). "Advances in Using Agile and Lean Processes for Software Development". Advances in Computers (Vol. 113, pp. 135-224). Elsevier.

Without proper communication, a discovered incomplete software product will not be addressed to the right authorities. This can cause a delay in deliverables and other project resources capabilities can cripple. The quality of the human software developers is paramount to keeping or divesting a dummy product. Developers who are so up to the task may decide to start a project afresh if the quality of the software is not attained. However, this approach of starting all over by developers may be frustrating and time-consuming. This is why communication is vital, to encourage progress despite hurdles. After all, this is an Agile system, flexibility is the order.

3.11 Working software over comprehensive documentation

The documentation of any agile software development process is a vital and valuable component but the amount of time and resources that are given to it must be controlled and optimized not to overwhelm the software development process. The usability of the intensive documentation is one of the traditional development limitations - Larson, D., & Chang, V. (2016). "A review and future direction of agile, business intelligence, analytics and data science". International Journal of Information Management, 36(5), 700-710.

Documentation is considered to be a beneficial part of the processes of software development. It is right to understand that not all software process models can incorporate documentation into all phases of development because of their mode of operation. Like the Agile model which is very flexible because of its concurrent integration and feedback. If documentation is done in Agile, then each phase will experience documentation waste, and loss of records due to the nature of the model. Documenting a dummy in a particular phase or the entire system is vital because of its subsequent reusability.

3.12 Customer collaboration over contract negotiation

Agile Software Development was invented so that changes experienced in each phase of the development can be worked on iteratively without affecting the overall processes of the system development. In the Agile model, collaboration can never be over-emphasized because it is the duty of stakeholders like, users, developers and owners to always put all hands on deck for proper outputs. Signing a contractual agreement is relevant, but most relevant is collaboration because it determines the quality of inputs and outputs. An incomplete software can easily be made popular, even relevant through collaboration. Once there is a mutual agreement between stakeholders, dummies can turn to a support mechanism.

3.13 Responding to change over following a plan

One purpose of the process is to find a common ground between the user and the developers. The common ground doesn't mean they quarrel before the project, it means that they have the same understanding, and the same knowledge of a workable system, bringing both parties to the same speed, energy and interest. A plan does not mean that the proposed system is rigid or can't be reinvented. A plan in the Agile model is more of a guide or a blueprint that can be utilized based on available specifications. When the specification is out-used or under-used, improvisation becomes relevant as long as it meets customers' expectations. If the system is incomplete, seen as a dummy while the processes are yet within the plan, then the Agile model is capable of turning a lot of redundancies and in-completion around through its flexibility.

3.14 The Journey of Development

A central finding was that the needs of developers vary depending on the phase of their journey, ranging from the earliest interest to continuous use. These different phases can be called the developer journey, corresponding to the concept of a customer journey (Zomerdijk and Voss 2010).

The needs in each phase of Software Development vary because of the technicalities involved in it. Each phase can not stand on its own while the job is expected to appear finished. The needs of each phase must connect to the succeeding phase. Even though the Agile model is flexible, there are still some sequences of instruction and processes that have to be followed to succeed in each phase of the development.

3.15 Practices Adopted in the Analysis and Design Phase

Scrum's focus on collaborative teamwork explicitly acknowledges the importance of self-organization and is the most often employed agile methodology (Hron and Obwegeser 2018).

During the stand-up session, the Scrum master pep the minds of developers to focus on important tasks. This allows individual developers to direct their energy towards attaining a high standard of effort, even though an incomplete product may emerge.

3.16 Building Error Free Software

As a result, software development has been spared the kind of second-guessing of quality control to which most other work is subject. ProCD, Bryan H. Choi, Crashworthy Code, 94 WASH. L. REV. 39, 75–76 (2019) (describing Congress's willingness to enact legislative immunities for software developers, including Section 230 and the Y2K Act).

Building error-free software systems had proved to be far more difficult than anyone had anticipated, and software developers successfully persuaded lawmakers that broad legal protections were needed to save the industry against claims of defective software. Lawrence B. Levy & Suzanne Y. Bell, Software Product Liability: Understanding and Minimizing the Risks, 5 BERKELEY TECH. L.J. 1, 1–2 (1989) ("[S]oftware vendors are likely to face increasing exposure to lawsuits alleging that software did not perform as expected. The consequences of such lawsuits to software vendors could be catastrophic.")

As long as the prospect of software developers is to deliver a working product to clients and an agreement is made accordingly, the notion of exposing developers to lawsuits will be minimal because clients must know what to expect. A failed system is not a working system and may not be termed as an incomplete or dummy system because it is out of the development phases and processes. A dummy software can still be worked on despite its inefficiencies. A dummy can still be disintegrated in a failed system. But a failed system has never met up with a standard. If I build a system that monitors the human mind, a system that counts the number of thoughts that cross the human mind. If the system never counts the thoughts, it has failed. Does this mean that all the system can be thrown away? Does this mean that some parts of the modules can't still function if they were integrated with a functional counting module? No software product is perfect until it continually meets the needs of its users.

Writing Bad Code Despite Smartness

The gold standard of the conventional engineering process is the "waterfall" method: a project cascades from a top-level planning phase where the project's requirements are fully mapped out and defined. In subsequent phases of the project, those requirements are converted into a preliminary design, implemented as a working model, and then tested to ensure that the final product meets all the initial requirements. The Problem with Software: Why Smart Engineers Write Bad Code 196 Adam Barr, (2018).

The waterfall is the traditional model of software engineering where the planning takes the basic aspect of the stages starting from Design and implementation, where the system is developed to work as designed. Testing is where the system is put to the test to make sure that it functions as designed and implemented. Any system that does not pass the testing phase can be considered to be in a dummy state because of its incompleteness. Dummy has been in existence for a long time, even since the existence of the waterfall model used traditionally.

3.17 The Pros and Cons of an Incomplete. Software seen as a Dummy

Pros

1) Dummy can be detected from functional and non-functional aspects of the system.

2) A dummy can be a re-usable tool for development

3) A dummy can serve as a guide for an improved structural design

4) A dummy can be kept in the register for references

5) A dummy can serve as an incomplete prototype for future designs.

6) Dummy makes testers familiar with patterns (pattern recognition)

Cons

1) Dummy can be a waste of resources

2) A dummy can be misleading to an unfamiliar development team

3) A dummy is a retrospective that may cause delays and setbacks in development

4) Dummy can not be kept forever, it is either thrown away or updated.

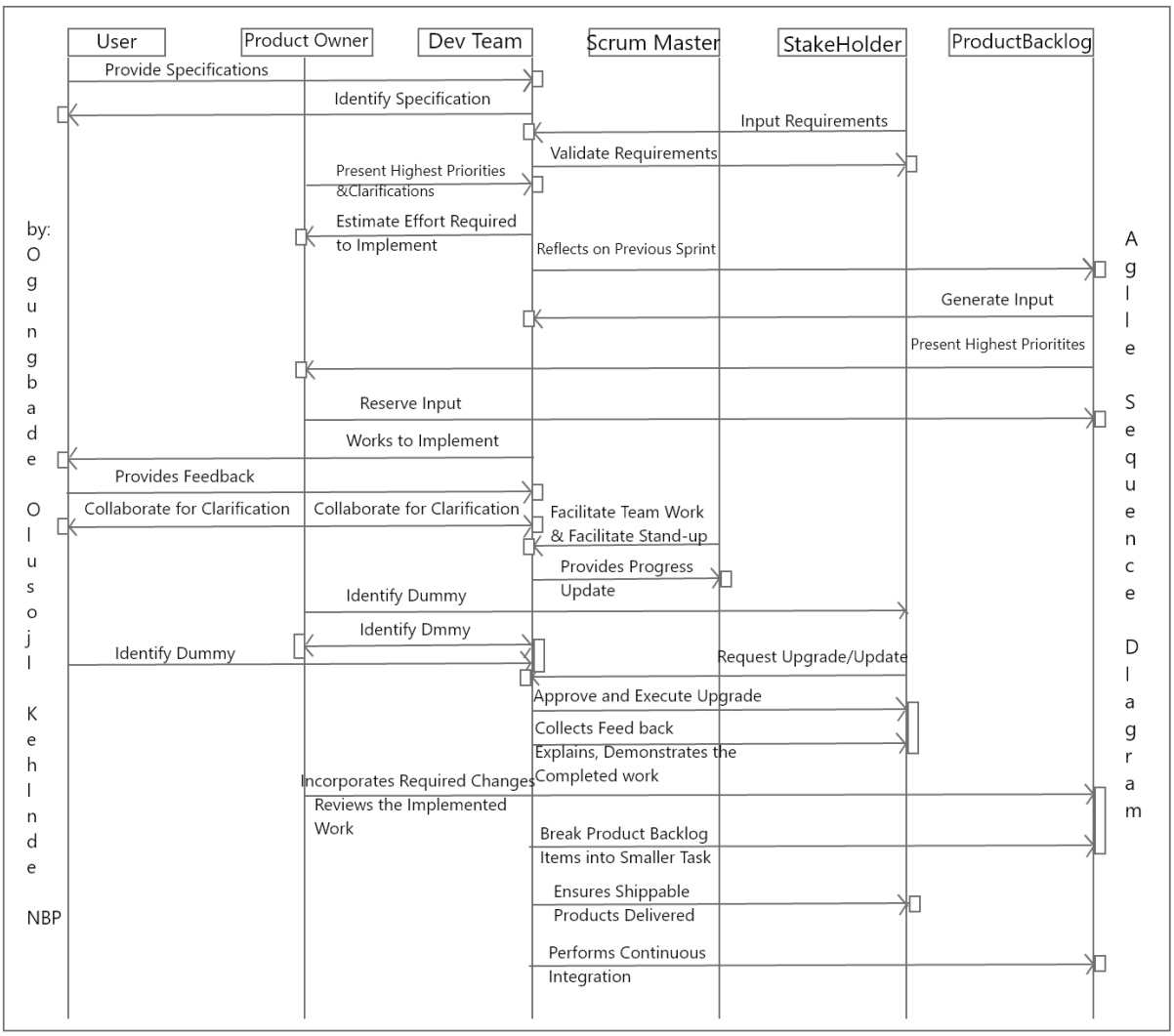
5) An updated dummy requires frequent monitoring just like every other system requires it.

4.1 METHODOLOGY (Qualitative Analysis)

4.2 Sequence Diagram Processing

1. Product Backlog(Actors)
   - Product Owner. Stakeholders
2. Sprint Planning(Actors)
   - Product Owner. Development Team:
3. Sprint(Actor)
   - Development Team. Scrum Master
4. Daily Stand-ups(Actors)
   - Development Team. Scrum Master
5. Sprint Review(Actors)
   - Development Team. Product Owner
6. Sprint Retrospective(Actors)
   - Development Team. Scrum Master
7. Sprint Backlog(Actor)
   - Development Team
8. Incremental Delivery(Actor)
   - Development Team
9. Continuous Integration and Testing(Actor)
   - Development Team

These are the actors and their associated activities involved in each of the iterative processes in Agile Software Development, Scrum method



agile sequence diagram: figure 4.2.1

The sequence diagram above is not there to restrict the work of developers. It is a depiction of an approach to solving software engineering problems in the Agile Scrum Method. Yet, it can be utilized for various process models that require a quick engineering intervention. It shows the interactions between entities, objects and their processes or activities.

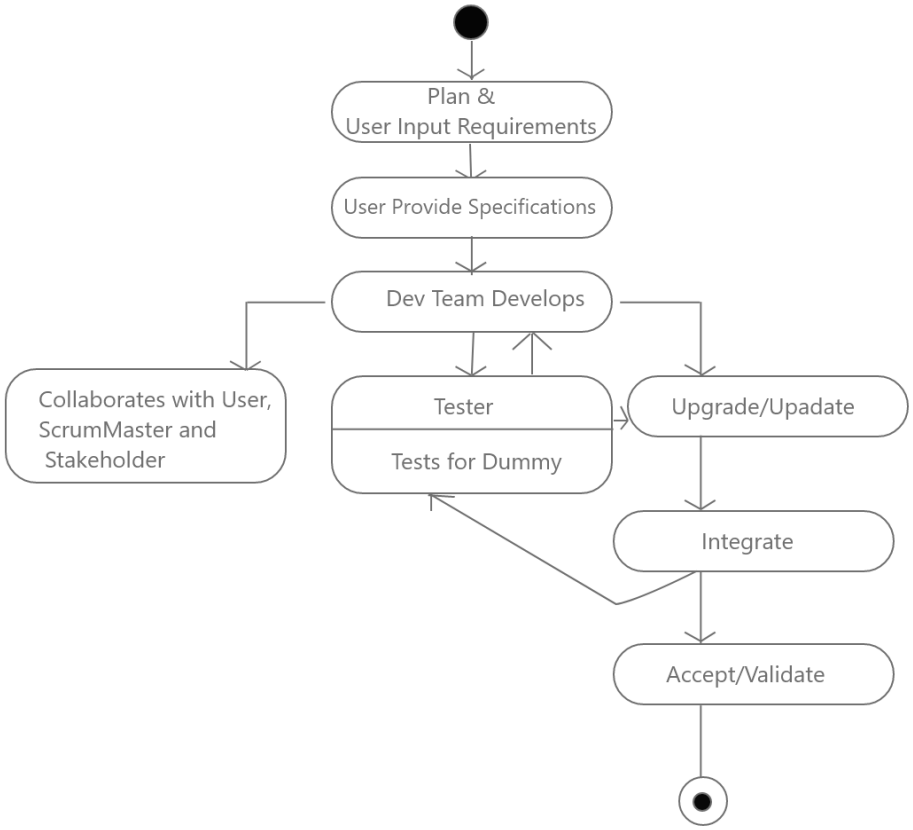
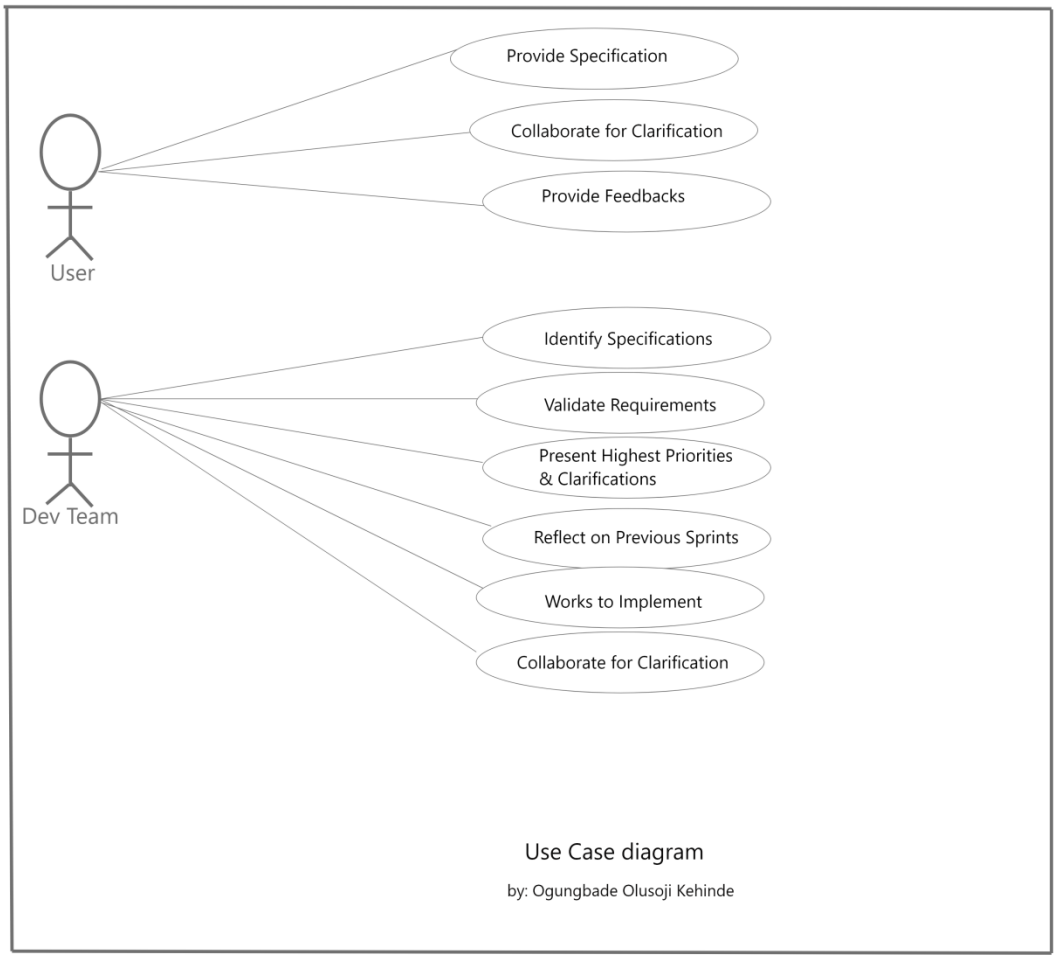


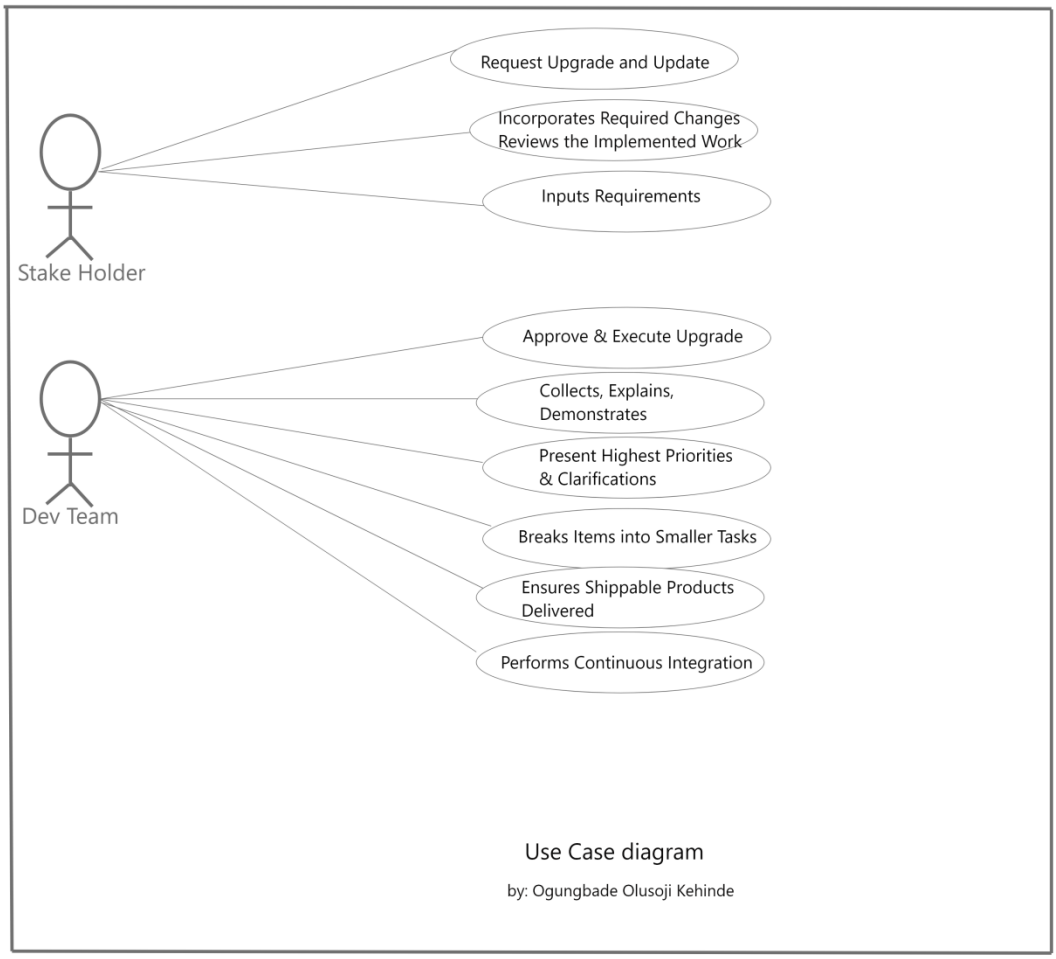
Figure 4.2.2 - State Diagram

The diagram above called the state diagram is there to create software model system visualization, especially the scrum method in the Agile model.

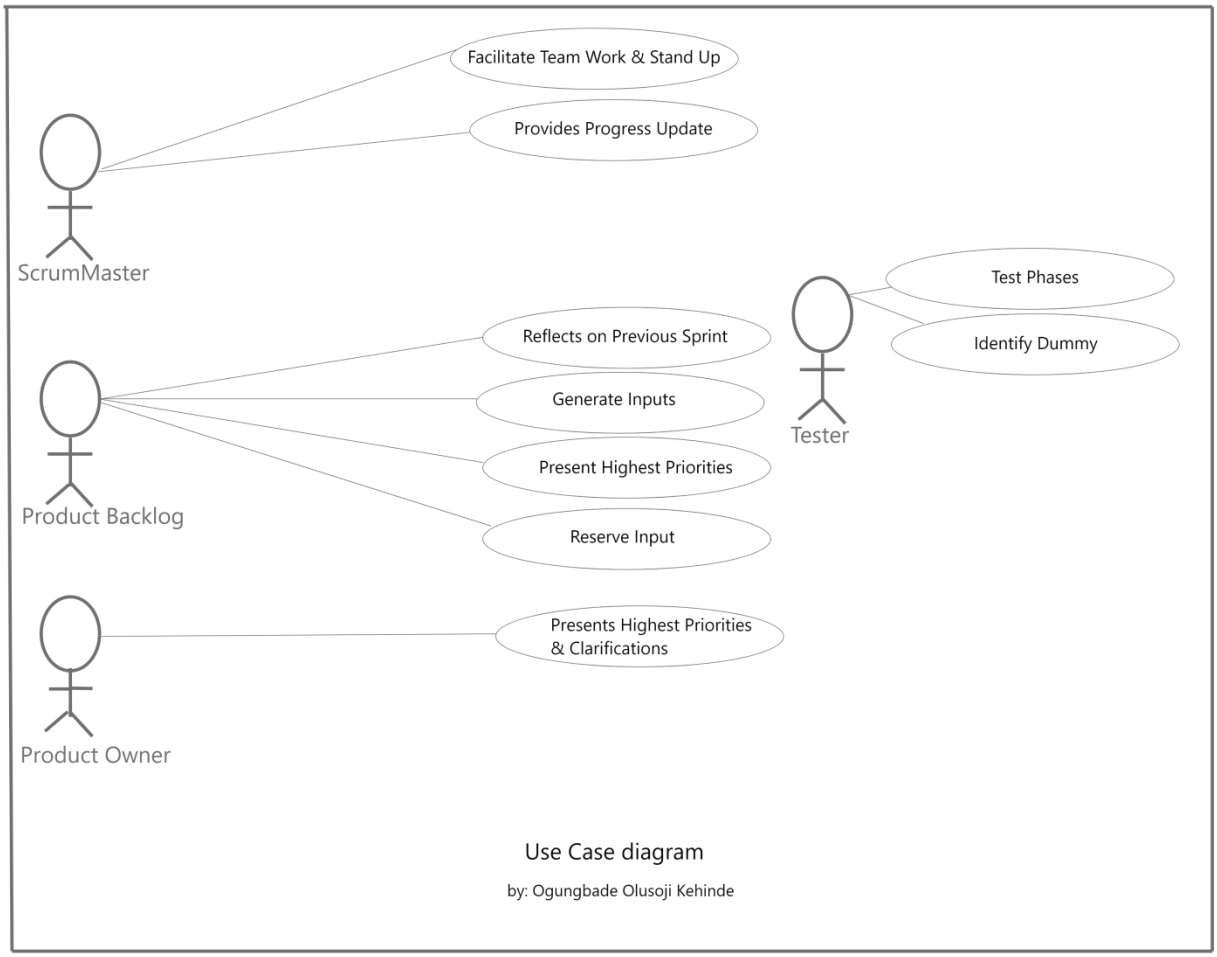


Use Case Diagram 4.2.3

The Use case above depicts how actors use the Scrum method in the Agile model.



Use Case Diagram 4.2.4



Use Case Diagram 4.2.5

4.3 Questionnaire Qualitative Analysis

Section 1

The questionnaire was developed using academic experiences coupled with individual experiences in the field of computer science, and software engineering to be precise.

So far active professionals in software engineering have contributed their inputs to the questionnaire. 100% of participants are male. With an age range of 26 to 45. They are all software engineers.

Section 2: Overview and Technical Know-How about Software Development

1. The different phases of the SDLC include Project Planning, Gathering Requirements and analysis, Design, Coding/Implementation, Testing, Deployment, and Maintenance. Each phase involves specific activities to ensure the quality of the software being developed. The SDLC ensures quality through activities such as requirement elicitation, documentation, code review, testing (unit, integration, system, acceptance), and ongoing maintenance and support.

2. The SDLC accommodates changes in requirements through various methods based on the adopted model. For example, the Iterative Model emphasizes repetition and allows for the creation of rapid versions of the software for testing and improvement. However, it is important to note that if not managed properly, the Iterative Model can quickly consume resources.

Section 3: Perception of Incomplete Software Development as a Dummy Product

4. The following responses were collected regarding the familiarity with the concept of incomplete software development seen as a dummy product in the SDLC:

- Very familiar: [Number of respondents 2]

- Somewhat familiar: [Number of respondents 1]

- Not familiar at all: [Number of respondents]

5. The responses to the question about whether incorporating incomplete software development as a dummy product within the SDLC has scientific significance are as follows:

- Strongly agree: [Number of respondents 2]

- Agree: [Number of respondents 1]

- Neutral: [Number of respondents]

- Disagree: [Number of respondents]

- Strongly disagree: [Number of respondents]

6. The following ratings were provided to assess the scientific impact of incomplete software development as a dummy product on various aspects of software engineering (each aspect rated on a scale of 1 to 5): where 1 indicates "no significance" and 5 indicates "high significance."

- Software quality improvement: [Average rating 3]

- User experience enhancement: [Average rating 5]

- Iterative and exploratory development approaches: [Average rating 3]

- Agile methodologies: [Average rating 3]

- Innovation and creativity: [Average rating 4]

- Early detection of design flaws: [Average rating 3]

where 1 indicates "no significance" and 5 indicates "high significance."

This indicates that the scientific impact of incomplete software development as a dummy product on various aspects of software engineering is more enormous and effective within the User Experience Enhancement category, having the highest significance. Followed by Innovation and Creativity. Whereas, the remaining factors are ranked the same.

Section 3: Benefits and Limitations of Incomplete Software Development as a Dummy Product

7. The respondents ranked the potential benefits of incorporating incomplete software development as a dummy product within the SDLC as follows: ranked from 1 (most significant) to 5 (least significant).

- Accelerated feedback loop for requirements validation: [Ranking 2]

- Early identification of potential challenges and risks: [Ranking 2]

- Enhanced collaboration between developers and stakeholders: [Ranking 2]

- Facilitation of rapid prototyping and experimentation: [Ranking 2]

- Improved documentation and knowledge sharing: [Ranking 3]

The first four factors were ranked most significant, while the last was ranked lesser with a point lower.

8. The limitations or challenges foreseen when incorporating incomplete software development as a dummy product within the SDLC were ranked: ranked from 1 (most significant) to 5 (least significant).

- Increased complexity of maintenance and future development: [Ranking 3]

- Difficulty in managing stakeholder expectations: [Ranking 2]

- Potential negative impact on team morale and motivation: [Ranking 3]

- Risk of delivering an unsatisfactory user experience: [Ranking 4]

- Limited external funding and resource allocation for such projects: [Ranking 4]

Section 4: Personal Experience and Opinions

9. The respondents indicated whether they have personally worked on a project that implemented incomplete software development as a dummy product during the SDLC:

- Yes: [Number of respondents 1]

- No: [Number of respondents 2]

10. Those who answered "Yes" to Question 9 provided descriptions of their experiences and observations regarding the scientific significance of incomplete software development.

Answer: The style of incomplete SD or product mapping or iterative prototyping has led me to in short time to bootstrap and work out all the different aspects of creating products with the benefit of seeing the dataset, ui-ux, server-serverless environment and overall the complex interrelations of every module in my product

11. Based on their experiences, the respondents shared their opinions on key factors determining the success or failure of incorporating incomplete software development as a dummy product within the SDLC. These factors include:

- Skill sets of the development team

- Team lead capacity

- Frequency and efficiency of stand-up meetings

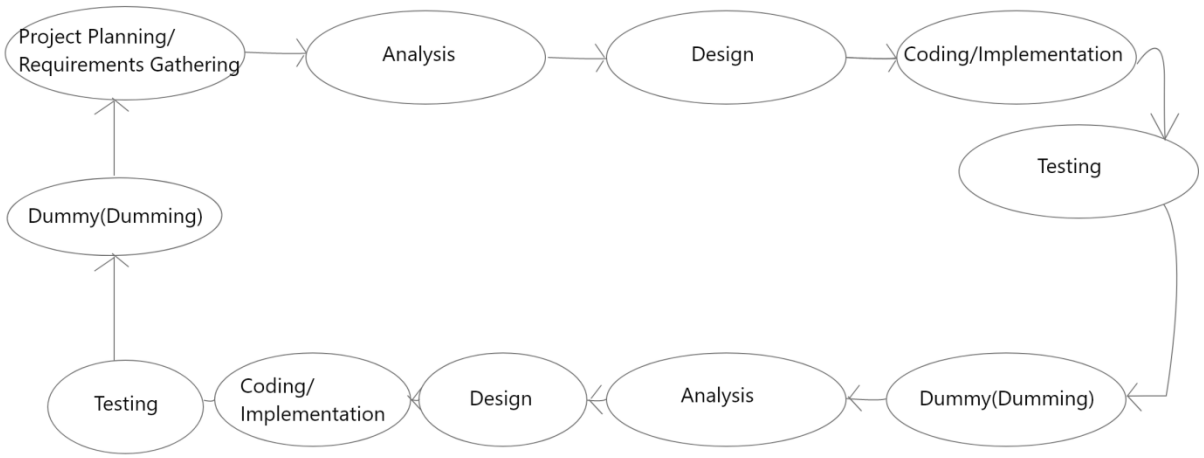
- Ease of reporting commits and receiving faster responses

- Management of bottlenecks

12. The respondents were asked to suggest research areas or specific aspects that should be explored further to understand the scientific significance of incomplete software development as a dummy product within the SDLC. The suggested areas include data modelling and artificial intelligence, which can facilitate faster prototyping and the curation of essential inferences for completing projects within set timelines

4.4 Conclusion

include Project Planning, Gathering Requirements and analysis, Design, Coding/Implementation, Testing, Deployment, and Maintenance. The introduced phases include; Project Planning, Gathering Requirements, Analysis, Design, Coding/Implementation, Testing, Dummy(Dumming), Analysis, Design, Coding/Implementation, Testing, Validation, Deployment and Maintenance. Dummy products or processes are a good source of experimentation. With an incomplete product, experiments can be carried out to know and understand the cause and possible outcome of working on the dummy.



Cycle of Phases

By: Ogungbade Olusoji Kehinde

Figure 4.4.1

Dummies are being detected during the testing phase. Because graphical and command line interface exposes little of the system's limitations. A dummy product may never be discovered in the early stage of software development because the early stages require fewer or no code inputs.

You may never intentionally make a dummy product because they are usually discovered during design, implementation, testing and maintenance

4.5 How a Dummy cannot be a Thrown away Product

I. Disintegrate. ii. Implement. iii. Integrate.

A discovered dummy can be disintegrated for proper attention. So it can be looked into for possible implementation.

Implementation can be carried out when the system is disintegrated to avoid affecting the other parts of the system. Disintegration makes the job easier.

Integration can be carried out after disintegration and implementation. The dummy parts have been recreated or upgraded. It can now work with the functioning parts of the system.

References:

Chen, L., Zhang, W., Jiang, X., & Li, X. (2020). Exploring the benefits of Agile development using an incremental approach. Journal of Systems and Software, 160, 110444.

Johnson, R., & Brown, K. (2019). Security concerns of using incomplete software developments as dummy products. Information Security Research Journal, 30(4), 543-560.

Nguyen, T., Tran, H., & Le, T. (2019). Cost reductions in software development through iterative prototypes. Proceedings of the 13th International Conference on Software Engineering and Applications, 209-215.

Smith, J., & Johnson, M. (2018). Using prototypes to enhance the early feedback stage in software development. Journal of Systems and Software, 145, 183-197.

Yao, Q., & Lee, Z. (2017). Managing user expectations regarding incomplete software developments. International Journal of Human-Computer Interaction, 33(2), 175-196.

Tools and Algorithms for the Construction and Analysis of Systems: Rajeev Alur, Kousha Etessami, P.Madhusudan. 2004).

“MARY JEAN HARROLD, Clemson University MARY LOU SOFFA, University of Pittsburgh”)

DevOps for Dummy by Emily Freeman

Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, P. (2018). "Advances in Using Agile and Lean Processes for Software Development". Advances in Computers (Vol. 113, pp. 135-224). Elsevier.

(Zomerdijk and Voss 2010) Zomerdijk and Voss 2010. https://jserd.springeropen.com/ Article number 6, 2018

(Hron and Obwegeser 2018). https://jserd.springeropen.com/ article number 5.

ProCD, Bryan H. Choi, Crashworthy Code, 94 WASH. L. REV. 39, 75–76 (2019) (describing Congress’s willingness to enact legislative immunities for software developers, including Section 230 and the Y2K Act).

The Problem with Software: Why Smart Engineers Write Bad Code 196 Adam Barr, (2018).