IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 6.252

Volume 2, Issue 1, August 2022

Performance Evolution of Software Engineering Development and Project Management: A Critical Evolution Software Engineering Process

Gurudev Sawarkar[1] and Dr. Dipesh Rajput[2]
PhD Scholar, Department of Computer Science and Engineering[1]
Associate Professor, Department of Computer Science and Engineering[2]
Swami Vivekananda University, Sagar, MP, India

Abstract: *Among the many pressing concerns in the realm of computers, software project development is among the highest. System development life cycle is a part of this (SDLC). A key goal of the software development life cycle (SDLC) is to reduce the likelihood of errors while improving the quality of the final result. Without a well-defined set of steps, the software development process is a pretty complicated affair. Method established to standardize and streamline software development. The introduction of the SDLC (Software Development Life Cycle) existence. What we have here is a methodical and organized approach to the process of creating software. Using the SDLC as a guide, entails a wide variety of steps and processes that must be completed before the final programmer is released. Software comes in many forms. Types of software development life cycles, each with their own benefits and drawbacks, are commonly employed in the software development process. Disadvantages Five of these software development life cycle (SDLC) models, including the waterfall model, the v-shaped model, and the prototype model, are presented in this study. Existing models are compared using a spiral and an iterative structure.*

Keywords: SDFC, Waterfall Model, Spiral Model.

I. INTRODUCTION

The software development life cycle (SDLC) has shown to be the most effective method for creating software. The Software Development Life Cycle (SDLC) is a framework that describes the activities to be carried out at each stage of the software development process. The Software Development Life Cycle (SDLC) is a method of monitoring and managing projects; it improves the clarity of project planning and the rate at which new features are introduced.

Stages of SDLC are:

1) Project Definition
2) Requirement
a) User requirement
b) System requirement
3) Analysis and Design
4) System Build
5) Testing and Implementing
6) Deployment
7) Sustainment

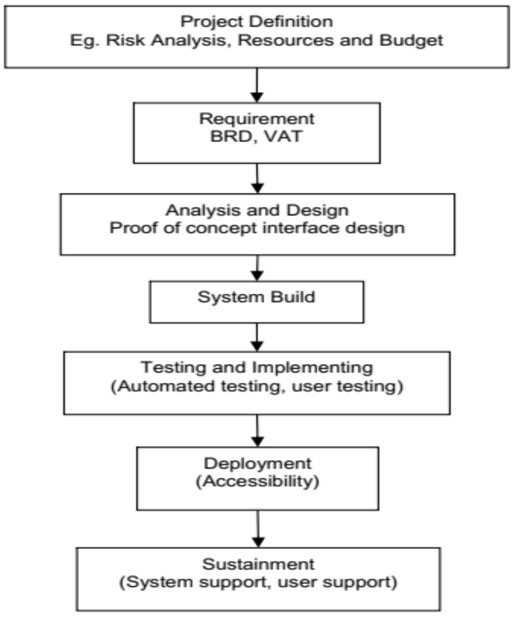

Project Definition
Eg. Risk Analysis, Resources and Budget
Requirement
BRD, VAT
Analysis and Design
Proof of concept interface design
System Build
Testing and Implementing
(Automated testing, user testing)
Deployment
(Accessibility)
Sustainment
(System support, user support)

FIGURE 1: Represent the Various Stages of SDLC.

II. SDLC MODELS

To manage the level of complexity, a number of SDLC methodologies or models have been created. These models are created to ensure success in software development process [3]. Below are the SDLC models followed in the software industry.

2.1 Waterfall Model

In terms of design methodology, this is a straight sequential approach with no iterative steps. Each stage of a waterfall model, such as successfully installed, requirements, analytical design, code and unit test, system integration, deployment, and upkeep, must be finished before the next stage can begin.
Waterfall approach supports the performance of software projects by removing the challenges that were previously experienced. With the waterfall paradigm, each stage's output feeds into the next.

A. Advantages

1. The requirements are well-defined; that is, they are straightforward and not difficult to grasp.
2. It's simple to control.
3. Detecting lapses in progress early on.
4. Four, the procedure and outcomes are meticulously recorded.

2.2 Spiral Model

Spiral model is a risk-driven process model generator for web applications. It manages to combine the idea of agile process with the governed and systematic parts of waterfall model.

A. Advantages

1. Software is made and managed in a strategic way, and it is easy and effective to keep track of projects.
2. Users can see how the system works early, which means that software is made early.
3. Changes can be made more quickly and even later in the life cycle.
4. Documentation control and strong approval.
5. The spiral model lets us make highly customized products.

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 6.252

Volume 2, Issue 1, August 2022

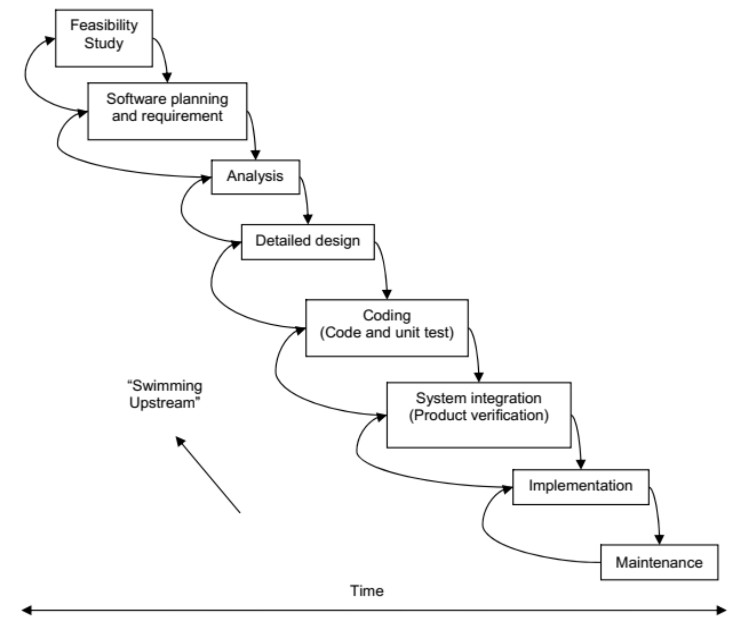

Feasibility Study
Software planning and requirement
Analysis
Detailed design
Coding (Code and unit test)
System integration (Product verification)
Implementation
Maintenance
"Swimming Upstream"
Time

FIGURE 2.1: Represent the Waterfall Model.

2.3 V-Model

The V-model is a method for making software in which the steps are carried out in a V-shaped order. The relationship between each development life cycle and the testing phase is shown by the shape of a V. Verification and validation model is another name for the V-model. In this prototype, the last step should be finished before the next step can begin.

A. Phases of V-model:

1. Verification Phase
   a. Business requirement and analysis
   b. System requirement analysis
   c. Architecture engineering
   d. Design
   e. Detailed specification

2. Coding Phase: - Coding is performed which are based on the coding guidelines and standards.

3. Validation Phase
   a. Unit testing
   b. Component testing
   c. System integration testing
   d. System testing
   e. Acceptance testing

B. Advantages

1. Very easy to use because each step has clear goals and objectives.
2. Development and progress are well-planned and step-by-step, and each phase is finished before moving on to the next.
3. The V-model works well for most small projects because the requirements are easy to understand.
4. Bugs are found earlier in the process of making a product.
5. Early phase as during software development life cycle gives it a greater chance of success than the waterfall model.

C. Disadvantages

1. It's not good for complex, object-oriented projects because it's difficult to change features later.
2. Less likely to meet customer needs because no prototypes are made.
3. There is uncertainty and risk because there is no way to do a risk analysis.
4. It locks in clarity and coherence.
5. If changes are made to a project while software is being made, all project documentation should be revised.

III. LITERATURE REVIEW

3.1 Defining Games and Gamification

Game refers to "a system in which participants participate in a manufactured conflict, specified by rules and leading to a measurable end," but there are still conflicts of view. It's on page 80 of the book [153]. In a typical game there are players, rules of interaction, a procedure or game scheme with which a actor interrelates, and a goalmouth often founded on a manufactured conflict or a result.

The concept of gaming is pervasive and strongly linked. There are several advantages to categorizing human activity. For complicated software artefacts that require a large amount of collaboration, games are a tremendous help in forming social structures. As a result, it should come as no surprise that social games offer significant social and economic advantages. They may even be able to provide remedies to societal issues in some circumstances.

To put it simply, gasification is the use of game design features and game-based thinking to inspire behavior similar to that of a gamer (such as competitiveness, cooperation, etc.). Game mechanics and design standards are being used outside of the video game industry for the first time. Given the magnitude of the sector, games have shown to be effective motivators for a predetermined period of time. There are several ways in which the application of gasification may change a tedious task into a favorite pastime. To add to this, some companies have already included gamification into their customer loyalty programmes, mostly to gain an edge over competitors by enticing consumers to utilize their products and services (for example, by awarding reputation points or accomplishment badges.

The phrase gamification, on the other hand, is still a bit of a grey area. Gamification might signify various things to different people. GAMIFICATION is defined as "the employment of game design aspects in non-game situations" by Deterring and others. Game-like elements may be used to increase the quality of services, which could be beneficial for marketing services.

There are many who believe that unlike serious games, which propose an entire game design, gamified apps should merely include parts of a game in order to be considered gamified [162]. A possible drawback to this explanation is that the definitions of game elements are not agreed upon. Even if a game is made up of self-similar pieces, the process of gamification might result in the creation of an entirely new product, such as a game itself. It's a third point made by this investigator that early images don't take procedure rational into consideration.

Gamified systems and components can only be created when players' interaction patterns and game mechanisms (i.e., rules of interaction) are defined. Gamers' desired degree of involvement should be taken into consideration while designing the game's mechanisms (such as levels, badges, etc.). Gamification is a relatively recent concept in the world of software development. Game-based frameworks can aid in the exploration of management issues, such as the best team configurations, in software engineering contexts. Non-gaming contexts can benefit from game-based ideas in avariety of ways. First and foremost, it inspires individuals more effectively than any other known strategy. An inherent motivation with an external reward can be transformed' Second, we argue that a game may be constructed from a collection of game components, and the results of that game can be examined using game theory.

3.2 Games in Software Engineering

In an information-based global economy, computer project's social and economic importance has grown. As a result, just as in subjects like sociology, economics, and computer science, certain methods to software engineering study make use of game theory.

Software development has been conceptualized as a type of positive and cooperative gaming in a few small studies to date. Cooperative game theory was used to build a model for optimizing work allocation in software engineering efforts. A game theoretic method, on the other hand, was proposed by Grechanik and Perry [6] because of the

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, August 2022
Impact Factor: 6.252

possibility for conflict between roles in a software development approach. Software development was also viewed as a series of games of creation and interaction by, who called the process of creating software "economic-cooperative gaming. "" It is as though two goals are contending for a resource in an iterative game. The mechanics and economics of communication is a new field he proposed as an emergent one "It's something that has to be looked into soon.

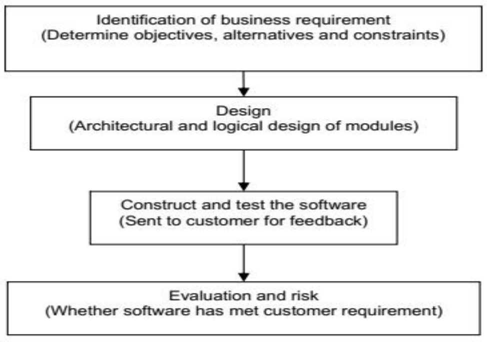

Identification of business requirement
(Determine objectives, alternatives and constraints)
Design
(Architectural and logical design of modules)
Construct and test the software
(Sent to customer for feedback)
Evaluation and risk
(Whether software has met customer requirement)

FIGURE 2.2(A): Represent the Steps Involved in Spiral Model.

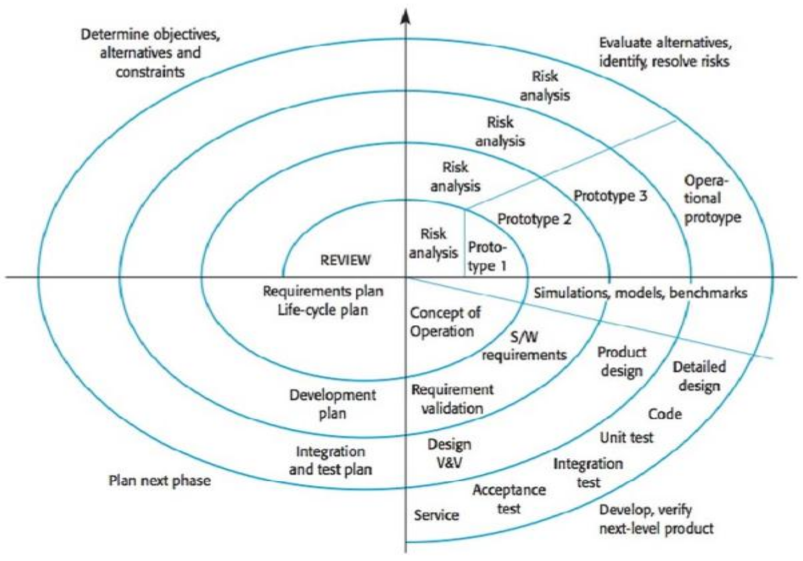


FIGURE 2.2(B): Represent the Spiral Model of a Software Process.

IV. EXPECTED RESULT

4.1 Research Process: Case Study I

In this part, we go over the steps we used to gather data for Case Study I. A case study examination of a medium-sized software firm was utilized to investigate empirically the factors impacting software development productivity. In terms of technique, the following phases are included in our study.

1. Considered as latent variables that can't be directly seen are: productivity, social productivity, and social capital As a result, we employ a number of different variables to narrow the field of candidates.
2. In order to construct our hypothesis, we use the specified variables to show that productivity, social productivity, and social capital all have an observable relationship.
3. After identifying the components that affect productivity, social productivity and social capital in a software development organization above, we do a literature analysis in order to identify the most important aspects that influence these outcomes.
4. To further investigate the relationships between the many latent variables that have been discovered, we group them into three different categories.
5. Fourth, we devised a survey instrument containing sixty items on a Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree) (strongly agree).

6. Before moving on to data collection and analysis, the questions are turned into a questionnaire. The majority of the data is utilized to examine the correlations between our three latent variables and the factors influencing them. As part of our research, we questioned participants about their job experience, gender, and preferred team size (see Appendix A for questions 18, 19, and 20).
7. For the fifth step, we use a case study to show how the framework works in practice by doing a confirmatory factor analysis and creating multiple structural equation models using single, double, and tripartite models.
8. With one latent variable models, we first use data from the literature to construct our first model and then refine it by doing a focus group research where we ask the firm for their thoughts on which factors are most important to the organization.
9. In order to compare a version from the literature with a version generated using indicators specified by the firm from an industrial viewpoint, we design alternative SEM models for each latent construct (see Model II, Model IV, Model VI).

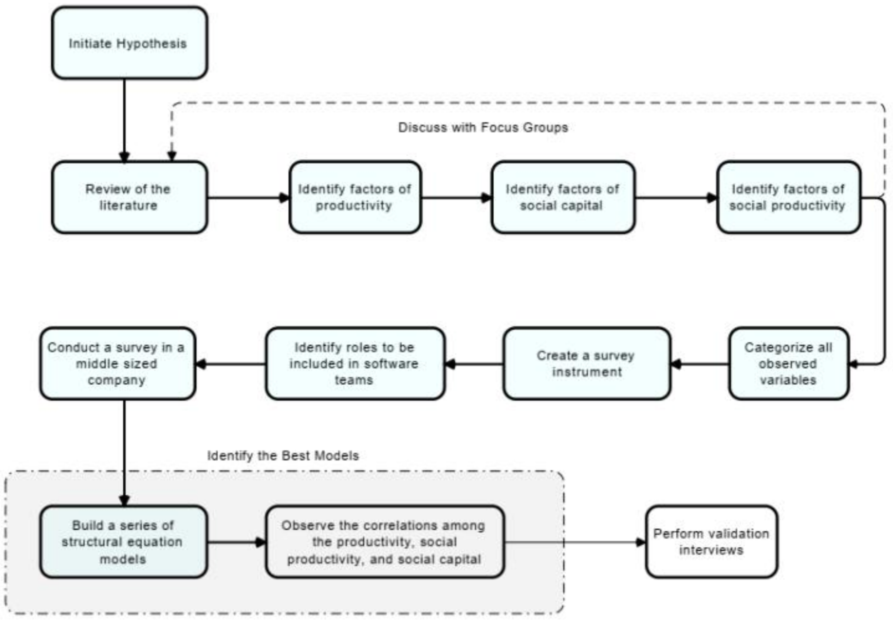

Initiate Hypothesis
Discuss with Focus Groups
Review of the literature
Identify factors of productivity
Identify factors of social capital
Identify factors of social productivity
Conduct a survey in a middle sized company
Identify roles to be included in software teams
Create a survey instrument
Categorize all observed variables
Identify the Best Models
Build a series of structural equation models
Observe the correlations among the productivity, social productivity, and social capital
Perform validation interviews

Figure 4.1: The Systematic Approach for Creating SEM Models of Productivity

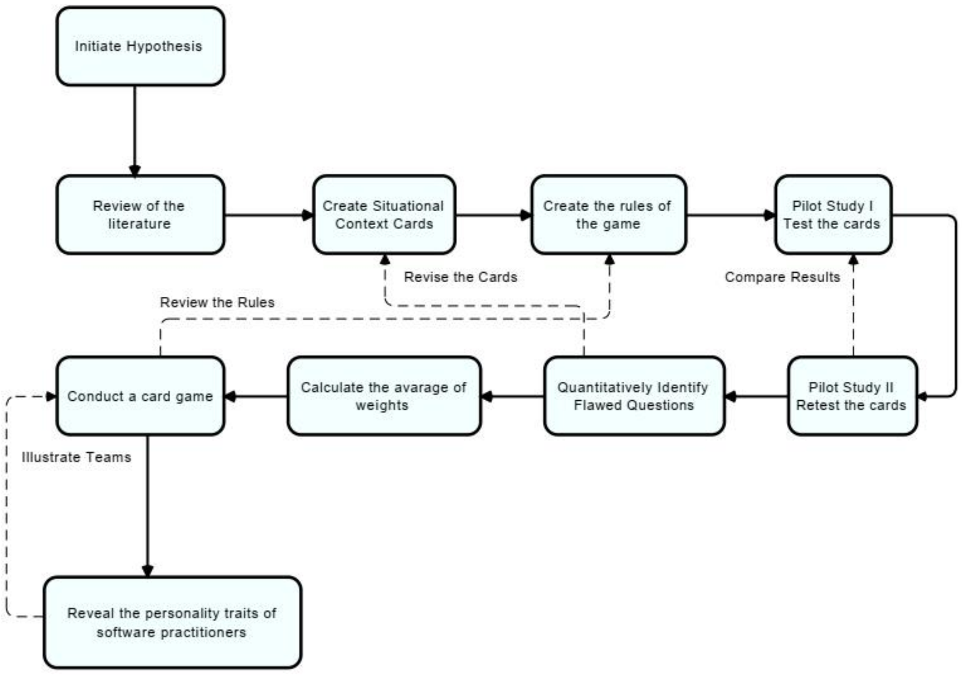
4.2 Research Process: Case Study II

For our second case study, we describe the research approach we used (Figure 3.2). The following are the specifics of the research procedure

1. In the beginning, we hypothesize that successful software teams have unique personality qualities that are linked to their ability to work well together.
2. Second, we look at the literature on the usage of MBTI testing in the field of software engineering, and we examine prior findings. As part of the grounded theory analysis, we'll look at coding that is based on context cards.
3. Finally, we develop the situational context cards (described in the next chapters).
4. In order to carry out a game-based examination of personality, we must first set the laws of the game.
5. First we perform a pilot research and gather data for the questionnaire in order to assess the situational context cards.
6. A second pilot research will be conducted with the same set of persons to recollect data for each topic.
7. Step 7 involves applying a quantitative analytic approach in order to identify issues that are troublesome in both pilot testing. We conduct a two-step case study in the industrial sector.
8. 216 individuals from the same software firm were surveyed in the eighth phase to examine the significance of the questions. Accordingly, the average weights for each element that influences personality characteristics are

calculated, which are used to assign weights to each question.

9. At this point, we rerun our game with a total of sixty software practitioners working on a variety of teams.
10. On the basis of this case study, we demonstrate five software team architectures in terms of their members' personality attributes.

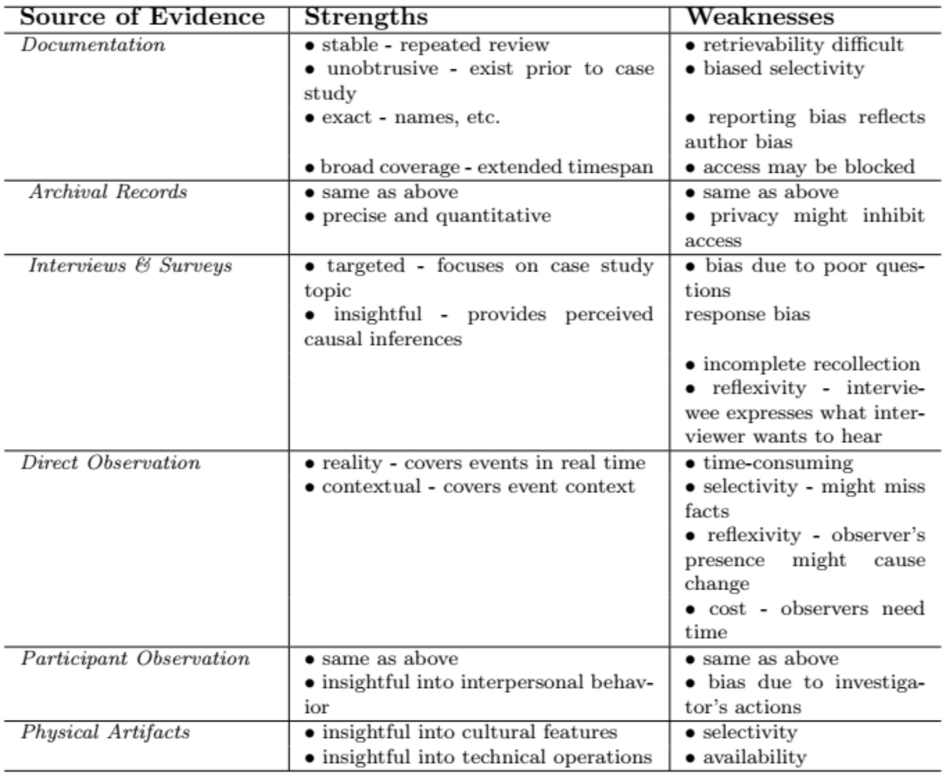

Initiate Hypothesis
Review of the literature
Create Situational Context Cards
Create the rules of the game
Pilot Study I Test the cards
Revise the Cards
Compare Results
Review the Rules
Conduct a card game
Calculate the avarage of weights
Quantitatively Identify Flawed Questions
Pilot Study II Retest the cards
Illustrate Teams
Reveal the personality traits of software practitioners

According to Yin, a case study ought to be an empirical investigation that investigates a state-of-the-art phenomenon in real-life scenarios on the basis of various data sources and fuzzy boundaries in the given context. Yin makes this recommendation in the context of a particular context. In his conversations about the significance of a case study as a research strategy, Verschuren produced the following argument: "A case study is a [triangulated] study design that can be qualified as holistic in nature, following an iterative-parallel way of preceding, looking at only a few strategically selected cases, observed in their natural context in an ajar way, clearly attempting to avoid (all variants of) tunnel vision, making use of analytical comparison of cases or sub-cases, , there are six distinct kinds of data (evidence) that are appropriate for use in case studies: I paperwork; (ii) archival records; (iii)interviews (or surveys); (iv) direct observation; (v) participant observation; and (vi) physical artefacts. Both independently and in conjunction with one another, they can provide useful information.

In general, there are four types of case research design available: (1) single-case ingrained, (2)single-case holistic, (3) multiple-case engrained, and (4) multiple-case holistic. Each of these particular instance study designs is based on a holistic analysis of individual case or numerous instances, and each of these case study designs can be based on individual unit of inquiry or numerous units of study. It is common for a researcher to use a case study as a way to relate the data he or she has gathered to the original research question.

There are several advantages to using case studies.

Software engineering activities may be merged with them, and if genuine projects are employed, there is no need to raise the size because they are already on the actual industrial scale, and (iii) they allow the researcher to assess the actualized and predicted advantages of the progress made. A case study has four primary steps: I planning, (ii) carrying out, (iii) analysing, and (iv) drawing inferences. If you're looking to evaluate software engineering processes and tools industrial case studies are a must. They help to reduce biases and assure validity (e.g., internal and external) from an evolutionary perspective.



Source of Evidence	Strengths	Weaknesses
Documentation	• stable - repeated review • unobtrusive - exist prior to case study • exact - names, etc. • broad coverage - extended timespan	• retrievability difficult • biased selectivity • reporting bias reflects author bias • access may be blocked
Archival Records	• same as above • precise and quantitative	• same as above • privacy might inhibit access
Interviews & Surveys	• targeted - focuses on case study topic • insightful - provides perceived causal inferences	• bias due to poor questions response bias • incomplete recollection • reflexivity - interviewee expresses what interviewer wants to hear
Direct Observation	• reality - covers events in real time • contextual - covers event context	• time-consuming • selectivity - might miss facts • reflexivity - observer's presence might cause change • cost - observers need time
Participant Observation	• same as above • insightful into interpersonal behavior	• same as above • bias due to investigator's actions
Physical Artifacts	• insightful into cultural features • insightful into technical operations	• selectivity • availability

VI. CONCLUSION

6.1 Discussion of the Case Study I

Using survey data provided in Appendix C, we conduct an empirical evaluation of the hypothesized correlations between the latent variables and many factors influencing them in the first case study. The findings reveal a modest relationship between social capital and productivity, and a strong relationship between productivity and social productivity.

In spite of this, a moderate relationship between social production and social capital is visible. These results lend significant credence to the argument that social variables have a significant impact on software output. Regarding the societal and organizational concerns raised at the outset of this investigation, it is now able to assert that the majority of the criteria chosen from the literature are influencing the efficiency with which a software development company operates.

The aforementioned findings, taken as a whole, provide important new insight into the economic and social determinants of productivity that can be measured with the structural equation model. Because it creates a multifactor productivity metric, this study is also the most thorough (empirical) research to date, which has important implications for both business and academics. Tripartite structural equation modelling (SEM) utilizes a multi-dimensional components structure, with seven variables, to assess three of our structures of interest. This is the first research of its kind to examine the effects of software development team size, social capital, and individual responsibilities.

6.2 Validation Interviews

One of the problems that arises from these models is the necessity to assess them through a set of model validation interviews [144] with members of the Simurg management team in order to determine how accurately we assess the output scale.

We used questions like "What do you think about the company-based outcomes we have identified using SEM models?" to get participants' feedback on the accuracy of our models "Do you think there is a component of the productivity model that is either missing or incorrect? Which ones, if any, have helped your company the most? Do you think these findings might aid the software development company in increasing productivity? "

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Volume 2, Issue 1, August 2022

Impact Factor: 6.252

The leadership team was excited to look at the results of this section of the work since they had previously reviewed a number of reduced versions of these models in a focus group research [330]. The Model IX, X, and XI were of most interest to them. Respondents were asked about their thoughts on the connections between the predictors and latent constructs.

The majority of respondents agreed that the findings were in line with their expectations, although a few offered small tweaks to the ranking of criteria. Our structural models allow for insightful conversation about social aspects, quantifiable latent characteristics, and most crucially, how to use these insights to boost organizational output.

Finally, this research suggests that software development companies may employ our method to gauge the effectiveness of internal variables. The upshot of this is that the upper management may build a scale and find the causal links between indicators to determine the underlying causal hierarchy of these factors.

6.2.2 Limitations

There are certain problems with the current research. To begin, the data included in this literature evaluation of productivity determinants comes solely from published sources. Therefore, the factors we included in our SEM models constitute a constraint.

Second, although we have nearly two hundred participants from an industrial company (Simurg), which can be considered as a substantial sample set in terms of software engineering to draw some evidence based findings, we gathered our data from a single software company, that should be tested with various settings for model comparison.

Third, there is the potential for unintentional sampling error. As a result, multi-latent-variable models were checked for a single-factor solution to assess the gravity of a typical technique mistake.

Fourth, although the sample size is large enough according to the SEM literature, we may decide to expand our research to a larger pool of businesses. Participants' anonymity was guaranteed to preserve their privacy. We were able to collect a sizable portion of the data, but there was no enforcement at the business level. Fifth, the self-report measure is used in this study. As a result, we were unable to determine whether or not the same outcomes would be seen using alternative data gathering techniques. Additionally, we used a cross-sectional study design, meaning that we collected data by conducting our survey at a static moment in time. Because the evidence does not provide substantial substantiation for causality, the direction of causation and causal ordering cannot be identified. That is to say, we can't make any definite conclusions about the causal correlations from our models because they're based on correlational data. However, the technique bias was mitigated by using a combination of case studies and questionnaires.

Few papers have been published in software engineering journals on the topic of structural modeling's efficacy in quantifying factors impacting software development productivity. For instance, there is a SEM model for increasing efficiency in software development ([331]) and another for gauging the likelihood of a project's success ([332]). Since there is a dearth of corroborating research, it is important to proceed with care, since the results may not generalize to other software development companies at this time.

6.3 Discussion of the Case Study II

The second case study's principal goal is to create a game-based tool to classify the personalities of software developers. The purpose of the tool used in personality testing is to quantify some facet of human performance. Therefore, experimental studies should be used to validate assessment of such an instrument. The card game was field-tested twice on sixteen individuals over the course of six months to assess question reliability (see Appendix E for both data sets). All sorts of business scenarios serve as backdrops for these cards. By applying grounded theory to a set of interview transcripts, we were able to collect context-specific keywords for use in the creation of our playing cards.

When all the pieces have been moved around, the player's final score reflects their dominant personality attribute on an MBTI-compatible scale. Second, we utilize these cards to illustrate the distinct personality types of 63 software development industry professionals working in a wide range of teams and departments (see Appendix H). Based on the findings in Simurg, we also utilize a questionnaire to isolate the variables that may make up each of the four Jungian personality types (on a 4-point Likert scale). When determining the relative importance of various external factors that may have an effect on a person's personality, this survey is employed. We use this data to determine an overall average weight for the questions (see Appendix G). Using the findings from our card game, we are able to identify the

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Volume 2, Issue 1, August 2022

Impact Factor: 6.252

quantitative form of different personality types. When people in the workplace reflect on the diversity in personality types among their colleagues, they often wonder how they may better interact with one another.

REFERENCES

[1]. R. Conradi and A. Fuggetta, "Improving software process improvement," IEEE Software, vol.19, no.4, pp.92–99,2002.

[2]. R. L. Glass, Facts and Fallacies of Software Engineering.Addison-WesleyProfessional,2002.

[3]. S. T. Acuna, N. Juristo, A. M. Moreno, and A. Mon, A Software ProcessModel Handbook for Incorporating People's Capabilities.Springer-Verlag,2005.

[4]. T. DeMarco and T. Lister, People ware: productive projects and teams. Dorset House Publishing Company,1999.

[5]. Y. Dittrich, C. Floyd, and R. Klischewski, Social thinking-software practice. The MIT Press, 2002.

[6]. M. Grechanik and D. E. Perry, "Analyzing software development as a non cooperative game," in IEE Seminar Digests, vol.29, 2004.

[7]. H. Van Vliet, "Editorial: Signs of a thriving journal," Journal of Systems and Software, vol. 86, no.1, p.1, 2013.

[8]. R. Charette, "Why software fails," IEEE Spectrum, vol. 42, no. 9, pp.42–49,2005.

[9]. D. Hartmann, "Interview: Jim johnson of the standish group," Infoqueue,Aug,vol.25,2006.

[10]. C. Jones, Software Engineering Best Practices: Lessons from Successful Projectsinthe Top Companies. McGraw-Hill Osborne Media, 2009.

[11]. J. E. Tomayko and O. Hazzan, Human Aspects of Software Engineering. Firewall Media, Dec. 2005.

[12]. G. M. Weinberg, The psychology of computer programming. Van Nos-trandReinholdNewYork,1971.

[13]. H. Robinson and H. Sharp, "Collaboration, communication and coordi-nation in agile software development practice," in Collaborative SoftwareEngineering.BerlinHeidelberg:Springer,2010,pp.93–108.

[14]. S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grunbacher, Value-based software engineering. Springer,2005.

[15]. Standish Group, "The chaos report," Available on-lineat http://www.projectsmart.co.uk/docs/chaos-report.pdf,1995.