A Capability Maturity Model for STP aware Software Development

[1]Geim Sllian and [2]Toi Mazur
[1]Center for Advanced Sttudies, European University Institute, Fiesole FI, Italy
[1]gsllianfie23@hotmail.com

Correspondence should be addressed to Geim Sllian : gsllianfie23@hotmail.com

Article Info
Journal of Machine and Computing (http://anapub.co.ke/journals/jmc/jmc.html)
Doi: https://doi.org/10.53759/7669/jmc202202022
Received 28 March 2022; Revised form 30 May 2022; Accepted 29 July 2022.
Available online 05 October 2022.

Abstract – There has been an increase in the importance of software Security, Trust, and Privacy (STP). Product systems must be designed with trustworthy STP protection methods while still rendering the required benefits of applications to its consumers. As a result of this large skill gap, colleges and the software sector have found themselves in a state of supply-and-demand conflict. STP-aware software development requires a new practice Capability Maturity Model (CMM) to address this issue. In order to help colleges progressively increase their students' capacity to apply what they have learned in the classroom, this contribution provides a model that consists of 4 levels: Awareness, Curriculum, Project, and Enterprise, for STP-aware software development. Software development that is STP-aware has been shown to be quite beneficial in the development of programming talent's practice capabilities for learners.

Keywords – Security, Trust, and Privacy (STP), Capability Maturity Model (CMM), Team Software Process (TSP)

I. INTRODUCTION

Traditional methods of system security are scarcely sufficing to mitigate the Security, Trust, and Privacy (STP) [1] challenges (such IDS and firewalls) today. These issues necessitate the development of STP-aware software applications. STPs are not adequately addressed in current software development practices. STP-aware software development is hampered by this issue. New technologies, such as blockchain, place additional demands on STP-aware software and apps. Because of the widespread use of Internet-based systems, attackers have multiplied and threat scenarios have shifted. Pervasive application software does not lend itself to traditional security measures. Pervasive applications necessitate a different approach when it comes to privacy concerns than traditional software applications. Ad hoc network connections and disconnection are commonplace among ubiquitous computing devices. This necessitates the development of new trust models for ubiquitous computing applications.

Teams and individuals can use the Team Software Process (TSP) [2] established by the Software Engineering Institute (SEI) to implement software development fundamentals. As compared to prevailing practices, software developed using the TSP has a defect rate of zero to one abnormality per thousand lines of code—that is, a defect rate that is one or two orders of magnitude lower. Secure Software Engineering (TSP-Secure) [3] is an extension of the TSP that focuses on software application security specifically. This project is a collaboration between the SEI's TSP and CERT programs. Developing a TSP-centric method that could dependability present secure software is the project's overarching goal. TSP-Secure takes a three-pronged approach to ensuring secure software development. First and foremost, TSP-Secure emphasizes the importance of planning for confidentiality. TSP-Secure also assists in the advancement of self-directed design groups and then places these teams in control of their own work because schedule demands and people issues prevent the implementation of best practices. TSP-Secure also aids in the management of quality all through the product development cycle because confidentiality and quality are intertwined. Finally, TSP-Secure includes security perception training for designers because people constructing secure applications must be aware of software security issues.

TSP-Secure allows teams to develop their own strategies. A project launch, a series of meetings lasting three to four days, is used to kick off the planning process. A competent team coach is in charge of the launch. As part of a launch, the TSP-Secure team comes together to agree on a typical set of security objectives and the approaches they will use to achieve those goals, and they get management support for the plan. It is common for the plan to include tasks like identifying security risks, gathering requirements for security, creating secure designs and code reviews, as well as employing static analytical techniques, system testing, as well as fuzz testing. During black-box testing, fuzz testing is used to generate random inputs for external program interfaces. The word is derived from the University of Wisconsin's fuzz testing program. Each member

of the team within the TSP-secure group has an option to select from one out of nine standard obligations (duties could be shared). The Security Manager position is one of the designated roles. All aspects of a product must be considered when it comes to security; the Security Manager ensures that these aspects are addressed during product development. He or she also provides timely assessment and disclaimer on security issues, and tracks any threats or issues to their resolution. An external security expert may be called upon when necessary.

To make sure that all security-related tasks are carried out properly after the launch, the team follows through on its strategy and puts it into action. Every management status briefing includes a discussion of security. Learners could visit web sites like Microsoft Security Software Consulting [4], US-CERT Technological Cyber Security Notification center site [5], MITRE Common Vulnerabilities and Exposures (CVEs) [6], and the SANS Institute's Top twenty list of security vulnerabilities site [7] to learn about the most popular software defects that lead to security flaws. TSP-quality Secure's strategic plan is to have numerous fault retrieval points during the software development process. With a greater number of defect removal points, it is easier to discover issues right away, allowing for faster problem-solving and a better understanding of the underlying causes.

Application products launched with lesser defects that might lead to threats have more defect discharge filters built into the development process. When defects are spotted earlier, reparative action can be taken before the software is even released. It was suggested in this study that a new model of practice capacity maturation for software engineering that takes STP into consideration may be developed. In order to help colleges progressively increase their students' capacity to apply what they've learned in the classroom, this model is based on 4 levels: Awareness, Curriculum, Project, and Enterprise. This paper provides a critical analysis of the model, which major focus on the four levels. The remaining sections of the paper has been organized as follows: Section II presents a background evaluation of the research. Section III discusses the ideas on the enhancement of practice capability. Section IV focusses on CMM review for STP-aware software development. Section V reviews the implementation of the STP-aware software development, which Section VI discusses the importance of teaching STP-aware software development. Lastly, Section VII draws conclusion to the whole paper.

II. BACKGROUND ANALYSIS

In recent years, international rivalry in the software business has grown, rendering it a strategic priority for many nations. The existence of a team of extremely qualified, complex, engineering-type professionals is the most crucial assurance for the long-term success of the software sector. A result of this is a focus on theory rather than practice in China's conventional educational techniques. Academia and industry face a significant supply-and-demand conflict due to the inability of graduate students to meet industrial needs. Many graduates are having problems finding work, while some software businesses have difficulty finding skilled personnel, even if they can afford to offer a respectable compensation. Accordingly, we undertook some study and inquiry into the training techniques utilized in nations with well-developed software businesses, such as India and Ireland. There is a great focus in these nations on the effect of practicing. The outcome is that the private sector is encouraged to engage with universities to guarantee that students have the skills and knowledge required to succeed in the job. If you want to be regarded a competent software developer, you need to have the following four qualities: basic theoretical skills, specialist knowledge, and expert skillset and carrier capability (see **Fig. 1**). Interpersonal skills, drive, and a strong work ethic are all part of this skill set, as are communication talents. The curriculum and practice systems are supposed to work together to create all of these characteristics.

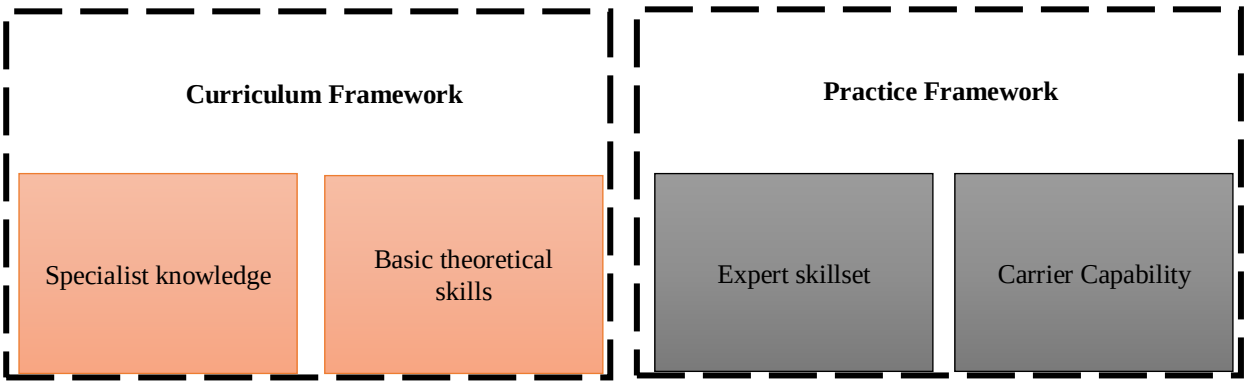


Fig 1. Expected qualities of a typical software development employee

CSDA Training Platform and Guidebook to the Software Body of Knowledge (SWEBOK) from Software Engineering Education Knowledge (SEEK) [8] and IEEE from ACM have both been brought to China as worldwide standard information systems for curriculum design and development. Every one of these worldwide standard information systems has a structured categorization of fundamental concepts and specialised information for many specialties. There is a difficulty in defining the maturity degree of professional training and career capacity in these fields of study, or improving the practice abilities of

learners, including in software engineering, through these curriculum structures. International application learning modes, coupled with our school's expertise and sentiments, have contributed to the growth of a practice maturity model for STP-aware software design that can be utilised to guide training in proficient training and career capacity. The model is divided into four stages of maturity: awareness, curriculum, project, and enterprise.

III. IDEAS OF IMPROVING PRACTICE CAPABILITY

For learners of software development, exercise and curricula are crucial components of a capability-building system, thus we should follow these notions while designing training programs for the subject. Training system applied in the field of Software Development at various universities. Both the syllabus and the exercise systems are intertwined according to semesters in the education program. CSDA (Certified Software Development Associate) [9] is used as a benchmark in Pedagogy System's knowledge base (see **Fig. 2**). STP-aware software engineering uses a practice maturation hierarchy that is quite close to the Capability Maturity Model (CMM).

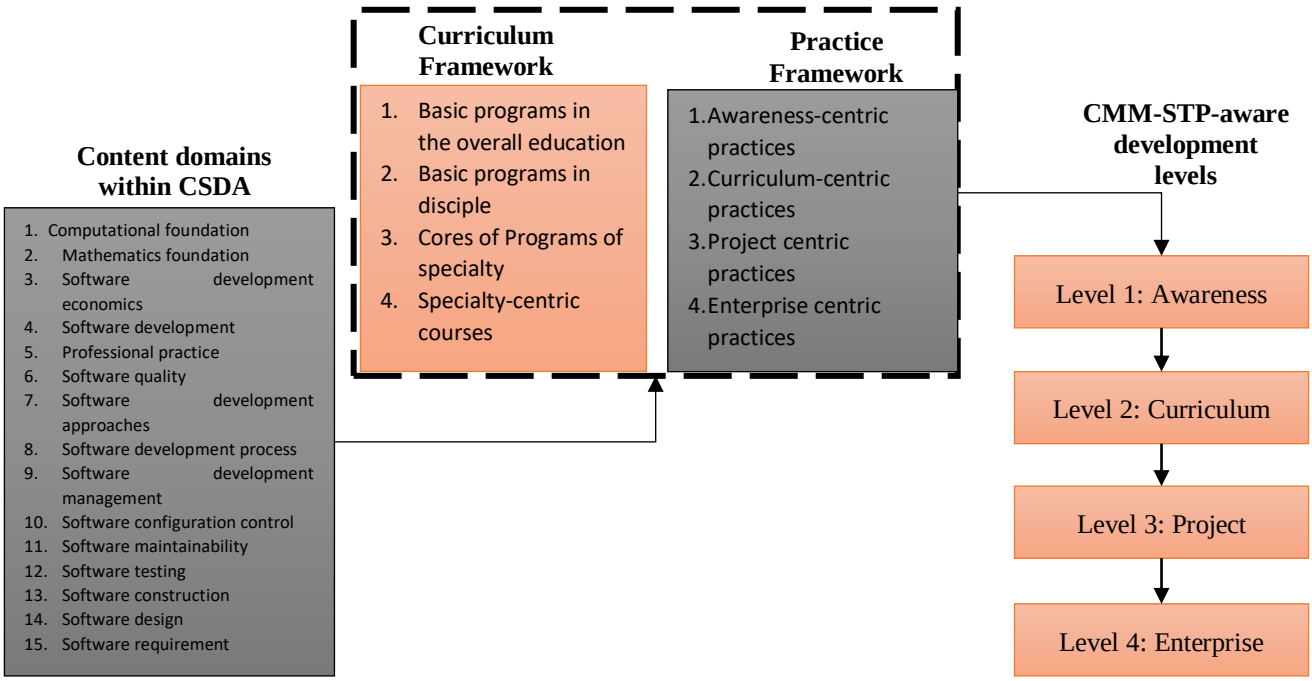


Fig 2. Cultivation framework employed in SSE

Following these guidelines will help us enhance students' practicing skills more effectively: (i) There will be a variety of material and objectives for each stage of STP-aware Software Engineering. (ii) Throughout the entire practice procedure, a standardized enterprise-like workflow is closely followed. (iii) Modern software development and project management practices are used in conjunction with cutting-edge development tools and technologies. (iv) As shown in **Table I**, the basic concepts of various practice activities are presented by stage.

IV. CMM – STP-AWARE SOFTWARE DEVELOPMENT

The phrase "STP-aware software development" refers to the process of transitioning from a novice to an expert. IT-related training programs, such software engineering, may find it useful in assessing students' practice ability. It's a fully-fledged replica.

The Development of STP-aware Software

Each time a pupil completes a practice task, their ability to practice improves more. Evolutionary paths to develop practicing skills follow maturity stages, which are well-recognized. If all of the primary process areas for a level are satisfied, the maturity level will rise to the next level.

CMMs may be used to compare the mature practices of different engineering disciplines. Using the model, an organization's current procedures can be evaluated and potential improvement areas can be suggested. The CMMs specify certain processes (software engineering, systems engineering, and security engineering) at a high level, however they do not give practical recommendations. They describe rather than define procedures; in other words, they explain what rather than how they accomplish things." Product or system certifications are not intended to be replaced by CMM-based evaluations." When it comes to assessing an organization's processes, however, organizational evaluations try to target just those areas that have been recognized as needing improvement.

Process maturity has typically been the focus of CMMs in order to achieve business goals such as better scheduling and quality management, as well as a reduction in the total defect rate in software, among other things. In terms of institutional and project operating procedures and verification methods, CMMs addresses two of the four main areas of concentration for a safe SDLC process. Insufficient attention is paid to security engineering and risk management It's not only the quantity of vulnerabilities that they're concerned with decreasing. In many cases, faults that aren't security-related don't cause security issues. One example of a security flaw that isn't the result of a software error is malicious code.

Table 1. Organization ideas for practice activities

Level of Maturity	Major Points within the Curriculum Model	Organization Ideas for Practice Activities
Awareness	Develop the model-level cognition of computing systems and data technology; training a number of basis knowledge for software engineering. The correlated programs integrate: College computing basics and the C programming language.	To conduct a number of major concepts of computing system. Within the programming concept, beginning to employ the latent framework and platforms, based on the mean timeframe, training the fundamental skills.
Curriculum	Learn the vital knowledge scheme concerning software development, which integrates data algorithm and structure analysis; principles of operating frameworks, and personal software procedures.	Developing the understanding concerning the foundational programs. Getting to identify and understand software engineering architectures and undergoing the software development cycle via practices and projects centered on the program projects.
Project	Training to employ technologies and theories of software engineering and software development to a particular form of application	Being acquainted with the information of the team's software procedure; naturing the capability of cooperation, innovation and communications.
Enterprise	Tracing more advanced skillset and the latest developments in the software development sector.	Undergoing the actual developing procedure of industry; accumulating knowledge or experiences of project management as well as utilizing front technologies.

Capability Maturity Model Integration (CMMI) may improve long-term business success for organizations. CMMI for Acquisition (CMMI-ACQ) and CMMI for Direct Contracting (CMMI-DC) are three unique configurations of the CMMI for Development (CMMI-DEV). Over 1,100 companies and 4,771 projects have been evaluated by the Software Engineering Institute since December 2005 [10]. Version 1.3 of the Capability Mature Model was released in each of its three constellations in November of that year. CMM offers a structure for acquisition groups to follow when acquiring products and services. In the process of providing, managing, and managing services, the CMMI for Services (CMMI-SVC) may be a beneficial tool.

Product and service creation, upkeep, and acquisition may all benefit from CMMI-best DEV practices, which include strategies to assist firms in improving their operations and criteria to measure process capacity and maturity. Software and systems integration, combined process and product design, and supplier acquisitions are all encompassed under this paradigm. CMM-DEV, which has been around for a long time and is well accepted, has replaced the CMM for software development. Since the mid-1980s, the CMM has been in use. The CMMI-DEV may be used to categorize process improvement and evaluation into four main categories.

In each category, there are a number of Processes that are included. Modeling and developing project management, supply chain management, quality assurance and measurement, as well as engineering methodologies, are all included in the CMMI-DEV (see **Fig. 3**). Risk management and engineering, and project and organizational security protocols are not specifically covered in this publication. Researchers continue to define new objectives and processes to assist assure the system's robustness even though the CMMI-DEV has already addressed many of these problems. The "Process and Practice Working Group" page may be found on the Software Assurance Society Tools and Data Clearinghouse website [11]. The SEI website has further information about the CMMI.

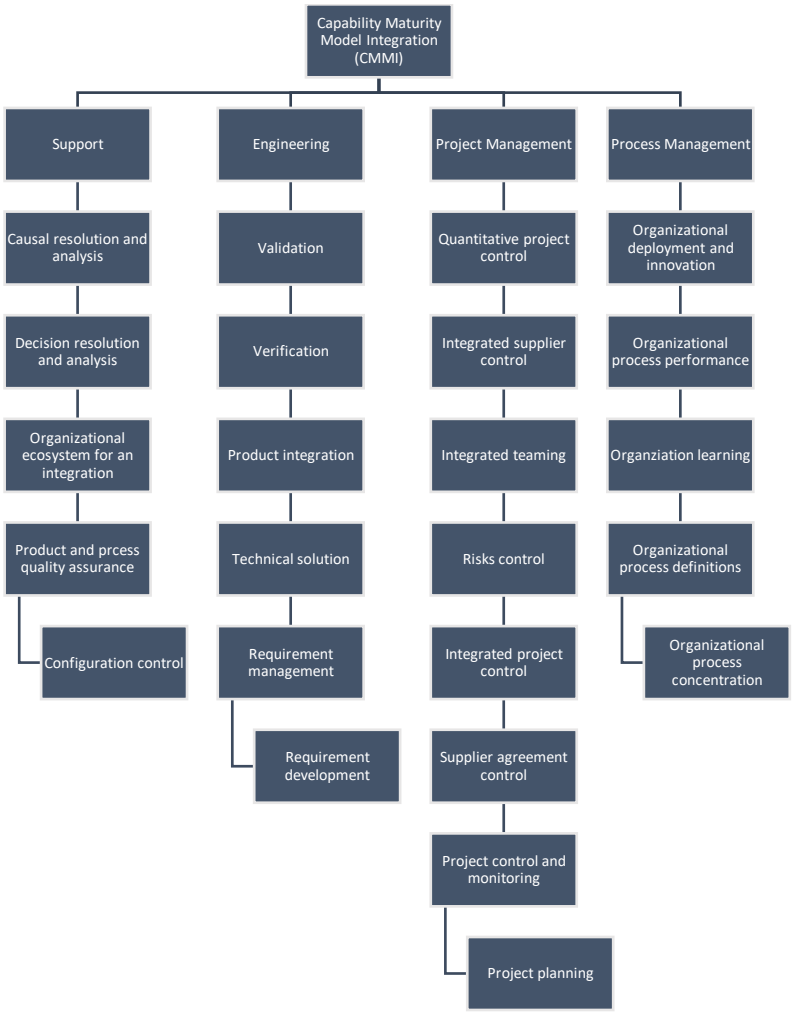


Fig 3. CMMI-DEV Process Areas

The Four Levels of STP-aware Software Enginneering

Fig. 4 depicts the main process areas at various stages of maturity, with a distinct set of Key Process Areas (KPAs) at every stage. The KPAs are a list of the most important criteria that must be met in order to reach a certain degree of maturity.

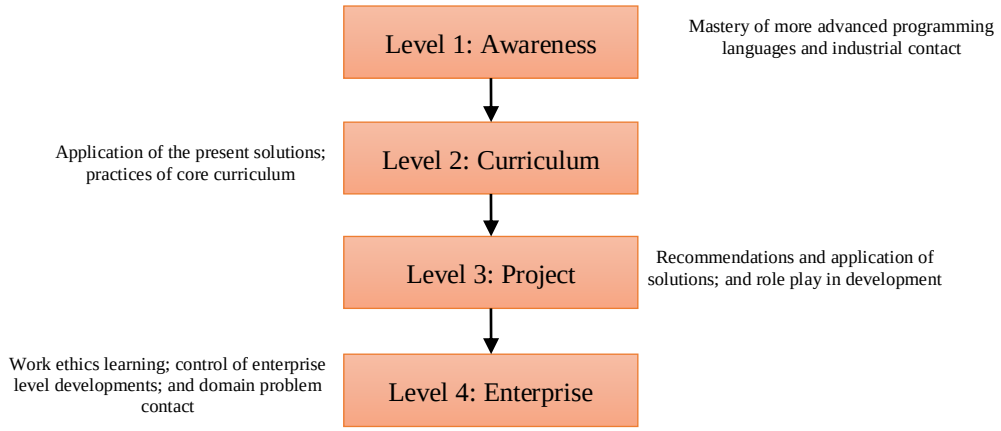


Fig 4. Vital process domains by the maturity level

Level 1 – Awareness Level

At Awareness Level, we presume that new students have no prior knowledge of software development or computer systems. Attending lessons in this level will help students get a basic understanding of their field. An overview of computer science

and an introduction to information technology are among the common courses.... A high-level programming language, such as Java or JavaScript, will also be taught to pupils. Generally speaking, students at practice capacity Level 1 lack a profound understanding of the software system and industry. Preliminary programming and computer system familiarization are the emphasis of this level of practice. The following are the most important process areas for the first level:

Expertise in Complex Programming Languages
The goal of this KPA is to teach students the fundamentals of computer programming while also giving them the skills to tackle basic programming challenges with the help of a more sophisticated programming language.

Business Relationships
With this KPA, students will have a better grasp of the software business and a fundamental concept of how software engineering may be used.

Level 2 – Curriculum Level
By the time students reach Curriculum Level, they have already learned the foundations of software engineering and are ready to go on to more advanced courses. Software Engineering, Information Structures and Algorithm Evaluation, Personal Process Stream, etc. are among the common courses. As a result, students' specialized knowledge and programming skills will significantly increase. Students at Level 2 will see a significant improvement in their ability to practice. There are thus both exercises in classes and the implementation of preexisting solutions to problems in a specific area, such as finance or gaming. As a result, students will have a basic understanding of project management concepts. Below are the main areas of focus for level 2:

Core Curriculum Application
Use this KPA as a teaching tool to help students learn how to go through programming exercises more effectively.

Use of Already-Developed Solutions
We want to show students how they can apply theoretical knowledge and programming expertise to put existing systems into practice.

Level 3 – Project Level
Scholars should have a good academic and practical basis by the time they reach Project Level. They will study core subjects in their field at this level, such as "Requirement Analysis," "Program Test," and "System Software in Practice," among others. For the first time, the practice material of this level is not restricted to a single area of expertise. Students begin working on projects of a smaller and more manageable scale. Teamwork, role-playing, and other aspects of the software development process will be covered. The vital process domains for level 3 are provided below:

Proposals and Application of Solution
This KPA's goal is to teach students how to study and analyze issues in a specific area, and then devise and execute their own solutions.

Roles in Development Process
These students will be guided through various responsibilities in a design team, and they will be encouraged to strengthen their communications and teamwork skills.

Level 4 – Enterprise Level
Students are prepared to operate as software developers at the Enterprise Level. At this level, students are mostly exposed to specialized courses, such as lectures on expert knowledge and the most recent developments in industry. This level's practice material focuses mostly on teaching students how to enhance their professional prospects. Students will participate in a variety of roles in a real-world software development environment and attempt to suggest and execute ideas that are relevant to the industry at large. Listed below are some of the most important things to focus on in level 4.

Domain Problems Contact
It is the goal of this KPA to assist students in conducting in-depth investigations and study into industrial challenges in order to develop solutions that can subsequently be put into action after they have been developed.

Enterprise Level Management and Development
KPA's goal is to provide students a first-hand look at what it's like to manage software engineering in a real-world software company.

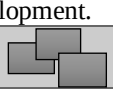
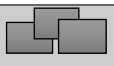
Training on Work Ethics

It is the goal of this KPA to instill in the pupils a strong sense of responsibility. When the learners begin to work, they will be committed to the enterprise's goals and the welfare of their coworkers.

Model Analysis

Table 2 shows the aims and aspects of STP-aware software development, which we shall examine further after a quick introduction to the four phases of STP-aware development.

Table 2, Aims and aspects of STP-aware software development

	Awareness	**Curriculum**	**Project**	**Enterprise**
Practice objectives	Freshmen years; gaining accessibility to the software sector; and stimulating the interests in software development.	Sophomore years; training them the ideologies of software development cycle as well as vital skillset of developmental life cycle.	Junior years; make scholars comprehend team development processes, e.g., role plays and cooperation.	Senior years; learning the capability of learners to role play considering the development of enterprise level projects and proposals and implementations of industry-centric solutions.
Practice elements	Preliminary programming practice; initial contacts with various applications in the industry; engaged in small-size projects with minimal control; time-consuming about two to three weeks.	Course centric application practices; related to the small-sized issues in a particular domain; engaged in simple project control framework; time-consuming three to five weeks.	Practice objectives acting within the industry; concerned on the issues with special roles in industries; engaged in more complex project control; time-consuming about six weeks.	Practice work internships and placements in software sector; concerning on issues with multiple objectives within the industry; engaged with more complex project control; time consuming about two months.
Vital Process domains	Mastery of more advancement programming languages; industrial contacts.	Practice of the core curriculum; application of prevailing resolutions.	Proposals and application of resolutions; role acting in engineering.	Domain problems contact; work ethics education; control of enterprise level developments.
Curriculum objectives	Construct primary cognition of computer sciences and learning basic concepts of software engineering	Learning vital information of application growth and having an access to the theories of application development.	Deploying training information and technologies to software engineering.	Learning advanced knowledge and tracking the latest trend of industries.
Representation forms				

Row 6's "Representation Forms" in **Table 2** demonstrates how practice systems influence the application of fundamental ideas and specialized knowledge. Student knowledge at the Awareness Level has a poor correlation, indicating a dispersed pattern. Students may apply what they have learned in their fundamental courses to more sophisticated challenges at the curriculum level, where the information they've acquired shows a consistent pattern. At the Project Level, students are able to build small and medium-sized initiatives by flexibly applying their knowledge of a given topic. Enterprise Level students begin to apply their knowledge and skills to industry-related issues, and they will participate in enterprise-level projects employing contemporary evolving technologies and equipment supported by specialized corporate culture.

V. IMPLEMENTATION OF STP-AWARE SOFTWARE DEVELOPMENT

Training in SSE's software engineering discipline has been based on grading in STP-aware development for some years now. STP-aware development is implemented using a variety of diversity training methods, such as using industry-oriented projects or project cases, utilizing the latest development tools and technologies, and occasionally having industry guest

lecturers give speeches. These relationships with renowned software companies allow for this variety of diversity training methods to be used during implementation.

Awareness Level

Practice at Awareness Level is mostly geared at preliminary programming. It is possible to organise this period in a centralized, distributed, or decentralized manner. A two-week, three-week and three-week duration is available for each type. There are six unique steps of implementing standard awareness, each with its own characteristics, which allows us to break down the process into manageable chunks. These phases include starting, project initiation, implementation, and testing. Because the breadth of practice activities is so limited, students need to be given a detailed description of the project's needs and system design.

Curriculum Level

About a month's duration of Curriculum Level practice training is required, and the implementation phase may be structured in any of three ways: centrally, dispersed, or decentralized. The period might range from 3 weeks to 5 weeks depending on the style selected. We divided the standardized procedure into eight steps based on the characteristics of this level and Personal Software Process 3: beginning, starting a project, planning a project, designing and executing the systems, testing, approving the result, and concluding. Students will be provided a detailed summary of the project's requirements at the outset. For this reason, students only need to focus on high-level designs in project design, and not on outline designs, due to the small size and intricacy of the practice exercises.

Project Level

There are nine phases of Project Level practice training, which lasts roughly six weeks and is broken down into the following: opening, program start, project plan. We advocate a combination of centralized and dispersed training methods at this level.

Enterprise Level

Enterprise-level training takes around two months to complete and is broken down into nine stages: beginning, starting the project, planning the project, analyzing the requirements, designing the system, implementing the system, testing it, and finally accepting the project. The training program should be centrally coordinated at this level. Initiatives at this level will take done at a practice facility, a unique location made possible by school-business partnerships. Tutors from a variety of different software businesses serve as coaches for the various practice teams.

VI. SIGNIFICANCE OF TEACHING – STP-AWARE SOFTWARE DEVELOPMENT

Understanding Secure Development Life Cycle

SDL (Secure Development Life Cycle) is a technique of incorporating security considerations into the normal Software Development Lifecycle (see **Fig. 5**) utilized in many firms building software and systems. Agile, Scrum, and Waterfall are just a few of the methods in which SDL processes have been used by numerous developers. There are several additional methods of development, such as spiral, fast software engineering, agile development approach, test-driven advancement, and others. In terms of critical stages, these models are frequently comparable, such as: 1) planning, 2) development, 3) certification, 4) implementation, 5) administration, and 6) end-of-life stage 6.

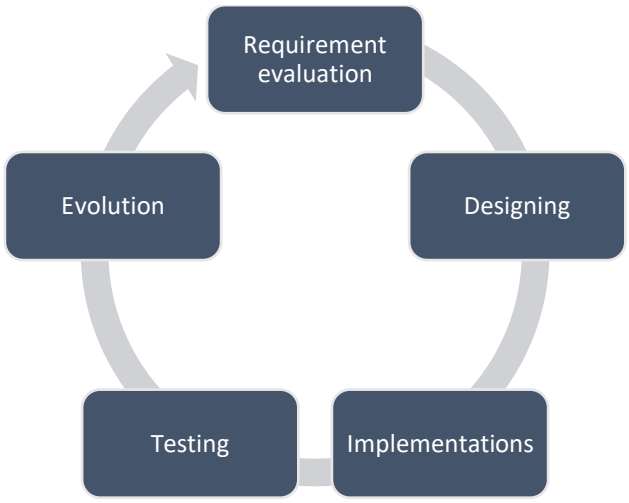


Fig 5. Software Development Life Cycle (SDLC)

Developers tend to focus on security considerations during the testing phase of the software development process, when they uncover faults and gaps in the system. One of the worst practices when it comes to security is evaluating the security of

software that was not designed to be secure in the first place, which is one of the worst practices. Instead, resources would be spent testing and strengthening security rather than deploying the software on time. When it comes to implementing security in SDLC, the notion of Secure SDLC is becoming more and more commonplace. When developing software, security-related elements should be considered as much as feasible throughout the whole process.

According to [12], the phases of the Software Development Life Cycle (SDLC) do not accurately represent the rules and procedures that must be applied at every management level. This condition might lead to embedded risks and weaknesses in the system. When it comes to this situation, the Secure Development Lifecycle (SDL) can be integrated into an organization's software development process. System Development Lifecycle (SDL) advocates the notion of incorporating security measures into all stages of the development process. To meet the needs of the customer, most software developers write code without thinking about the security risks and vulnerabilities that could be present in their work. The only way to prevent malicious behaviors like denial of service and data leaking is to include security safeguards in the SDLC.

For example, security may be measured at each stage of the SDLC. Sahraoui [13] shows that every stage of SDLC should provide functionality as well as a high level of quality and security. The software developer's goal is to supply all of the features as quickly as possible. It may take more time, money, and resources to fix a security vulnerability-caused casualty. For a safe application, you must include security measures into your software development lifecycle. A longer development period is required, and there is no assurance that the final product will be impenetrable from an assault. However, using the approach reduces the likelihood of an attack occurring. A high-quality, safe piece of software is the ultimate result. So, the company's reputation in the eyes of the public and its peers will rise.

Risk management is a security step that must be taken at the beginning of the requirements gathering process. Determine, access and limit the danger that arises from the functions and processes. Static analysis checks the system code for appropriate coding convention and compliance with industry standards. FindBugs, developed by Huang, Shao, Fan, Yu, Yang and Zhou [14], is a well-known Java static analysis tool. Security testing and code review follow static analysis as an additional step. Authentication, authorisation, access control, validation of data, error detection, logging, and encryption are the eight components of a secure code review.

Assuring a secure setup and conducting a security evaluation complete the process. Security methods may be implemented or updated to counter the threats and vulnerabilities discovered during the analysis phase. For example, encryption techniques and protocols, authentication methods, user rules, and legislation are all examples of security mechanisms. The human user is the primary source of system vulnerability. Controlling and regulating user behavior might be accomplished by user security policy.

Understanding Software Security Weaknesses

Throughout the software development process, developers must be able to discover all of their security flaws. For a deeper comprehension, Perlmutter and Frankel [15] compiled a large number of terms from various literature studies on software security. One of the terms used is Asset, which refers to products or assets that are critical to the firm and must be protected. The assets are stated to be the primary and primary goals of a threat or assault, and if such are disclosed, it might suffer major repercussions. The program must be able to protect against hacker assaults or ensure that it does not rely too much on data kept outside of the asset's boundaries.

A software vulnerability is a flaw in the security system's implementation that might lead to the program's detriment. It is a design defect or a loophole in the system itself. Software vulnerabilities may be classified into two types: design flaws and observable bugs in the code. Design flaws are the root cause of most vulnerabilities, although implementation bugs can also lead to vulnerabilities. Although the attacker has located the weakness, the assets have not been compromised, hence this poses a danger to the system. The flaw in the security program itself is another weakness that might be compared to a vulnerability. This may be due to a design error in the software's security features. While vulnerabilities may be patched and redesigned, this defect cannot and has to be completely redone from the ground up. Defects should be discovered as soon as feasible so that they may be addressed and remedied throughout the improvement process.

Understanding Problems in Secure Coding Practices

While developing software, one must make sure that all of the software development's inner workings are secure and ready before beginning the real development process. Security has been an issue in many situations since it was not taken into account during the design process. The issue of safety is often on people's minds, but they don't take it seriously enough. There are a number of possible reasons why secure coding principles are being flouted. One of the most prominent causes of poor security procedures is a lack of coordination between development teams. It's not uncommon for a team to have a misconception about how the software will be produced in the workplace, no matter what industry they operate in (for example). Moreover, as remote working becomes more prevalent across the globe, the majority of respondents said that working in a team is preferable than working alone.

In this case, it is critical that engineers and other stakeholders communicate openly and clearly to ensure everyone is on the same page. Another issue is the absence of testing of software. In the Secure Software Development Life Cycle, improper development scheduling techniques may be to blame for this. When the program is deployed, there will be additional zero-day attacks because of the absence of testing. Due to time and effort limits, it may be difficult to find and fix bugs. When testing is necessary, it should be done in a collaborative manner rather than by a single person just for the purpose of speed

or haste to achieve a deadline. Many scholars e.g., those in [15] have conducted surveys and evaluations on this issue, which may be useful for comparing security measures from the recent past and the present.

VII. CONCLUSION

A curriculum for Security, Trust, and Privacy (STP-aware) software development has been used for years in the Schools of Software Engineering (SEE) field of software engineering to increase student practice capabilities. Two benefits of STP-aware software development should be noted. On the other hand, the pupils had received a number of high-profile awards. To progressively increase students' practice abilities, this research presents a revolutionary software talent practice for STP-aware software development. This training may be used as a standard to manage, implement, or lead the practice process of students in software engineering training. It may also be applied to the practice of a wide range of other disciplines. Years of STP-aware software development in the Software Engineering Discipline of SSE have proved that it is very important for the training of students' practical capabilities in the field of engineering. In the software sector, STP-aware software development improves the skills of software developers.

Data Availability

No data were used to support this study.

Conflicts of Interest

The author(s) declare(s) that they have no conflicts of interest

References

[1]. B. McMillin and T. Roth, "Cyber-Physical Security and Privacy in the Electric Smart Grid", Synthesis Lectures on Information Security, Privacy, and Trust, vol. 9, no. 2, pp. 1-64, 2017. Doi : 10.2200/s00784ed1v01y201706spt021.

[2]. G. Coleman, "Organizing the rable - Introduction to the Team Software Process[Book Review]", IEEE Software, vol. 17, no. 6, pp. 109-110, 2000. Doi : 10.1109/ms.2000.895179.

[3]. D. Kaur, P. Kaur and H. Singh, "Secure Spiral: A Secure Software Development Model", Journal of Software Engineering, vol. 6, no. 1, pp. 10-15, 2011. Doi : 10.3923/jse.2012.10.15.

[4]. "Security Consulting Services", Microsoft.com, 2022. [Online]. Doi : https://www.microsoft.com/en-us/securityengineering/sdl/consulting. [Accessed: 15- May- 2022].

[5]. "Homepage | CISA", Cisa.gov, 2022. [Online]. Doi : https://www.cisa.gov/uscert/. [Accessed: 15- May- 2022].

[6]. "CVE -CVE", Cve.mitre.org, 2022. [Online]. Doi : https://cve.mitre.org/. [Accessed: 15- May- 2022].

[7]. M. Haase, R. Auger and K. Wyk, "Top 25 Software Errors | SANS Institute", Sans.org, 2022. [Online]. Doi : https://www.sans.org/top25-software-errors/. [Accessed: 15- May- 2022].

[8]. L. Druffel and R. Little, "Software engineering for AI based software products", Data & Knowledge Engineering, vol. 5, no. 2, pp. 93-103, 1990. Doi : 10.1016/0169-023x(90)90006-y.

[9]. "Certified Software Development Associate Certification", IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 10, pp. 1546-1546, 2010. Doi : 10.1109/tpds.2010.147.

[10]. "INCOSE Position on Capability Models and the Capability Maturity Model Integration (CMMI) Effort", INSIGHT, vol. 2, no. 2, pp. 19-20, 1999. Doi : 10.1002/inst.19992219.

[11]. S. Shang and S. Lin, "Understanding the effectiveness of Capability Maturity Model Integration by examining the knowledge management of software development processes", Total Quality Management & Business Excellence, vol. 20, no. 5, pp. 509-521, 2009. Doi : 10.1080/14783360902863671.

[12]. G. Nedeltcheva, "Quality Measurement in the Software Development Life-Cycle by Statistical Metrics – A Survey", Lecture Notes on Software Engineering, vol. 3, no. 2, pp. 145-151, 2015. Doi : 10.7763/lnse.2015.v3.180.

[13]. A. Sahraoui, "The rationale paradigm in system development lifecycle", International Journal of System of Systems Engineering, vol. 4, no. 1, p. 44, 2013. Doi : 10.1504/ijsse.2013.053479.

[14]. Z. Huang, Z. Shao, G. Fan, H. Yu, K. Yang and Z. Zhou, "HBSniff: A static analysis tool for Java Hibernate object-relational mapping code smell detection", Science of Computer Programming, vol. 217, p. 102778, 2022. Doi : 10.1016/j.scico.2022.102778.

[15]. A. Perlmutter and B. Frankel, "SECURITY STUDIES and Security Studies", Security Studies, vol. 1, no. 1, p. iv-iv, 1991. Doi : 10.1080/09636419109347452.