https://dx.doi.org/10.22161/ijaers.5.7.21 ISSN: 2349-6495(P) | 2456-1908(O)

A Case Study on Identifying Software Development Lifecycle and Process Framework

N. Devadiga

Abstract*—*This paper analyzes and determines which software development lifecycle and process framework would be appropriate in the following case studies: Microsoft office business unit, Denver Baggage, Avionics development, and Department of Transportation. The analysis for decision takes into consideration the stakeholders involved, the targeted audience, technology, business drivers, culture, time/schedule, resources, scope, and quality.

Keywords*—*Software process, SDLC, Software framework.

I. INTRODUCTION

Technology is taking over the world at a rapid pace. With the exponential growth of technology across diverse industries, software solutions have become essential in every facet of the business. Because of the size and complexity of current software, there is a need to have a guiding process for development. However, with such diverse applications of software, there is a need to determine the appropriate process in the context of the situation. This paper investigates the following four case studies.

II. CASE STUDY: MICROSOFT OFFICE BUSINESS UNIT (OBU)

Microsoft released Word for Windows word processor in 1989, after five years of development. The product received significant acclaim, and the sales concluded higher than Microsoft's projections, however, the project faced several project management issues in its execution [1]. The project had issues ranging from ill-defined requirements, lack of planning, inadequate project management, and random role assignment [1]. Most of Microsoft's products at that time were among the best available products on the market. Though the product was built in Microsoft's standard style, Office Business Unit project needed a structure - a process framework to guide the development.

Following points were noted before identifying the software process and framework:

Microsoft work culture: The work culture at Microsoft at that time was informal - software engineering staff handled project execution decisions; roles were interchangeable, and projects were carried out without formal requirements documentation.

Microsoft's release strategy: Microsoft's preferred strategy was to deliver the product in many small releases with short durations.

Time constraint: The initial project was scheduled to be delivered in one year.

Focus on programming: Microsoft's projects at that time relied heavily on programming aka build and demonstrate model. It had always worked for them in the past. Developers and managers were not very concerned with the software architecture or process methodologies.

Small team sizes: Development team size was typically limited to 10 people.

Unclear requirements: The requirements for the project were not well defined. Microsoft wanted to add as many innovative features in the word processor, without defining the project scope.

Based on the above factors and as per [2] an agile process like Extreme Programming (XP) [14] would be a better fit for the OBU project. Below are potential reasons as to why extreme programming would serve the project better:

Time criticality / Small releases: The primary focus of Microsoft was to release the product to market as soon as possible. With XP, it could be achieved by releasing an early version of the software and then incrementally adding functionalities to it with later releases. Such incremental deployments are not feasible with traditional software processes like Rational Unified Process (RUP) [11], due to its monolithic development style. With Agile, the product can be built incrementally; particularly with Extreme Programming (XP) process, a simplistic model of the system is released to production and newer versions are released in short cycles.

Undefined requirements: Since the requirements were unclear and volatile, it makes sense to choose an agile process that could quickly respond to changes.

Code-centric development: The nature of the project suggests that it was going to be code intensive (a word processor with many innovative features). Also, at Microsoft, significant emphasis was on programming, rather than on system architecture documentation. In XP, programming forms the core, and it allows programmers to take decisions about the design. This would have worked well with engineers like Hunt – one of the

https://dx.doi.org/10.22161/ijaers.5.7.21 *ISSN: 2349-6495(P) | 2456-1908(O)*

programmers responsible for deciding on the features for Word.

Informal work culture: Traditional methodologies are rigid and do not work well in informal settings [13], but XP can work very well in such configurations. For example, pair programming, one of the tenets of XP, can be beneficial when developers are comfortable working closely with each other. Small team sizes at Microsoft could support such practices. Furthermore, it is unrealistic for them to use a cumbersome process such as RUP which requires a highly structured and complex team with many roles and requires tool support.

Focus on quality: Bill Gates wanted this to be the "best word processor ever" and much time was to be spent on getting every feature right. Characteristics of XP such as refactoring (restructuring the program to improve quality) and continuous testing (continuously writing unit tests, which must run for the development to continue) would serve this purpose greatly.

Having a working system at all times: Some Microsoft managers were of the opinion that a "shippable" product should be available at all times – after a piece of development is complete, all error and boundary cases should work, and it should successfully integrate with the rest of the system. XP facilitates just that with continuous integration. It says that the system should be built many times a day, every time a task is completed.

User collaboration: Since the market focused on multiple large business corporations and government agencies, the way to elicit requirements should be through user collaboration. An iterative process is required to elicit user requirements and feedback. An agile process like XP best does this.

Extrapolating the engineering culture and project management structure at Microsoft an iterative and incremental lifecycle with a light, agile process like XP would be a good candidate for the MS Word project by providing structure for new requirements, delivery under time constraints, and code-intensive development.

III. CASE STUDY: DENVER BAGGAGE SYSTEM

Before determining what process and framework would be useful for the Denver Baggage System (DBS) [10], notes are taken on the nature of the project. There are several stakeholders on the DBS project, and each has their expectation for the system (see the table below).

Stakeholder	Need
Airport	The project must be completed on time as delays cost money
Airlines	Planes must be loaded as quickly as possible
Passengers	The system must be accurate, so bags are not misplaced
Airport Staff	The system must not break since there is no backup in place

The needs of the stakeholder's lead to the project's requirements. Based on the date the airport is scheduled to open, the project must be completed within 22 months. It has to be entirely accurate for bags to be delivered to the right place. It cannot have any downtime. It also has to move the bags physically faster than any other system before, which allows planes to have a faster turnaround time. However, the system is far too complex to design and implement within the desired time window. As Neufville pointed out [8], planning the people mover in the Atlanta airport was the subject of two years of research and a doctorate dissertation, and that system was comparatively simple. As the development cannot realistically be completed within the scheduled timeframe, it is assumed the DBS is delivered in increments to have a working system eventually.

By studying the system requirements, of the Denver Baggage System, it seems the creation of the Denver Baggage System would be best handled with a traditional Rational Unified Process (RUP) framework [11]. RUP is appropriate for a variety of reasons:

RUP puts a strong emphasis on the design of a system, this is required as the complexity of the system requires thorough planning.

RUP promotes component-based architecture which enables modeling of real-world systems and integrates well with the development of those systems [2]. This is very important for the DBS project since the physical design of the DBS is constrained by the architectural design of the airport and the physical realities of the conveyor system.

RUP process is designed for delivery in increments. As explained before, it is not possible to deliver the entire system in working condition by the deadline. Delivering some sub-portion of the system should be possible. RUP's incremental delivery design allows the system to expand as it is developed.

Delivering the system in increments forces the creation of a manual backup system. Some bags would have to be manually transported to the terminal until the entire system is online. This helps maintain system reliability because if the system fails, there is a process and procedure for replacing the lost functionality.

Due to many investors in the DBS project, project accountability is a requirement. The extensive documentation and artifacts produced by RUP provide the accountability mentioned above.

When comparing RUP to other process frameworks, it is apparent why, in this case, it is the superior process. RUP

https://dx.doi.org/10.22161/ijaers.5.7.21 ISSN: 2349-6495(P) | 2456-1908(O)

has advantages over more agile frameworks like XP in that this project is very design heavy. Much planning is needed to ensure that all the parts of the system integrate together successfully.

RUP is better than waterfall-style processes since the DBS project needs incremental deliveries not present in those frameworks.

The DBS project does not need the risk management of a spiral process since the risk is managed by the forced development of a backup system.

IV. CASE STUDY: PENNDOT21

The goal of the PennDOT21 project is to provide on-line vehicle registration services by making a web interface for the PennDOT registration system [12]. This system should be a secure and easily accessible service to all licensed Pennsylvania drivers. The critical factors in determining a lifecycle for this project are as follows:

Stakeholders: The significant stakeholders include the Pennsylvania Department of Transportation, its employees who work with the system, and all licensed Pennsylvania drivers. Because the technical competence of the end users varies, the web interface must provide a highly accessible and intuitive GUI. It suggests an iterative lifecycle with feedback to determine GUI requirements.

Market: We assume that the PennDOT21 system is mostly the first of its kind and therefore may serve as an example system for other states in the future, this suggests a process with clear indicators of progress.

Technology: PennDOT21 to provide an interface with the older PennDOT vehicle registration system. Thus, there must be proper testing to ensure that this integration is secure and robust.

Business Drivers: The business goal of this project is to reduce errors and work required in the existing manual registration process. Because a manual process already exists, this suggests a backup exists for PennDOT21 in case of failure and also that continuous deployment is possible.

Culture: End-users and employees are unaccustomed to using the web interface. Thus, a gradual deployment with training is required for a successful project.

Time/schedule: A time constraint is not a primary requirement of this project because there is already a manual process by which drivers can register their vehicles. Since the interface is dependent on the manual process, any changes in the manual process might affect the schedule.

Scope: The scope listed in the project description only covers an interface for vehicle registration. However, it is feasible that the scope might be extended in future projects by the DOT if the project is successful (since the DOT covers many more functions than just vehicle registration). Thus, PennDOT21 should be modular and modifiable.

Quality: One of the main concerns for the PennDOT21 is security, as transmitted data might include sensitive information such as registration numbers. Furthermore, the system must provide 24x7 access and thus must be error-free and robust. Concurrency and scalability is an issue, since there may be a large number of users accessing the system at one time.

From the above factors, the most critical project requirements are summarized as follows:

- Robust, secure, scalable, modular and modifiable back-end communication with PennDOT.
- Intuitive and flexible, but secure front-end web interface.
- Clear indicators of project progress.
- Extensive testing to ensure code integrity.

The points above show a dichotomy in the requirements for this project. On the one hand, the robust back-end suggests a traditional process with particular attention to design and architecture. On the other hand, the easy-to-use front-end suggests an iterative, agile process with extensive feedback to make the interface as intuitive as possible. Therefore, the best fit process is a merge of both agile and traditional processes.

ACDM [7] with Rapid Prototyping [9] provides the best fit for this project. ACDM's architecture-centric approach gives the best chance of success in fulfilling the need for a robust, secure, and scalable system. Furthermore, ACDM provides a clear way to track progress by use of the architecture [3], even though rapid prototyping itself may not produce clear progress indicators. Rapid prototyping is used in the production phase because its attitude towards changing requirements and extensive feedback allows it to provide an intuitive and easy-to-use interface. ACDM guides the development, so there is no loss in security or robustness. Furthermore, rapid prototyping's code-centric attitude ensures a minimum of bugs, and this is especially true for PennDOT21 which would be a small or medium software size [2].

ACDM with Rapid Prototyping [9] is the best possible process for this project. Security, scalability, robustness, and modifiability are all attributes that are addressed while examining the architecture of a project. Furthermore, PennDOT21 is not a life-critical system, and has a backup manual registration service (as assumed), so heavyweight processes like Spiral or RUP are not essential to its development. Next, ACDM should be combined with an agile process for development since the exact requirements for an intuitive web interface cannot be well-defined early in the project. In this case, Rapid prototyping is the best agile process to combine

https://dx.doi.org/10.22161/ijaers.5.7.21 *ISSN: 2349-6495(P) | 2456-1908(O)*

with ACDM because of Rapid prototyping's code-centric approach and attitude towards changing requirements. Another approach like scrum might focus on the management side, which may not be necessary for this project (depending on the specifics of the development team).

V. CASE STUDY: FLIGHT CONTROL SYSTEM

The aircraft flight control system (FCS) is a high-risk flight system that controls every aspect of an airplane operation to ensure safer, smoother flight; it consists of the flight control surfaces, cockpit controls, and the necessary mechanism to control the aircraft's direction in flight.

FCS requires:

- Good aircraft handling properties
- Low pilot workload
- Model simulation or prototyping is required to analyze whether digital processing signals represent the desired implementation, to avoid any mishap during the ground or flight testing[4].
- Backup or failover plan in case of software or hardware fault.
- The system developed should be comprehensively tested for an extensive set of faults and have thorough ground-based testing. The system and its inherent functional design should be free from errors.

Additionally, FCS requires adherence to the highest level of quality standards. Any failure in the system can cause loss of aircraft and human lives; the probability of success should be very close to 100%. However, a test to prove 100 percent correctness is almost impossible. Thus, a trade-off is done by deploying many reliable, redundant artifacts, a thorough design and development process, and test-cases under all possible combinations of inputs. Redundant artifacts would be used as backup during any software fault.

The project is high risk, safety-critical, and requires zero defect deliverables along with continuous risk assessment. Thus, a spiral model is proposed as the software development process along with six sigma business management strategy. This gives a combination of prototyping, continuous refinement and near-zero defects.

Here are all of the factors taken into consideration:

Stakeholders:

- Pilots, Passenger
- FAA (Federal Aviation Administration)
- Airlines
- Market:
  - Private and military avionics industry

Technology:

- Real-time, Embedded
- Communication between each device has to be near real-time

Business drivers:

- Early generations of FCS were mechanically based, so pilots had to physically steer and control the aircraft, which was limited by the physical capabilities of the pilot [4].
- Development of digital FCS would automate the process.
- Increase in safety as the pilot can concentrate on high-level tasks rather than routine control tasks.

Culture(s):

The spiral model [5] along with Six Sigma strategy is a good fit for the project. The project would consist of interactions between software engineers, embedded systems developers, six sigma black belt members (to aid high quality and defect free deliverables), testers, change management group (risk, impact analysis and versioning), analysts and pilots (for live testing of the system).

Time/schedule, resources, scope, and quality:

This project, being safety critical, requires thorough testing, simulation, high-quality standards, zero defects, and adequate documentation. The spiral model incorporates the above requirements with a fast-iterative approach, and a team of six sigma competent members would work on quality, risk management, cost, and estimation in sync with spiral model phases. Hence, the spiral model fits the project well.

Six Sigma:

Due to the lack of emphasis on documentation with the spiral model, its weakness is strengthened by combining it with Six Sigma strategy. Six Sigma [6] improves the quality of process outputs by identifying and removing the causes of defects and minimizing variability in manufacturing [7]. In a Six Sigma process, 99.99966% of the product is expected to be bug-free. The five phases of six sigma process are defined, measure (identifying critical to quality and risks), analyze (high-level design), design (simulate and optimize) and verify (set up pilot runs). This along with the spiral model would provide a thoroughly tested, well documented, bug-free, high-quality deliverable.

Considering that the key for developing aircraft flight control is safety, we have concluded that the Spiral process is the excellent fit for this project. Spiral model encapsulates iterative development with prototyping, verification and validation, and a waterfall approach in

https://dx.doi.org/10.22161/ijaers.5.7.21 *ISSN: 2349-6495(P) | 2456-1908(O)*

incremental order. Finally, six sigma provides the documentation that the spiral model lacks, as well as ensure further quality control to the highest level.

VI. CONCLUSION

We have seen in the above four cases that different circumstances can call for very different development models. High-risk applications such as the Flight Control System require traditional models with features such as risk assessment and thorough testing or simulation. On the other hand, products in a highly competitive market, such as MS Word, might require a more agile process for faster time to market. Many factors such as stakeholders, business culture, technology, and risk must be considered for selecting the most appropriate model, and a full analysis of any project should be carried out before selecting a process.

REFERENCES

[1] Gill, G., &Iansiti, M. (1994). Microsoft corporation: Office business unit. *Harvard Business School Case Study*, 691-033.

[2] Tsui, F., Karam, O., & Bernal, B. (2016). *Essentials of software engineering*. Jones & Bartlett Learning.

[3] Lattanze, A. J. (2005). *The architecture centric development method*. Carnegie Mellon University, School of Computer Science [Institute for Software Research International].

[4] Pratt, Roger W. "Flight Control Systems: Practical Issues in Design and Implementation, 2000." *The Institution of Electrical Engineers*.

[5] Boehm, Barry. "A spiral model of software development and enhancement." *ACM SIGSOFT Software engineering notes*11.4 (1986): 14-24.

[6] De Feo, Joseph A.; Barnard, William (2005). JURAN Institute's Six Sigma Breakthrough and Beyond - Quality Performance Breakthrough Methods. Tata McGraw-Hill Publishing Company Limited.

[7] Antony, J. (2004). Some pros and cons of six sigma: an academic perspective. *The TQM magazine*, *16*(4), 303-306.

[8] De Neufville, R. (1994). The baggage system at Denver: prospects and lessons. *Journal of Air Transport Management*, *1*(4), 229-236.

[9] Devadiga, N. M. (2017, October). Tailoring architecture centric design method with rapid prototyping. In *Communication and Electronics Systems (ICCES), 2017 2nd International Conference on* (pp. 924-930). IEEE.

[10] Montealegre, R., Nelson, H. J., Knoop, C. I., & Applegate, L. M. (1996). BAE automated systems (A): Denver International Airport baggage-handling system. *Harvard Business School Teaching Case*, (9-396), 311.

[11] Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

[12] Poister, T. H., & Larson, T. D. (1988). The Revitalization of PennDOT. *Public Productivity Review*, 85-103.

[13] Devadiga, N. M. (2017, November). Software Engineering Education: Converging with the Startup Industry. In *Software Engineering Education and Training (CSEE&T), 2017 IEEE 30th Conference on* (pp. 192-196). IEEE.

[14] Ambler, S. (2002). *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons.