An Economic Analysis of Software Development Process based on Cost Models

Asst. Prof. Dr. Ediz Şaykol (Beykent University, Turkey)

Abstract

Software development process generally includes requirement analysis, design, implementation, and testing phases. The overall process is called Software Development Life Cycle (SDLC). Each of these steps was executed sequentially in earlier times, i.e. the output of a step is used as the input to the following step. This sequential execution is called waterfall process, and since the total duration of SDLC has been increasing, more dynamic models need to be employed in today's software engineering methodologies. V-Shaped model, Spiral model, Incremental or iterative software development model, are some examples of these methodologies. On the other hand, due to the increase in the total number of projects for a company and due to the product variability, reusability aspect has entered into the domain. This aspect gained importance in the recent years, leading to the execution of framework-based models and software product line engineering process. In this study, the above methodologies are analyzed from an economic perspective with respect to their cost models. Reusability will require upfront investment, but the gain will be higher as the number of common software items increases, which are determined in commonality/variability analysis phase. Improvements in SDLC might also require organizational changes to adapt new methodologies. These considerations are discussed along with the cost model analysis, and a cost-evaluation criterion is provided in the paper.

JEL code: L86

1 Introduction

Software Development Process, also called Software Development Life Cycle (SDLC), is the set of tasks including requirement analysis, design, implementation and testing that should be carried out throughout the process. Some scientists prefer using life-cycle term since they think this term is more general than development process, and hence many software development processes can cope with one SDLC model. For example, spiral life cycle model can be implemented using different development processes. However, other than the decision of which term we use, there exists various methodologies that can be used for the software development process management.

As the variability of the methodologies in SDLC increases, a need for standardization becomes inevitable. The ISO/IEC 12207 standard establishes a process of lifecycle for software, including processes and activities applied during the acquisition and configuration of the services of the system. Each Process has a set of related outcomes. This standard has the main objective of supplying a common structure so that the buyers, suppliers, developers, maintainers, operators, managers and technicians involved with the software development use a common language. This common language is established in the form of well defined processes. The structure of the standard was intended to be conceived in a flexible, modular way so as to be adaptable to the necessities of whoever uses it. The standard is based on two basic principles, namely modularity and responsibility. Modularity means processes with minimum coupling and maximum cohesion. Responsibility means to establish a responsibility for each process, facilitating the application of the standard in projects where many people can be legally involved (Levine, 2006). The set of processes, and the activities and tasks within processes can be adapted according to the nature of the software project. These processes are classified in three types: basic, support and organizational. The support and organizational processes must exist independently, and the basic processes are instantiated according to the situation.

During the time in which various firms produce software products or the projects of the firms include software development, SDLC methodologies experience a continuous improvement process based on the modularity and responsibility aspects suggested by the ISO/IEC 12207 standard. Not only the production time has been decreased during this improvement, but also the throughout cost has been decreased. These alternatives, or improved SDLC models, are called Single System Development Models, and will be referred in this way in this paper.

Another term has gained importance in the recent years along with the improvement in single system development models, namely *reusability*. As the variety and variability of the products increase, developers try to identify the unchanged/shared software components of the projects; hence these shared components can be developed and deployed a priori. This leads to the development of Framework-based Software Development Process.

As the gains of assembling shared/similar components together to satisfy reusability are clearly observed by the companies, the engineers try to work on broadening the scope of reusability. Not only the common components, but also the way of developing and utilizing the common and/or variable parts could be reused. The process of assembling such a broader reusability paradigm is called Software Product Line Engineering (SPLE) (Clements and Northrop, 2002), and it is a trendy topic in software product development. The key motivation for adopting a product line engineering process is to develop products more efficiently, get them to the market faster to stay competitive and produce with higher quality (Tekinerdogan, et. al, 2010a). SPLE suggests organizational improvements for the companies to improve future gains; however, the upfront investment is high due to the complexity of initial analysis and settlement.

In this paper, SDLC models are briefly explained in Section 2 in three categories: Single System Development Models, Framework-based Software Development Process, and Software Product Line Engineering. These methodologies are analyzed from an economic perspective with respect to their cost models since reusability will require upfront investment, but the gain will be higher as the number of common software items increases, which are determined in commonality/variability analysis phase (Ali, et. al, 2009). Improvements in SDLC might also require organizational changes to adapt new methodologies. These considerations are discussed along with the cost model analysis in Section 3. Finally, Section 4 concludes the paper.

2 Software Development Life Cycle Models

Each of the steps of Software Development Life Cycle (SDLC) was executed sequentially in earlier times, which means that the output of a prior step is used as the input to the following step. This sequential execution is called waterfall process, and since the total duration of SDLC has been increasing, more dynamic models need to be employed in today's software engineering methodologies. Below, some examples of single system SDLC models and methodologies including Waterfall model, V-shaped model, Spiral model, Incremental or iterative software development model are to be explained briefly. Along with the improvement based on modularity and reusability aspects, Framework-based software development process and software product line engineering paradigm are evolved in the recent years, and will be briefly discussed in sequel to provide a broader sense in software development process before formulating cost analysis (Tekinerdogan, et. al, 2010b).

2.1 Single System Development Models

2.1.1 Waterfall Model

The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance. The first formal description of the waterfall model is credited to Royce, though Royce did not use the term "waterfall" in this article. Royce presented this model as an example of a flawed, non-working model (Royce, 1970). In 1983, the paper (Benington, 1983) Benington pointed out that the process was not in fact performed in strict top-down, but depended on a prototype. Figure 1 illustrates the flow of steps as a waterfall in so called waterfall model.

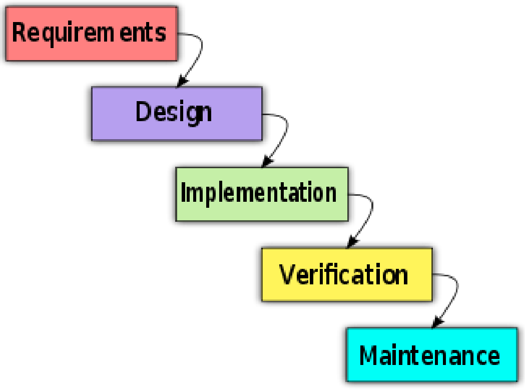

***Figure 1**. The steps of software development process based on waterfall model. Source: (Wikipedia, 2010a).*

2.1.2 V-Shaped Model

The V-Shaped model represents a software development process (also applicable to hardware development) which may be considered an extension of the waterfall model (FHWA, 2005). Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Shaped model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-to-grain abstraction uppermost), respectively.

Other than being easy-to-use, the advantages of V-shaped model has higher chance of success over the waterfall model due to the development of test plans early on during the life cycle. The V-shaped model has specific deliverables at each phase, and it works well when the requirements are easily understood, which is not frequent in real life. Like the waterfall model, V-shaped model is not very inflexible; but has little flexibility and adjusting scope is difficult and expensive. Figure 2 shows the steps of software development organized like the letter V, hence called the V-shaped model.

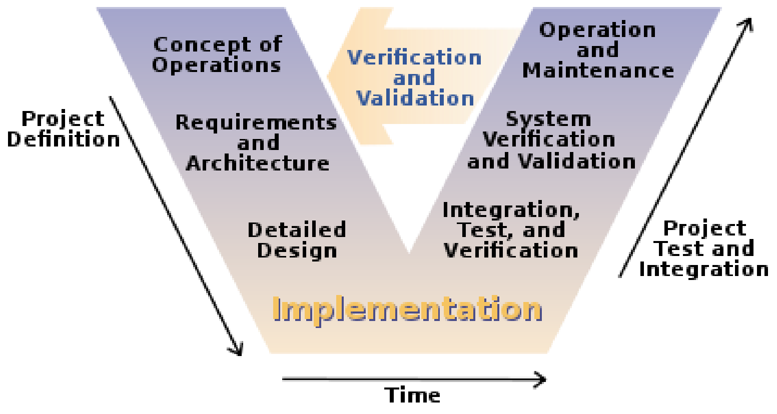
Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Figure 2. *The steps of software development based on V-shaped model. Source: (FHWA, 2005).*

2.1.3 Spiral Model

The spiral model is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts (Boehm, 1986). Also known as the spiral lifecycle model (or spiral development), it is a systems development method (SDM) used in information technology (IT). The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations. Figure 3 gives pictorial explanation of the software development process based on spiral model.

Main advantage is that the estimates (i.e. budget, schedule, etc.) become more realistic as work progresses, because important issues are discovered earlier. However, there is high cost and time to reach the final product; special skills are required to evaluate the risks and assumptions apriori.

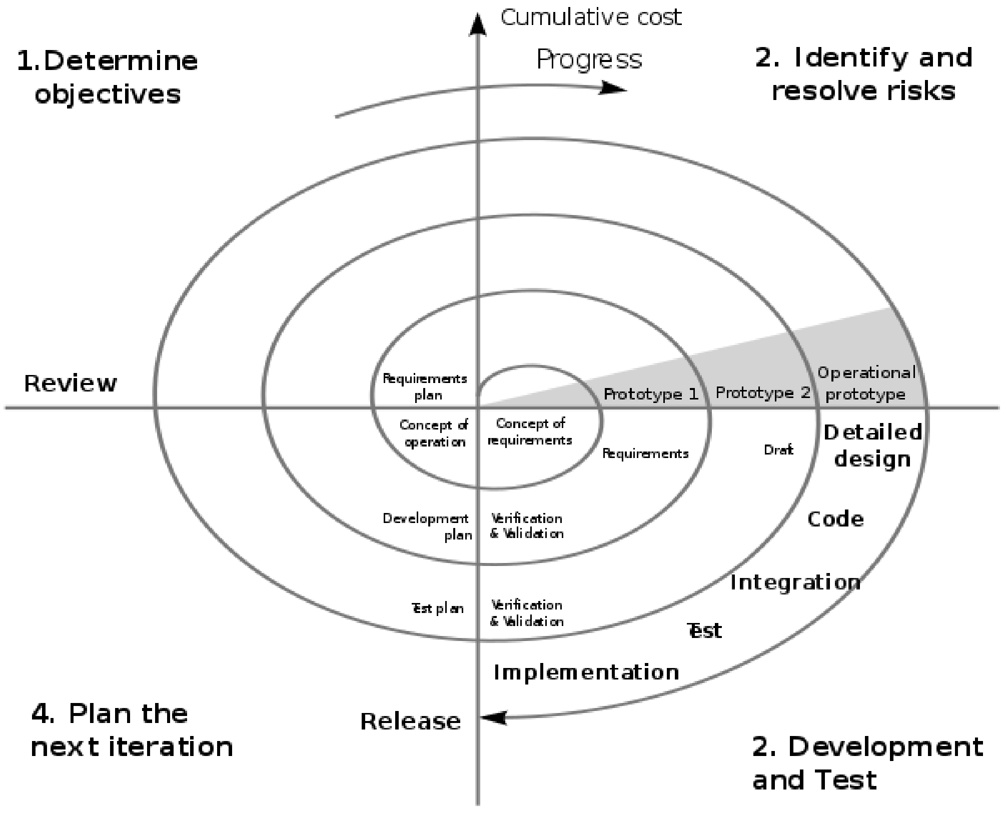

Figure 3. *The steps of software development based on Spiral model. Source: (Boehm, 1986).*

2.1.4 Iterative and Incremental Model

The Iterative and Incremental model is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system (Larman and Basili, 2003).

The procedure itself consists of the initialization step, the iteration step, and the Project Control List. The initialization step creates a base version of the system. The goal for this initial implementation is to create a product to which the user can react. It should offer a sampling of the key aspects of the problem and provide a solution that is simple enough to understand and implement easily. To guide the iteration process, a project control list is created that contains a record of all tasks that need to be performed. It includes such items as new features to be implemented and areas of redesign of the existing solution. The control list is constantly being revised as a result of the analysis phase. Figure 4 shows the iterative and incremental steps of software development process.

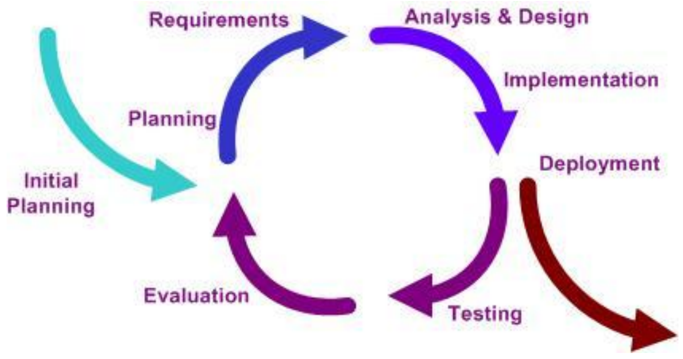

***Figure 4**. The steps of software developmentprocess based on Iterative ans Incremental model. Source: (Wikipedia, 2010b).*

The iteration involves the redesign and implementation of a task from the project control list, and the analysis of the current version of the system. The goal for the design and implementation of any iteration is to be simple, straightforward, and modular, supporting redesign at that stage or as a task added to the project control list. The level of design detail is not dictated by the interactive approach. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal Software Design Document may be used. The analysis of a single iteration is based upon user feedback, and the program analysis facilities available. It involves analysis of the structure, modularity, usability, reliability, efficiency, & achievement of goals. The project control list is modified with respect to the analysis based on previous steps.

2.2 Framework-based Software Development Process

A software framework is an abstraction in which software providing generic functionality can be selectively changed by user code, thus providing application specific software. A software framework is a universal, reusable software platform used to develop applications, products and solutions. Software Frameworks include support programs, compilers, code libraries, an application programming interface (API) and tool sets that bring together all the different components to enable development of a project or solution.

Frameworks contain key distinguishing features that separate them from normal software development reusable codes and/or software libraries. The main ones are inversion of control, extensibility. The framework code, itself, is not allowed to be modified by the software developers who are using the framework for developing another software product. They can extend the framework, or tailor the usage of it via configuration files, but not allowed to modify framework code. These principles are alleviated by “reusability” aspect of software products, in general, where a company might identify unchanged/shared components of its products, and hence might reduce the development cost by building up a framework for itself for these shared parts. In other words, framework-based software development process is like bringing the pieces of a puzzle (product) together while tracing software development tasks. Frameworks are also important for software companies because they can increase their impact in the market by building frameworks regarding their area of expertise. This intial framework building phase might require additional upfront investment, however; the future gains will surely be higher as long as the framework building steps are followed correctly.

One example on the framework-based development process is the Havelsan case (Saykol, et.al, 2006). Havelsan is a Turkish software and system development company having business presence in defense and IT sectors. Primarily for the projects in the defense domain, a decision was made to develop a software framework to be used in its naval and air projects, which was called Command and Control Software Framework (CCSF).

The shared components among naval or air software products are put together in the Core part of CCSF, which the developers only extend this framework-code for the development of the actual project (Karaca, et. al, 2010). A new team is established for the development and maintenance of the CCSF, and new software interfaces are defined. This required additional organization and technical cost to the company, but the impact of Havelsan is increased in local market based on this area of expertise. Since the future gains were expected as higher than this additional investment, the company decided to invest further to this framework (Tekinerdogan, et. al, 2010b).

2.3 Software Product Line Engineering

The process of assembling a broader reusability paradigm including the analysis of common and/or variable components of the projects that a company produces along with how should they be analyzed and developed is called Software Product Line Engineering (Clements and Northrop, 2002). Starting an SPLE from scratch is not preferred. Instead, a better way is first developing a set of products/projects, and then transitioning to a framework based on area of expertise, and then adopting a software product line.

The benefits for adopting a product line approach has been analyzed and discussed before by several authors (Buhne, et. al, 2003) (Pohl, et. al, 2005). The key motivation for adopting a product line engineering process is to develop products more efficiently, get them to the market faster to stay competitive and produce with higher quality. In alignment with these goals different software product line engineering processes have been proposed (Clements and Northrop, 2002). Unfortunately, a transition to software product line engineering is not easy. In general it requires large upfront investments and as such forms a serious risk if the desired return-on investment is not achieved. Furthermore, there is not a single fixed strategy for adopting a PLE. This is because each organization has its own specific context and usually decides to adopt a product line approach with specific business goals in mind (McGregor, et. al, 2002) (Pohl, et. al, 2005).

The obstacles of the framework-based approach were clear with the real practical experiences but a shift to a more mature reuse approach, the product line engineering approach, would require even more additional investments while it could lead to unforeseen risks and costs. As such, if the shift to a product line engineering approach would not be well managed, it could even decrease the benefits that were gained with the conventional framework-based approach. On the other hand, it was necessary to improve the reuse level of the company and achieve a more systematic reuse approach from the future perspective with the expected return on investments.

As argued in (Tekinerdogan, et. al, 2010b), three different types of upfront investment could be possible: upfront investment for moving from a single system development to a product line engineering approach (InvPLE), upfront investment for moving from a single system development to a framework-based approach (InvFW), and upfront investment for moving from a framework-based approach to a product line engineering approach (InvFW-PLE). From the same perspective, there are three different types of ROI: ROI for framework-based approach with respect to singles system development (ROIFW), ROI for product line engineering approach with respect to single system development (InvPLE), and ROI for product line engineering approach with respect to framework-based approach (ROIFW-PLE) (see Figure 5).

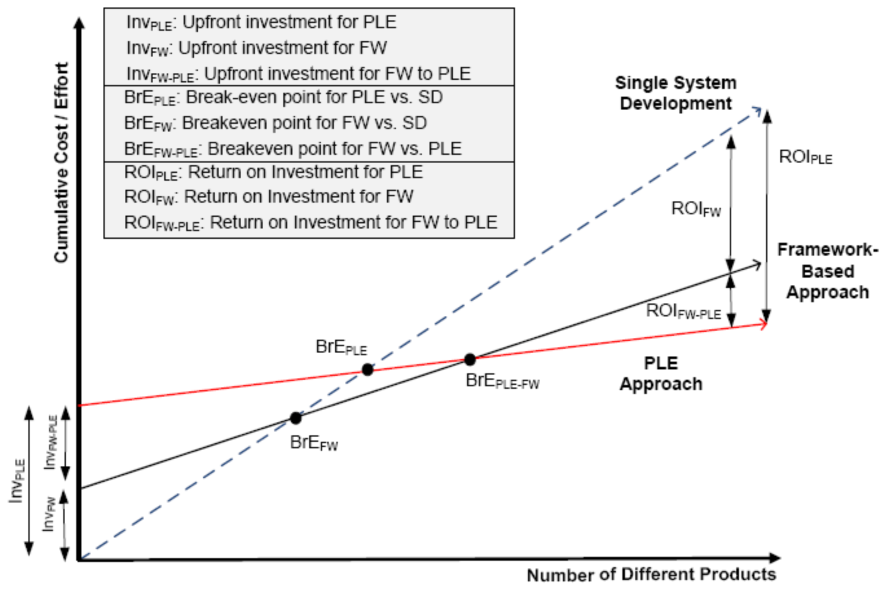

Figure 5. *Three strategies for evaluating the transition to Framework-based model and SPLE paradigm, revisited from (Tekinerdogan, et. al, 2010b).*

3 Analysis Based on Cost Models

In this paper, SDLC models are categorized in three categories: Single System Development Models, Framework-based Software Development Process, and Software Product Line Engineering. These methodologies are analyzed from an economic perspective with respect to their cost models since reusability will require upfront investment, but the gain will be higher as the number of common software items increases, which are determined in commonality/variability analysis phase.

To provide a fair analysis and evaluation, the cost of producing *n* products is considered for each of the three categories, since the latter two has future gains with an initial investment. For the single system development case, this type of economic analysis is not frequent in the literature. However, (Tekinerdogan, et. al, 2010b) and (Ali, et. al, 2009) could be given as examples of economic analysis of software product lines based on cost models. Here, we follow the analysis based on (Tekinerdogan, et. al, 2010b) and extend it for each of the three SDLC categories.

3.1 Single System Development Cost Analysis

In single system development, cost of producing n products can be estimated as n times cost of producing single item, if it is ssumed that cost of developing every item is almost the same.

For the cost of Waterfall model C_W, the total cost is the sum of costs of each step since there is linear execution of the steps as shown in Figure 1.

$$C_{\mathrm{W}} = \sum_{i=1}^{n}(CReq + CDes + CImp + CVer + CMai)$$

The most time consuming part of the Waterfall model is hidden in the above equation; such that there is a context switching cost in between every two step. Having finished Requirements step for instance, before starting Design step, both the development team and the organization need some adaptation time.

For the cost of V-shaped model C_V, the total cost can be computed similar to the C_W; however, the hidden context-switching time is almost zero in this case, since the steps are overlapped as shown in Figure 2.

For the cost of Spiral model C_S, the total cost has an additional term $C_{R\&E}$ for the cost of risks and evaluation other than the total cost of producing *n* products. Even though we an an additional term to C_W, to obtain C_S, C_S is lower than C_W, since the cost of each step (e.g., $CReq, CDes$) is very much reduced due to the prototyping that is employed at each loop as shown in Figure 3. For a sufficiently large project, it is straightforward that C_{Des} in the Waterfall model is strictly higher than C_{Des} in a prototype.

For the cost of Iterative and Incremental model, the total cost is very similar to than of Spiral model; however, $C_{R\&E}$ term is replaced with C_{Init}, and $C_{R\&E} < C_{Init}$ since at each iteration the developers embed what they have learned to the next iteration as shown in Figure 4.

3.2 Framework-based Development Cost Analysis

The cost equation for the Framework-based software development can be estimated as follows:

$$C_F = C_{\text{F-}Org} + C_{\text{F-Core}} + \sum_{i=1}^{n}(\text{CUnique i} + \text{CReuse i}),$$

where $C_{F\text{-}org}$ is the cost for adapting the organization for framework-based software development, $C_{F\text{-}Core}$ is the cost of developing a framework, *CUnique* is the cost to develop unique parts of the application that are not reused from framework, and *CReuse* is the cost of using the framework to develop unique applications.

As stated in (Tekinerdogan, et. al, 2010b), an interesting observation is the fact that, C_F is strongly dependent on the decision that the framework is either to be used within the company itself or to be sold to other companies. If the main purpose of framework development is to obtain inside-company reuse, the cost model applies as is. However, if the framework is to be sold to other companies, the costs *CUnique* and *CReuse* reduce to 0, and $C_{F\text{-}Org}$ becomes very low.

3.3 Software Product Line Engineering Cost Analysis

Instead of initiating an SPLE approach from scratch, a better way is first developing a set of products/projects, and then transitioning to a framework based on area of expertise, and then adopting a software product line. The cost equation for transitioning from framework-based development to software product line engineering can be expressed as follows:

$$C = Corg{-}transition + Cfrw{-}to{-}cab + \sum_{i=1}^{n}(\text{CUnique i} + \text{CReuse i})$$

Corg-transition is the cost of organizational transition to a software product line organization from framework-based organization. *Corg-transition* is assumed to be higher than $C_{F\text{-}Org}$ since the scope of organizational tasks is broader here.

Cfrw-to-cab is the cost of an organization to convert the framework into a set of reusable assets as described in product line architecture. Since framework is similar to the reusable assets concept in software product line engineering, *Cfrwk-to-cab* is assumed to be significantly less than the cost of developing reusable core assets from scratch.

To elaborate on this analysis, Figure 6 is given to show the Havelsan's approach for transitioning to software product line engineering (Tekinerdogan, et. al, 2010b). In the figure, the line corresponding to Product Line Engineering (PLE) Approach is a theoretical basis for evaluating the total cost for various software development processes. The total cost increases proportionally in case of single system development models. For the framework-based approaches, there is an upfront investment in the beginning, and the rate of increase is lower than single system development, but higher than product line engineering. For a typical transition, which is in fact the suggested way of transitioning to PLE, the company starts with single system development up to some decision point d1. After that decision point, the company invests on framework-based development until another decision point d2 where they invest more to adopt PLE approach.

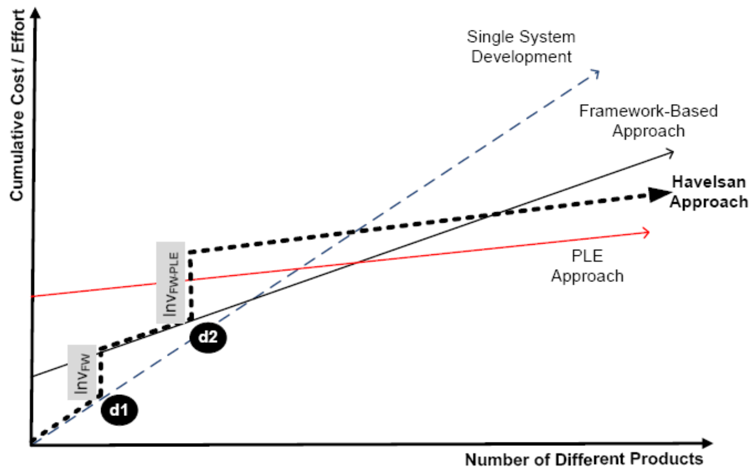

Figure 6. *The cost analysis of transitioning to a reusable software development model. The PLE Approach line is a theoretical cost baseline, and hence it can be seen that investing on reusable software development model eventually yields in higher ROI, revisited from (Tekinerdogan, et. al, 2010b).*

4 Conclusion

We provided a categorization of SDLC models into three: Single System Development Models, Framework-based Software Development Process, and Software Product Line Engineering. Brief explanations on Waterfall model, V-shaped model, Spiral model, Iterative and incremental model are provided as single system development models. To increase modularity and to achieve a reasonable degree of reusability, framework-based development models are discussed. To a higher degree of reusability, and also in a broader sense, trendy topic of Software Product Line Engineering is briefly mentioned.

The mentioned SDLC categories and approaches are analyzed in terms of their cost models. This analysis might be helpful for software companies whether they are to plan an adaptation to a better reusability to improve their future gains. The Havelsan case is briefly mentioned in the paper as an example to express that instead of initiating an SPLE approach from scratch, a better way is first developing a set of products/projects, and then transitioning to a framework based on area of expertise, and then adopting a software product line. This argument is also shown via cost models. To provide a fair analysis and evaluation, the cost of producing n products is considered for each of the three SDLC categories, since the latter two has future gains with different initial investments due to the scope of initialization.

References

- Ali, M.S., Babar , M.A., Schmid, K. (2009), “A Comparative Survey of Economic Models for Software Product Lines”, *in Proceedings of 35th Euromicro Conference on Software Engineering and Advanced Applications*, Patras, Greece.
- Benington, H.D., (1983), "Production of Large Computer Programs". *IEEE Annals of the History of Computing (IEEE Educational Activities Department)* 5 (4): 350–361.
- Boehm B., (1986), "A Spiral Model of Software Development and Enhancement", *ACM Software Engineering Notes*, 11(4):14-24.
- Buhne, S., Chastek, G., Kakola, T., Knauber, P., Northrop, L., Thiel, S., (2003), “Exploring the context of product line adoption”, in *Proceedings of the 5th International workshop on Product Family Engineering*, LNCS, van der Linden F. (ed.), Siena, Italy.
- Clements, P. and Northrop, L. (2002), **Software Product Lines: Practices and Patterns**. Boston, MA:Addison-Wesley.

- FHWA (2005), Clarus Concept of Operations. Publication No. FHWA-JPO-05-072, Federal Highway Administration (FHWA).
- Karaca, H.N., Yüksel M., Tüzün E., Kılınç, I., Baykal B., (2010) "A Data-Centric Framework for Network Enabled C4ISR Software Systems", *Lectures Notes in Electrical Engineering ISCIS 2010,* Springer Belin, Volume 62, 978-90-481-9793-4, pp 173-176.
- Larman, C. and Basili, V.R., (2003). "Iterative and Incremental Development: A Brief History", *IEEE Computer*, 36 (6): 47–56.
- Levine, M.H. (2006), "Analyzing the Deliverables Produced in the Software Development Life Cycle". Audit Serve, Inc., accessed at 16 May 2012.
- McGregor, J.D., Jarrad, S., Northrop, L.M., Pohl, K., (2002), "Initiating Software Product Lines", *IEEE Software*, vol. 19, no. 4, July 2002, pp.24–27.
- Pohl, K., Böckle, G., van der Linden, F. (2005), **Software Product Line Engineering – Foundations, Principles, and Techniques**, Springer.
- Royce, W. (1970), "Managing the Development of Large Software Systems", In *Proceedings of IEEE WESCON* 26: 1–9.
- Saykol, E., Saran, D., Kılınc, I., Yazgec, M., Aydın, M., Yıldız, O. (2006), "Komuta-Kontrol Sistemleri için Ulusal Yazılım Mimarisi Geliştirme Projesi", (in Turkish) 1. *Ulusal Yazılım Mimarisi Konferansı, UYMK 2006*, Yıldız University, Istanbul, Turkey.
- Tekinerdogan, B., Tuzun, E., Saykol, E., (2010a), "Multidimensional Classification Approach for Defining Product Line Engineering Transition Strategies", in *Software Product Lines: Going Beyond, Lecture Notes in Computer Science (LNCS) Proc. of the 14th Software Product Line Engineering Conference (SPLC 2010),* Vol. 6287, pp. 461-465, Edited by J. Bosch and J. Lee, Jeju Island, South Korea.
- Tekinerdogan, B., Tuzun, E., Saykol, E., (2010b), "Exploring the Business Case for Transitioning from a Framework-based Approach to a SPLE Approach", *Proc. of the 14th Software Product Line Engineering Conference (SPLC 2010)*, Industry Track, Jeju Island, South Korea.
- Wikipedia, (2010a), Waterfall model, http://en.wikipedia.org/wiki/Waterfall_model
- Wikipedia, (2010b), Iterative and Incremental Model, http://en.wikipedia.org/wiki/Iterative_and_incremental_development