

# Application Development Process -- A Pipeline Framework

Chaya Yerrapragada  
Department of Computer Sciences  
University of North Texas  
Denton, Texas-76203  
*chaya@geninet.com*  
Prof. Paul Fisher  
*fisher@compsci.com*

## Abstract

*Application development process involves a set of activities to produce an application. Multiple dependencies among multiple activities can effect the development time of the overall application. These activities can be represented as process phases such as analysis, design, implementation, testing and so on. While some development approaches treat each phase to start after completion of the previous phase; others attempt to start a phase before completion of the previous phase. However, there is no clear methodology to follow to start the phases of development at the earliest possible time. We propose a framework for the application development process, where communication, dependency, and relative time between phases are important factors. The methodology proposed to start the activities of application development transforms the development process into a pipeline structure.*

## 1: Introduction

The fundamental activities of developing an application are specifying the problem, designing, implementing, validating, and maintaining the product as a whole. There are many challenges in application development. For example, it is important to understand and satisfy the user requirements. Another such challenge is to start and complete the development activities at the earliest possible time. In this paper, we propose a framework, for developing applications, that focuses on reducing the development time. Section 1.1 briefly discusses the related work in application development

process. Section 2 describes our framework for development. Section 3 presents conclusions and possible future work areas.

### 1.1: Application Development

Detailed development models are still the topic of research but there are, in practice, a number of different general models or paradigms of application development. The emergence of the Waterfall process model [1] helped tackle the growing complexity of development projects. The model assumes that the requirements are known, defined, and will not change. Various development models have been proposed to deal with customer feedback on the product to assure that it satisfies the requirements. The main idea in these models is to have an evolutionary approach where an initial implementation is developed for the customer to use and provide feedback. The rapid throwaway prototyping was made popular by Gooa and Scott [2] and is used widely in the industry, especially in application development. Quick and dirty prototypes are built, evaluated by the customers, and thrown away until a satisfactory prototype is reached, at which point the full-scale development begins. The incremental development model [3] aims at developing only small parts of the system at a time that can be evaluated by the user to provide feedback to the development team. The spiral model [4] formed the generic process model that explicitly recognized risk in development. The object-oriented approach is based on the concept of objects and classes. Branson and Herness [5] proposed an eight-step object-oriented development process in 1992. Cyclic development process [6] consists

of three sets of incremental life cycles. Each set consists of a set of application development phases. The incremental nature of the process allows for validation of the development work. Computer-supported cooperative work (CSCW) is an environment where people work together, using computer technology, to achieve a common goal. This environment is suitable and essential in developing applications. CSCW includes use of email, hypertext that includes awareness of the activities of other users, videoconferencing, chat systems, and real-time shared applications, such as collaborative writing or drawing. People may be working together at the same time (such as video conferencing) or may be coordinating their work across longer periods of time (such as e-mail).

The current application development models are well defined and understood. The models use different techniques to deal with satisfying user requirements and managing risks. Most commonly used are the prototyping and incremental development techniques. All the current development models have advantages and disadvantages. However, the current models do not provide a structure that helps in reducing the development time. Additionally, they do not address the development interactions and dependencies during the development process. Our framework focuses on providing a structure to the development process that addresses these aspects. The structure provides the following:

- A representation and methodology where maximum concurrency is reflected. This translates into reducing the development time.
- A pattern in the development process that is flexible and repeatable at every phase of the development. This translates into having a unified way of development.

## 2: Application Development Process -- A Pipeline Framework

A framework is a reusable design or structure of all or part of a system that is represented by a set of abstract classes and the way their instances interact [7]. A framework for application development is a structure of the development process represented by a set of abstract descriptions of the steps and how they interact. The abstract descriptions are the multiple phases such as analysis, design, implementation, and testing. Each phase consists of multiple activities. For example, in multimedia application development, the work in the analysis phase is divided into the analysis of different components of multimedia. Further, the analysis of any given media

would be made up of multiple tasks. Each hierarchical level namely phases, activities, and tasks could be considered as entities at their respective hierarchical levels. This is the built-in pattern provided by our framework [8]. The pattern repeats at each level. Figure 1 shows the basic units of our framework.

Figure 1(a) shows an example of two entities  $X_1$  and  $Y_1$  progressing in time. Entities  $X_1$  and  $Y_1$  are at level 1 ( $L_1$ ). Figure 1(b) shows entity  $X_1$  is composed of two sub-entities  $X_{1,1}$  and  $X_{1,2}$  at level 2 and the interactions and relationships between the sub-entities. Similarly, entity  $Y_1$  is composed of two sub-entities  $Y_{1,1}$  and  $Y_{1,2}$  at level 2 and the interactions and relationships between the sub-entities. The sub-entities of  $X_1$  and  $Y_1$  are progressing in time. The entities may have relationships and dependencies between them. There are three important aspects to our framework: communication, dependencies, and relative time.

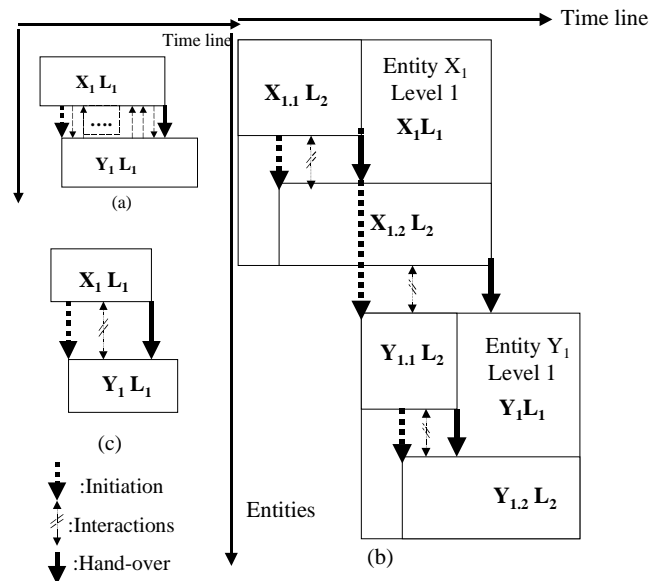


Figure 1: Basic units of the framework

### 2.1: Communication and relative time

Figure 1(a) shows the three aspects. In relative time, work in  $Y_1$  starts a little later than  $X_1$ . In other words, work in  $Y_1$  is dependent on some aspect of work in  $X_1$  to be completed before it can start. Completion of that aspect of work in  $X_1$  provides the initiation to  $Y_1$  to start its work. The initiation is shown in Figure 1(a) using a dashed bold arrow from  $X_1$  to  $Y_1$ . Entities  $X_1$  and  $Y_1$  communicate or may have many interactions during the course of functions or work performed in the respective entities. The communication or the interactions between

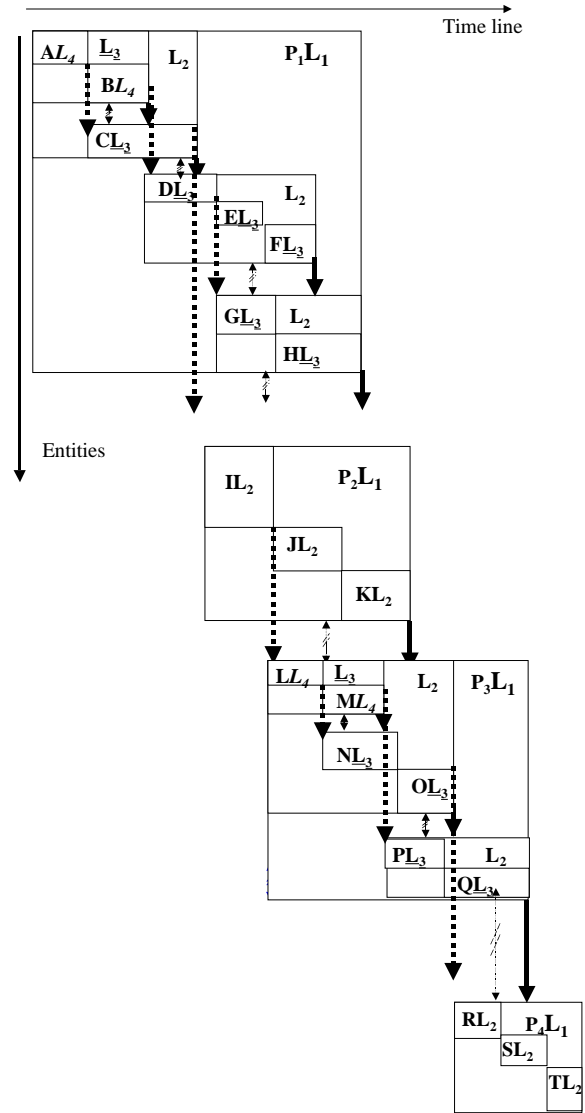
$X_1$  and  $Y_1$  are shown in 1(a) using dashed light arrows. A final hand-over of work performed in  $X_1$  is made from  $X_1$  to  $Y_1$ . Work in entity  $Y_1$  may continue further, in time, to completion. This basic unit of communication is carried over between entities, sub-entities, sub-sub-entities, and so on. Thus, we can have three types of communication between any two entities: initiation, interactions, and hand-over. To simplify the representation of multiple interactions between  $X_1$  and  $Y_1$ , the interactions are depicted with a single variable (dashed light) arrow as shown in Figure 1(c). The initiation of work is depicted using a dashed bold arrow and the hand-over of work is shown using a solid bold arrow.

## 2.2: Dependencies

Figure 1(b) shows the composition of entities  $X_1$  and  $Y_1$  and the relationship between them. The figure shows the relationships to the depth of level 2. It is clear from the figure that entity  $Y_1$  is dependent on entity  $X_1$ . Entity  $X_1$  provides initiation to  $Y_1$  at which point entity  $Y_1$  is ready-to-start. Similarly, sub-entity  $X_{1,2}$  at level 2 is ready-to-start after receiving initiation from sub-entity  $X_{1,1}$  at level 2. Sub-entity  $Y_{1,2}$  at level 2 is ready-to-start after receiving initiation from  $Y_{1,1}$  at level 2. In order to reduce development time, we need to have the earliest possible initiations of entities, sub-entities, and so on, so that work can progress concurrently in different entities but with interactions between them. We use the term "ready-to-start" because even though  $Y_1$  receives a go-ahead signal from  $X_1$ , it may not really be able to start due to its own resource constraints. Alternately, the entity may factor in the dependency of an entity subsequent to it to decide to start.

The earliest possible initiations of entities is accomplished by dividing entities, in an iterative fashion, into smaller entities such that the first sub-entity of an entity at a level initiates the first sub-entity of the next (following in time) entity at the same level. For example, the first sub-entity of  $X_1$  at level 1 is  $X_{1,1}$ , which will initiate the first sub-entity of  $Y_1$  at level 1, which is  $Y_{1,1}$ . Similarly, the first sub-entity of  $X_{1,1}$  at level 2 will initiate the first sub-entity of  $X_{1,2}$  at level 2. In other words, the time required for  $Y_{1,1}$  to be ready-to-start, is the time required for  $X_{1,1}$  to complete. This is an important feature of the framework, where we have the same pattern between entities at their respective levels. We have a pattern between entity  $X_1$  and  $Y_1$  at level 1. The same pattern is followed by sub-entities  $X_{1,1}$  and  $Y_{1,1}$  at level 2 and so on. The same pattern within a pattern simplifies complex structures, allows for expansion with ease, and

facilitates automation. Figure 2 illustrates the pattern in the framework.



**Figure 2:** Pattern in the framework

The figure shows four entities  $P_1$ ,  $P_2$ ,  $P_3$ , and  $P_4$ . Each of these entities has been divided to their lowest levels. At the lowest level, we have the smallest practical invisible tasks into which an entity can be divided. These tasks cannot be subdivided further. The dependencies and communication between these entities is shown. The figure also clearly shows the methodology that we use in dividing the entities to provide the earliest initiation to the subsequent entities. In order to provide the earliest initiation, the entities are divided to lowest level, where a smallest piece of work causes the initiation to another smallest piece of work in the subsequent entity. At the

lowest level of modularization, the entities form a linear model, with just one type of communication (hand-over) between the entities. For example, entities A and B in Figure 2 are linear. At the lowest level, the pattern looks like multiple pipeline processes but with communication between the pipelines. Figure 3 illustrates the pipeline framework for the application development process. In practice however, due to variety of reasons (such as reducing communication overhead, interfaces, and resources), the number of levels that an entity is divided is a subjective matter and depends on individual development environment.

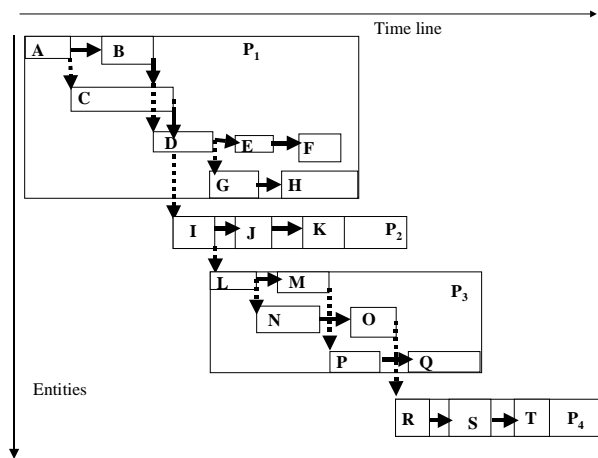


Figure 3: A pipeline framework

### 2.3: Application Development Process

We adapt the basic units of the framework to the application development process as shown in Figure 4. Application development is divided into different phases and these phases are divided again into smaller and more manageable activities and further into tasks. The resultant pieces of work can be spread out over a given amount of time. In a phased approach, it is clear that some phases or activities or tasks cannot start before others. For example, you cannot start implementing or coding before you have started design or designing before you have started to analyze the requirements or analyzing requirements before you have started to gather requirements from the user. However, there is a considerable overlap and merging of activities across the phases producing results in less time. Additionally, development process is iterative. Due to changes in requirements the tasks may be repeated. The framework provides a structure to represent these dependencies and helps in reflecting maximum

concurrency.

There are interactions and handoffs (communication) between each phase of development. The interactions and handoffs result in respective resultant interactions and handoffs to the next phase. Over time, in a corporation, communication can be self-refined by gathering intelligence during the development process. Communication is important for any development work. It can take different forms such as initiation, general interaction, or hand-over between phases. General interaction is the (feedback loop) continuous feedback, modification, and clarification that is so important during the development process. Development models that freeze phase output early cause problems to the success of an application. But there is a danger; too much feedback and changes can also cause problems draining resources (time and money). In our framework, any new change requested by the user flows through the development process that can evaluate the change, before it is incorporated.

The framework also shows various aspects that are considered during the development process [9]. They are shown around the circle encompassing the authoring framework. For example, the development effort requires data transmission, synchronization, and group management services so that named groups of people interact and collaborate on the development activity. Additionally, we have shown project co-ordination as an entity that is always active, interacting with the phases of development on a continuous basis. It keeps track of progress, communicates any change in responsibilities, constraints, ordering of tasks or expectations to the development team and to the user.

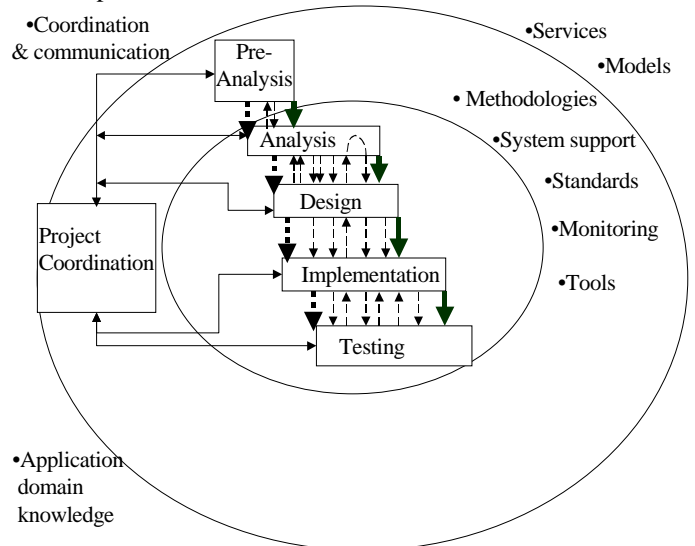


Figure 4: Application development framework

### 3: Conclusions and Future Work

We have discussed the current practice of application development models. We believe that the development model should provide a representation and methodology for reflecting maximum concurrency. We have proposed a general framework applicable to applications and system development. We have limited the discussion of the framework to general principles of modularization, and interaction & dependency between entities at different hierarchical levels.

To demonstrate the viability of this framework we are working on adapting it to all phases of multimedia application development. This framework is not limited to application development. As future work, we intend to study the framework in other disciplines. Other issues to work on are the dynamics of development with resource constraints and multiple projects, application of queuing principles to the framework, details of interaction and dependencies between entities in a framework, modularization of entities in application development, and types of relationship between entities.

### References

1. Royce, W. W., "Managing the development of large software systems: concepts and techniques," Proceedings IEEE WESTCON, Los Angeles, 1970, pp. 1-9 [9].
2. Gomaa, H., and D. Scott, "Prototyping as a tool in the specification of user requirements," *Proceedings 5<sup>th</sup> IEEE International Conference on Software Engineering*, March 1981, pp. 333-342.
3. Mills, H. D., O'Neill, D., Linger, R. C., Dyer, M., and Quinnan, R.E., "The management of software engineering," *IBM Sys. J.*, 1980, 24 (2), 414-77 [144].
4. Boehm, B. W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1988, pp.61-72.
5. Branson, M. J., and E. N. Herness, "Process for building object oriented systems from essential and constrained system models: Overview," *Proceedings of the Fourth Worldwide MDQ Productivity and Process Tools Symposium: Volume 1 of 2*, Thornwood, N. Y.: IBM Corp., March 1992, pp. 577-598.
6. John C. Canessa, "Cyclic development process," *NTCC'96*, February 14-17.
7. Ralph E. Johnson, "Frameworks = (Components + Patterns)," *Communications of the ACM*, October 1997, Vol. 40, No. 10, pp. 39-51.
8. Chaya Y., K. Murthy, P. Fisher, "A unified framework for multimedia applications development," *Inform/CORS Montreal Meeting* 1998.
9. C. Yerrapragada and P. S. Fisher, "Software Development Process Model For Multimedia Applications," *DSP & CAES Conference*, Nicosia, CYPRUS, July 14-16, 1993.