OBJECT-ORIENTED SOFTWARE DEVELOPMENT EFFORT PREDICTION USING DESIGN PATTERNS FROM OBJECT INTERACTION ANALYSIS

A Dissertation

by

OLUSEGUN ADEKILE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2008

Major Subject: Computer Science

OBJECT-ORIENTED SOFTWARE DEVELOPMENT EFFORT PREDICTION USING DESIGN PATTERNS FROM OBJECT INTERACTION ANALYSIS

A Dissertation

by

OLUSEGUN ADEKILE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	Dick B. Simmons
	William M. Lively
Committee Members,	Richard Furuta
	Ergun Akleman
Head of Department,	Valerie E. Taylor

December 2008

Major Subject: Computer Science

ABSTRACT

Object-Oriented Software Development Effort Prediction Using Design Patterns from Object Interaction Analysis. (December 2008)

Olusegun Adekile, B.S., University of Georgia

Co-Chairs of Advisory Committee: Dr. Dick B. Simmons
Dr. William M. Lively

Software project management is arguably the most important activity in modern software development projects. In the absence of realistic and objective management, the software development process cannot be managed in an effective way. Software development effort estimation is one of the most challenging and researched problems in project management. With the advent of object-oriented development, there have been studies to transpose some of the existing effort estimation methodologies to the new development paradigm. However, there is not in existence a holistic approach to estimation that allows for the refinement of an initial estimate produced in the requirements gathering phase through to the design phase. A SysML point methodology is proposed that is based on a common, structured and comprehensive modeling language (OMG SysML) that factors in the models that correspond to the primary phases of object-oriented development into producing an effort estimate. This dissertation presents a Function Point-like approach, named Pattern Point, which was conceived to estimate the size of object-oriented products using the design patterns found in object interaction modeling from the late OO analysis phase. In particular, two measures are

proposed (PP_1 and PP_2) that are theoretically validated showing that they satisfy well-known properties necessary for size measures.

An initial empirical validation is performed that is meant to assess the usefulness and effectiveness of the proposed measures in predicting the development effort of object-oriented systems. Moreover, a comparative analysis is carried out; taking into account several other size measures. The experimental results show that the Pattern Point measure can be effectively used during the OOA phase to predict the effort values with a high degree of confidence. The PP_2 metric yielded the best results with an aggregate PRED (0.25) = 0.874.

DEDICATION

To my family for their love, patience and encouragement.

ACKNOWLEDGEMENTS

This dissertation would not have been successful without the help and support of many people. I would like to particularly express my gratitude and deep appreciation to my advisor, Dr. Dick Simmons, for his guidance, inspiration, and the countless hours he spent on me during my graduate studies at Texas A&M University. His knowledge and experience have enriched both my academic and work experience. This research work would not have been possible without his guidance and support.

I would like to thank my research committee co-chair, Dr. William Lively for his guidance in the area of software engineering and for his interest in my research. I would also like to thank Dr. Akleman and Dr. Furuta for their support and guidance as members of my research committee. I would like to thank my wonderful parents, Dr. and Mrs. Adekile. I grow in appreciation daily of having been blessed with such caring and loving people as parents. Lastly, I would like to thank my lord and my savior Jesus Christ.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

"The process of controlling a software engineering project may well be the most talked about and least understood of all the project managers' functions." Lehman [40]

1.1 Motivation

Traditional software effort estimation techniques rely on analytic equations, statistical data fitting, expert judgment or some combination of the three. Although these are continually updated, they are still notoriously inaccurate. This is the case because no two software projects are the same and some of the assumptions made in the estimations are never realized in the actual course of the project, others are either unaccounted for or are inaccurately estimated. Furthermore, traditional methodologies were not designed for the object-oriented software development paradigm and are thus ill suited for OO projects [34], [35].

There are two bases that make the approach taken in this dissertation feasible and practical. The first is the increasing popularity and advancements made in the CASE (Computer Aided Software Engineering) tools. The CASE tools make a recording of the entire length of a software project at any stage of development readily available. Secondly, a key characteristic of the object-oriented paradigm is the continual realization and refinement of the same system artifacts/objects at each phase of development or within each development iteration (depending on the chosen project life cycle). These two factors make it possible to define a comprehensive model that can use data gathered

---

This dissertation follows the style of *IEEE Transactions on Software Engineering*.

unobtrusively from the CASE tools in the stages of development preceding implementation, to predict the effort required to further realize, refine and develop these and other system artifacts regardless of the level of realization or refinement of the existing artifacts.

1.2 SysML

This dissertation proposes an effort prediction model – the SysML Point Model - for object-oriented development systems that is based on a common, structured and comprehensive modeling language (OMG SysML), which can be built using the CASE tools from which data can be unobtrusively gathered and applied to prediction equations.

The Object Management Group Inc (OMG), established in 1989, is a not-for-profit, open membership, computer industry standards consortium that produces and maintains computer industry specifications for portable, reusable and interoperable enterprise applications in distributed, heterogeneous environments [98].

OMG's specifications include: CORBA (Common Object Request Broker Architecture); UML (Unified Modeling Language); CWM (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets. OMG member companies write, adopt, and maintain its specifications following an open and mature process. OMG's specifications implement the Model Driven Architecture (MDA), maximizing return on investment through a full-lifecycle approach to enterprise integration that covers multiple programming languages, operating systems, middleware and networking infrastructures, and software development environments. More information on the OMG is available at http://www.omg.org/.

OMG SysML [98] is a specification that defines a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities. SysML is intended to be supported by two evolving interoperability standards: the OMG XMI 2.1 (XML) model interchange standard for UML 2.1 modeling tools and the ISO 10303-233 data interchange standard for systems engineering tools. SysML reuses a subset of UML 2.1 and provides additional extensions needed to address the requirements in the UML for SE RFP.

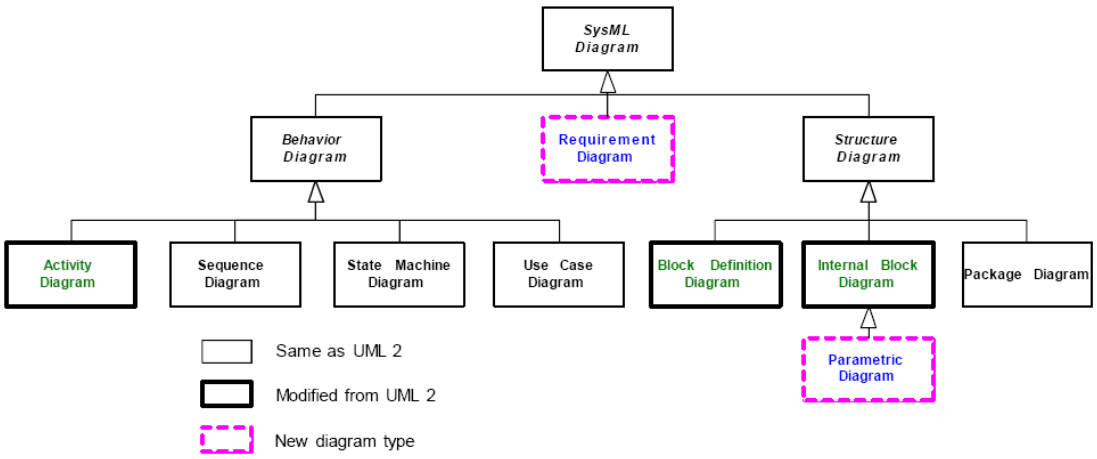


Figure 1: OMG SysML taxonomy

Following is the description of the OMG SysML taxonomy [98] as seen in Figure 1:

Structure Diagram

This defines the static and structural constructs in SysML.

- Block Definition Diagram

  The Block Definition Diagram in SysML defines features of a block and relationships between blocks such as associations, generalizations, and dependencies. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree. Blocks are modular units of a system description, which define a collection of features to describe a system or other elements of interest. These may include both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit. SysML blocks are based on UML classes as extended by UML composite structures.

- Internal Block Diagram

  The Internal Block Diagram in SysML captures the internal structure of a block in terms of properties and connectors between properties. A block can include properties to specify its values, parts, and references to other blocks. Ports are a special class of property used to specify allowable types of interactions between blocks. Constraint Properties are a special class of property used to constrain other properties of blocks. Various notations for properties are available to distinguish these specialized kinds of properties on an internal block diagram.

- Parametric Diagram

  A parametric diagram is defined as a restricted form of internal block diagram. A parametric diagram may contain constraint properties and their parameters, along with other properties from within the internal block context. All properties that appear, other than the constraints themselves, must either be bound directly to a constraint parameter, or contain a property that is bound to one (through any number of levels of containment).

- Package Diagram

  A package diagram depicts how a system is split up into logical groupings by showing the dependencies among these groupings. As a package is typically thought of as a directory, package diagrams provide a logical hierarchical decomposition of a system.

Behavior Diagram

- Sequence Diagram

  The Sequence Diagram is the most common of a classification of diagrams called Interaction Diagrams, others include Communications Diagram, Interaction Overview Diagram, and Timing Diagram. The sequence diagram describes the flow of control between actors and systems (blocks) or between parts of a system. This diagram represents the sending and receiving of messages between the interacting entities called lifelines, where time is represented along the vertical axis. The

sequence diagrams can represent highly complex interactions with special constructs to represent various types of control logic, reference interactions on other sequence diagrams, and decomposition of lifelines into their constituent parts.

- State Machine Diagram

  The State Machine package defines a set of concepts that can be used for modeling discrete behavior through finite state transition systems. The state machine represents behavior as the state history of an object in terms of its transitions and states. The activities that are invoked during the transition, entry, and exit of the states are specified along with the associated event and guard conditions. Activities that are invoked while in the state are specified as "do Activities," and can be either continuous or discrete. A composite state has nested states that can be sequential or concurrent.

- Use Case Diagram

  The use case diagram describes the usage of a system (subject) by its actors (environment) to achieve a goal that is realized by the subject providing a set of services to selected actors. The use case can also be viewed as functionality and/ or capabilities that are accomplished through the interaction between the subject and its actors. Use case diagrams include the use case and actors and the associated communications between them. Actors represent classifier roles that are external to the

system that may correspond to users, systems, and or other environmental entities. They may interact either directly or indirectly with the system. The actors are often specialized to represent taxonomy of user types or external systems.

- Activity Diagram

  Activity modeling emphasizes the inputs, outputs, sequences, and conditions for coordinating other behaviors. It provides a flexible link to blocks owning those behaviors.

Requirement Diagram

The requirements diagram can depict the requirements in graphical, tabular, or tree structure format. A requirement can also appear on other diagrams to show its relationship to other modeling elements. The requirements modeling constructs are intended to provide a bridge between traditional requirements management tools and the other SysML models. Several requirements relationships are specified that enable the modeler to relate requirements to other requirements as well as to other model elements. These include relationships for defining a requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements, and refining requirements.

The proposed SysML Point model is composed of four separate estimation models that correspond to the middle tier of the SysML taxonomy (Figure 1), and is designed to cover the primary phases in a typical object-oriented development effort such as the Unified Process [99].

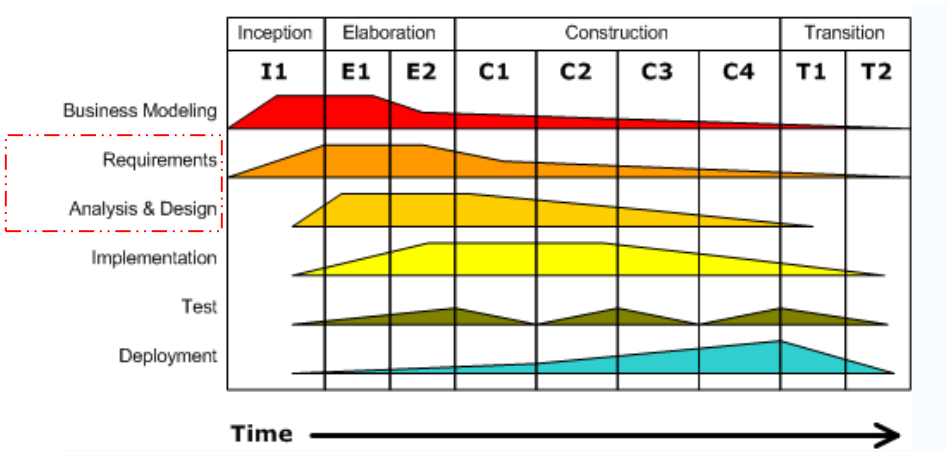


Figure 2: The unified process

That is, the requirements gathering, object-oriented analysis (behavioral artifacts) and the object-oriented design (structural artifacts) activities (Figure 2), with linkages between the three and a mechanism that allows for the substitution of refined artifacts as they occur in the estimation model. While in the requirements gathering phase, a SysML Point practitioner would apply the Function Point methodology [30] in producing a development effort estimate (Figure 3). This estimate would be refined in the early and late analysis stages using the Use Case Point [9] and Pattern Point methods respectively. Finally, prior to the commencement of the implementation phase, the Class Point method [29], which utilizes artifacts from the late design phase would be applied in generating a further refinement of the estimate.

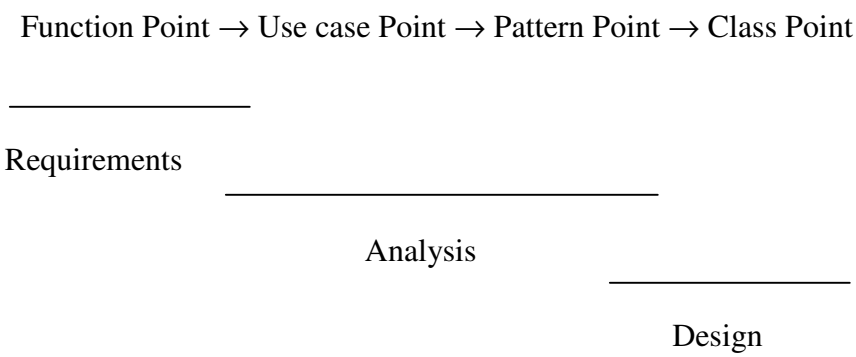


Figure 3: Object-oriented development stages and corresponding effort estimation models

1.3 Research Objectives

The focus of this dissertation is to define and validate the Pattern Points (PP) method of the SysML Point approach. The PP method utilizes artifacts from the late object-oriented analysis phase to produce an effort estimate. Object-oriented analysis (OOA) is concerned with the transformation of software engineering requirements and specifications into a system's object model, which is composed of a population of interacting objects (rather than the functional views or traditional data of systems) [108]. Some of the benefits of OOA include: “maintainability through simplified mapping to the real world, which provides for less analysis effort, less complexity in system design, and easier verification by the user; reusability of the analysis artifacts which saves time and costs; and depending on the analysis method and programming language, productivity gains through direct mapping to features of Object-Oriented Programming Languages” [108].

The Pattern Points (PP) model is an empirical parametric estimation method that uses object interactions and the class structure of object-oriented design patterns to predict development effort in the late analysis phase of an object-oriented project. It relies on a sizing of each of the 23 object-oriented design patterns as defined in the seminal book *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994 (Gamma et al) [3]. In software engineering, a design pattern is a common reusable solution to a frequently occurring problem in software design. It is a description or template for how to solve a problem and not a finished design that can be transformed directly into code. A design pattern can be used in many different situations. Typically, object-oriented design patterns display relationships and interactions between classes or objects without specifying the final application classes or objects that are involved.

The remaining effort prediction methodologies in the SysML Point model are already in existence. The Function Point method was introduced by Albrecht [30] to measure the size of a data-processing system from the end-user's point of view. It is based on the functional requirements of the system. The Use Case Point model, which is based on *use case* counts called use case points, is defined in Carol et al [9]. A use case is a description of a system's behavior as it responds to a request that originates from outside of that system. Use cases are refined into object interaction diagrams such as sequence diagrams, and analysis classes in the late analysis stage. Lastly, the Class Point method as defined by Costagliola et al [29] produces an estimate of the effort based on the design/structural artifacts.

Hypothesis. A reliable development effort estimate can be produced using an empirical parametric estimation model, which is based on the object-oriented design patterns that are found in analysis artifacts in an object-oriented development project.

1.4 Organization of the Dissertation

Following this introductory section, this dissertation is presented in six additional sections. Section 2 presents relevant background research. This includes literature on the subject of software project management and effort estimation. Design patterns and the Pattern Point model are described in Section 3. Section 4 describes the theoretical validation of the Pattern Point model. Section 5 explains the project experiment results used to empirically test the research model. Section 6 presents the conclusions and discusses future extension of this research.

2. LITERATURE SURVEY

2.1 Software Project Management

Software project management is a major endeavor that helps to realize a successful software project. It is a sub-discipline of project management in which software projects are planned, monitored and controlled. The determination of success for a software project is a software product that is delivered on time and within budget to a satisfied customer. In large complex systems, project management is the biggest challenge in the development process for managers. To help in this undertaking, various methodologies and techniques have been studied, and several CASE (Computer Aided Software Engineering) tools have been introduced to assist managers in solving the reoccurring problems. Presented in this section are some of the techniques used in software project management, in particular effort estimation.

Evaluating the status of a software project entails the collection, validation, analysis and presentation of software metrics and project data in a timely manner. Thus, the main functions of software project managers are planning, estimating, tracking, and decision-making. As long as the progression of the project matches the plan, the project is expected to succeed. Conversely, if there are some mismatches between the progress and plan, then a control process needs to be initiated to return the project activities back on track.

Software project management is “deciding what to do, how to do it and who does it, setting objectives, breaking work into tasks, establishing schedules and budgets, allocating resources, setting standards, and selecting future courses of action” [44].

Planning is central to software project management; it involves the identification of the activities, milestones, and deliverables produced by a project [39]. The project plan is a key deliverable in which the manager describes how the project will be developed, what resources will be required, and how those resources will be utilized. It is an evolving document that guides the software project manager and other staff members through the software development process. The Software Engineering Process Office (SEPO) of the United States Navy, describes the Software Development Plan (SDP) as the essential planning document for a software development project.

An important function of an SDP is to categorize the project development process and sub-processes that would be used in the construction of the software product. Boehm [45] uses the *WWWWWHH* principle as an organizing criterion in the planning process to identify the process model or models (such as waterfall, evolutionary, spiral, incremental, design-to-cost or –schedule, or a hybrid): who, what, where, when, why, how and how much, as follows:

Objectives - Why is the system being developed?

Milestones and Schedules - What will be done? When?

Responsibilities - Who is responsible for a function? Where are they organizationally located?

Approach - How will the job be done technically and managerially?

Resources - How much of each resources are needed?

The planning process must involve both the product that is to be developed and the accompanying processes that are needed to support the software product [47]. Once a

development process that fits both the product and people is identified, the activities can be broken down into tasks, which are then executed according to the selected process model [46].

The Software Engineering Institute's Capability Maturity Model (SEI CMM) [48]-[50] first described in the book "Managing the Software Process" by Watts Humphrey [54] provides a benchmark of software process maturity. It aids in the definition and understanding of an organization's processes and is widely employed in the industry to evaluate the maturity of an organization's software process. The SEI Maturity Questionnaire includes a scenario on software project planning that is used to assess the completeness of the planning framework. It is as follows:

1. Are estimates (e.g. size, cost, and schedule) documented for use in planning and tracking the software project?
2. Do the plans document the activities to be performed and the commitments made for the software project?
3. Do all affected groups and individuals agree as to their commitments related to the software project?
4. Does the project follow a written organizational policy for planning a software project?
5. Are adequate resources provided for planning the software project (e.g. funding and experienced individuals)?

6. Are measurements used to determine the status of the activities for planning the software project (e.g., completion of milestones for the project planning activities as compared to the plan)?
7. Does the project manager review the activities for planning the software project both a periodical and event-driven basis?

Within the Capability Maturity Model (CMM) Key Process Area (KPA) [49], software project planning is a Level 2. Passing the KPA is a major step toward reaching Level 2 (Repeatable). The KPA requires the development of a project Software Development Plan (SDP) and a written process for planning a software project. The CMM [49] defines 15 activities for the Project Planning KPA. The 15 steps in the CMM Planning KPA are as follows:

1. The software engineering group participates on the project proposal team.
2. Software project planning is initiated in the early stages of, and in parallel with, the overall project planning.
3. The software engineering group participates with other affected groups in the overall project planning throughout the project life.
4. Software project commitments made to individuals and groups external to the organization are reviewed with senior management according to a documented procedure.
5. A software life cycle with predefined stages of manageable size is identified or defined.

6. The project's software development plan is developed according to a documented procedure.
7. Software work products that are needed to established and maintain control of the software project are identified.
8. Estimates for the size of the software work products (or changes to the size of software work products) are derived according to a documented procedure.
9. Estimates for the software project's effort and cost are derived according to a documented procedure.
10. Estimates for the project's critical computer resources are derived according to a documented procedure.
11. The project's schedule is derived according to a documented procedure.
12. The software risks associated with the cost, resources, schedule and the technical aspects of the project are identified, assessed, and documented.
13. Plans for the project's software engineering facilities and support tools are prepared.
14. Software planning data are documented.
15. Measurements are made and used to determine the status of the software planning activities.

According to Humphrey [54], there are five basic important components of a software project plan:

1. **Goals and Objectives:** For a software project to be considered a success, the software product must be delivered on time and within budget to a satisfied

customer. The goals and objectives of the project are determined in the requirements negotiation phase. The initial statement of work must be clear, straightforward, and stable because it will be the statement from which the software development company will determine the product's functional goals.

2. **Work Breakdown Structure:** The WBS provides a hierarchical view for the whole project. After the requirements have been declared, an estimate of the product size and project effort is required. To produce an effective estimate, the project has to be broken down into its various work elements, which comprise the project WBS. The project structure and the selected software development process affect the WBS. After the project structure is formalized, the tasks for each unit of the project is defined and then apportioned to respective owners. The WBS was introduced into software project planning in the early 80's [51].
3. **Product Size and 17 Other Dominators:** This is perhaps the key portion of the planning process. The 17 project dominators are as follows: Development Schedule Constraints, Project Life Cycle process, Volume, Amount of Documentation, Programming Language, Complexity, Type of Application, Work Breakdown Structure, Management Quality, Lead Designer, Individual Developers, Personnel Turnover, Communications, Number of People, Software Reuse, Customer Interface Complexity, and Requirements Volatility Dominators are project attributes that cause effort (and productivity) to vary by an order of magnitude (10 to 1) [52]. A poor size and dominators estimate is the root of many problems in the software industry. Dominators may or may not appear as

variables in effort models. For example, we often assume that all projects are properly managed, even though they may not be. The result can be a failed project dominated by poor management. Dominators like management often do not have a 10:1 affect on reducing effort, but they definitely can have over a 10:1 affect on increasing effort [52]. Productivity is improved when managers reduce the effort to produce a product, as effort required to produce a product is inversely related to productivity. Dominators that affect effort prediction are everywhere in the project life cycle and are not independent of each other. Product size is useful for predicting effort. Two units are common for size measurement: lines of code and function points. A line of code is a fixed unit and easier to count, but is language-dependent. Function points are a more subjective and abstract unit, which is subject to bias [52].

4. **Resource Estimates:** There are always resource constraints that limit the amount of time that is spent on a project [52]. However, it is possible for a manager to estimate the resources that are required to design and implement the software product once an estimate of the size of code that is needed is available. Human resource is the most critical as it plays the most essential role in determining the cost of implementation. There is a myriad of methods and accompanying tools that are available for cost estimation and some of them are visited in the following section (2.1). However, a software company's historical performance plays the most important role when it comes to estimation of resources. The historical productivity rate can be applied to a new estimate to convert a size

estimate into a corresponding estimate of resources .If a cost model such as COCOMO or SLIM is used, its calibrations must match the software company's historical experience [52].

5. **Scheduling:** The resource estimates (described above) determine the scheduling of a project. Two scenarios surface in scheduling, depending on which side sets the release date for the product [53]. Usually, the project manager will make the decision on the release date based on an appropriate starting date and schedule. However, market forces/pressures or the customer could require that the product be complete by a certain date. In either case, the software organization may or may not be able to meet the deadline depending on their existing obligations. If the organization is not able to meet the deadline, overtime and extra staff may be required or certain features may need to be left off the release. In developing a schedule, the manager has a number of tools at his disposal; some of the commonly used tools in creating a schedule include Milestone Documents, Project Evaluation Review Technique (PERT) Charts and Gantt Charts.

Disentangling the problem of software project management consists of two primary phases: "planning, including creation and scheduling, and on-going project control" [55]. These incorporate "what is to be done, a decision regarding how to do it, the control of how it is being done, and an evaluation (or measurement) of what was done" [56]. The software development plan typically covers the "what"; however, many more tasks have to be executed in order for a manager to properly manage a software

project. These typically fall into the following categories: planning, organization, staffing, monitoring, controlling, innovating, and representing [57].

According to Simmons [110], every project, irrespective of the industry or job function, including a software development project, is a compromise between three variables: scope, time, and cost:

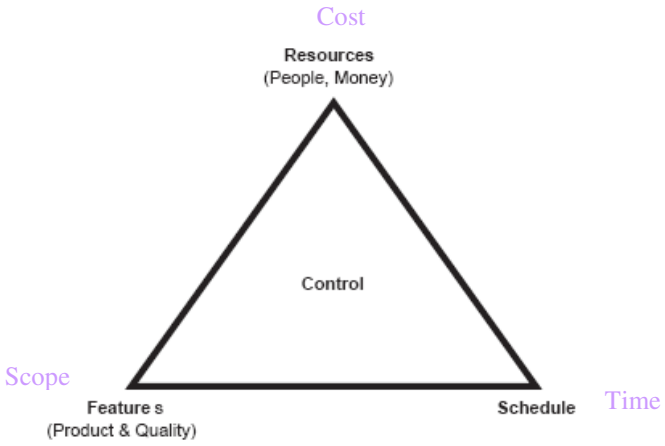


Figure 4: Simmons' project triangle and cost

Planning a project constitutes finding a compromise between the activities that comprise the vertices of the project triangle as shown in Figure 4. The *scope* is the breadth of the project: the sum of the activities that will lead, at the end of the project, to the software product. The total resource usage that is needed to complete the activities identified in the scope is the *cost* or budget. The *Time* is the total elapsed time, from inception to product delivery that is needed to bring the activities identified in the work scope to completion. Software project management is a process of adjusting the variables as preferred to handle the impact of any change across all three.

For a software project to succeed, a manager must reach a compromise between *resource*, *feature*, and *schedule* as seen in Figure 4, so as to maintain compliance with the plan. If any one of the triangle vertices is adjusted, one or both of the other vertices must be modified and the plan has to be tailored for the project to stay on track. If, for example, a project is ahead of schedule, the manager can choose to reduce resources or increase features. If a project is behind schedule, the manager can increase resources or decrease features. If the manager wants to decrease resources, s/he must reduce features or lengthen the schedule. But if the manager wants to add features, s/he must prolong the schedule or add additional resources.

According to Dwayne Phillips [58], all undertakings in a software project include the 3Ps: product, process and people. In order to conform to a project plan, a successful software project needs to keep these three variables in harmony. Without a product, there is no customer, no income, and no software organization; the objective of software development is to create a product. The product must be completed within budget and to a satisfied customer for the project to be deemed a success. In recent years, process has become the most discussed aspect of the 3Ps. This includes some of the famous software process improvement methods, the Capability Maturity Model, the ISO 9000 series, and Best Practices.

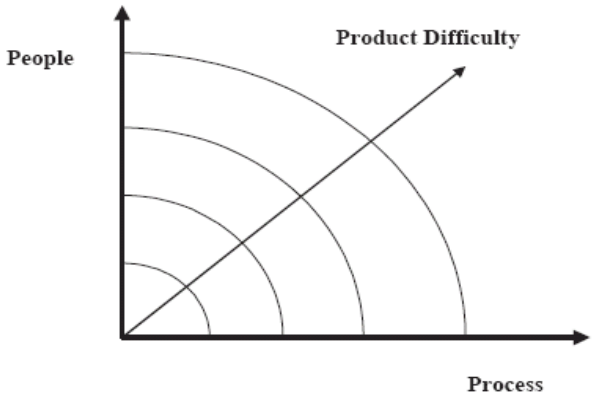


Figure 5: Considering people, process, and product together

Process is significant because it lets people efficiently build products by imposing a structure on the progression of the project. The manager first defines a process needed for the project prior to the commencement of any other activity. A good process is repeatable; however, the same process does not fit all projects, even though they might have similar goals. Some contemporary examples of software development processes include the Unified Process [99], Rapid Application Development [59], Extreme Programming [32], and the Scrum [100]. Typically, managers select a development process based on the organizational culture and the type and estimated size of the software product that is to be built. Lastly, software development is people-intensive; people are critical to software development and maintenance. The best asset on a software project is people who know how to build the product. Figure 5 shows how people, process, and product fit together. The axes represent the capabilities of people and process. The distance from the origin of the graph represents how difficult the

product is to build. The goal of a manager is to keep 3Ps in balance to create a good quality product.

As noted in the CMM Planning KPA [49], a critical portion of developing a plan for a software project is the estimation of the size and cost of the software product. This affects the selected development process, budgeting and scheduling amongst other planning activities. In the next section, we take an in-depth look into research efforts addressing the creation of models aimed at estimating the size and cost of a software product.

2.2 Effort Estimation

Even though the difficulties of software cost estimation were discussed 30 years ago in "The Mythical Man Month" [42], it is as much a relevant area of research now as it was then.

Effort estimation is critical because of the following [107]:

- Exploring the practicality of developing or purchasing a new system
- Determining a price or schedule for a new system
- Planning how to staff a software development project
- Understanding the impact of changing the functions of an existing system

In spite of the importance, software cost estimates are more often than not imprecise, and there is no indication that the software engineering community is making significant gains in making better predictions. We see estimates with greater inaccuracy [66], [67], and in reality most estimates are made informally. The latter fact also

suggests that software developers also have difficulty in applying existing research on software cost estimation.

The consequences of inaccurate estimates of software cost and delivery times is typified by cost overruns that may make a project unprofitable, and delays in delivery time that may result in project failure. Conversely, an overestimate of effort may also adversely affect the competitiveness of a business, for example, where a decision is made to cancel what would otherwise have been finished in time or where the overestimate leads to subsequent overstaffing when a project is completed.

In the course of the past three decades there has been significant research in the area of effort estimation with five classes of estimation models emerging:

- Empirical parametric models
- Empirical nonparametric models
- Analogical models
- Theoretical models
- Heuristic

2.2.1 Empirical parametric models

The most prevalent of estimation models are empirical parametric models. Empirical parametric methods analyze data to establish a numerical model of the relationship between measures of the attributes in the empirical model. Where effort is estimated based on one or more simple measures, these models have been extended, in some cases, by the use of up to 36 of cost drivers.

The most straightforward form of an empirical parametric model is a function that relates a size measure to the effort required to develop a system or program. In this instance, a size measure is a count of some feature of a product of the development process, for example, a count of the number of lines of code in a program. The effort required for development, on the other hand, is typically measured in person-hours, person-days or person-months. Statistical regression modeling is an example of the empirical parametric approach. The most commonly explored models have linear and exponential relationships between effort and the size measure; irrespective of the exact niceties of the model, the general form tends to be:

$$E = a \times V^{b}, \tag{2.1}$$

where E is effort, V is Volume typically measured as lines of code (LOC) or function points, a is a productivity parameter and b is an economies or diseconomies of scale parameter [43], [69], [70].

An alternative empirical parametric methodology is to calibrate a model by estimating values for the parameters (a and b in the case of (2.1)). The most basic method is to assume a linear model that is set b to unity, and then use regression analysis to estimate the slope (parameter a) and possibly introduce an intercept so the model becomes:

$$E = a_1 + a_2 \times V. \tag{2.2}$$

In the above, a_1 represents fixed development costs and a_2 represents productivity.

There are some inherent drawbacks to the development of empirical parametric models, especially if the data set used in their formulation is insufficient. It is not

uncommon for models based on the empirical parametric approach to produce a relatively high error rate, whether the functional form is linear or nonlinear. Conte et al. [72] give an example of a linear model with a correlation coefficient, R^2, of 82% and mean absolute relative error of 37%. Miyazaki et al. [73] give an example of a calibrated COCOMO model with a lower mean absolute relative error of 20%. Courtney et al. [71] suggest that researchers who embark on learning empirical relationships by experimenting with differing combinations of measures and functional forms before selecting the one with the highest correlation typically construct a better model especially with small data sets.

In addition, the possible range of estimates of an empirical parametric model increases with the number of input parameters (each with a range of possible values). The variation in values for an estimate also broadens when the uncertainty in input values is combined with the uncertainty associated with the model. Conte et al. [72] report that a variation in effort of up to 800% is possible in Intermediate COCOMO when the range from highest to lowest values for each cost driver is combined. However, such a wide variation in input values would not necessarily be evidenced in practice.

The re-calibration of empirical models is typically required when they are utilized outside of the organization or environment on whose data they were formulated [74], [76], [77]. Even more generic examples such as COCOMO fail to make precise estimations without calibration. Boehm and Miyazaki et al. [43], [73] illustrate methodologies on how to calibrate estimation models. Models that include a large

number of cost drivers are difficult to calibrate, mainly because the data set required for calibration can be much larger than is available within the organization.

COCOMO II [111] is one of the popular software engineering cost models, which is based on the multiple regression approach. COCOMO was first published in 1981 by Barry J. Boehm [43] as a model for estimating effort, cost, and schedule for software projects. COCOMO II is a relatively recent update of the COCOMO model that was developed in 1997 and finally published in 2001. COCOMO II provides two estimation models: the Early Design and Post-Architecture. The Early Design model is used when a rough estimate is needed based on incomplete project and product analysis. Where as the Post-Architecture model is applied when the top-level design is complete and detailed information is known about the project [112]. The system should have a life-cycle architecture package that includes information on cost driver inputs, architectural alternatives and incremental development strategies.

The Early Design and Post-architecture models use the same functional form to estimate the amount of effort and calendar time it will take to develop a software project. These nominal-schedule (NS) formulas exclude the Cost Driver for Required Development Schedule (SCED). The amount of effort in person-months, PM_{NS}, is estimated by the formula:

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i, \tag{2.3}$$

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

E is the scaling exponent for the effort equation, and F scaling exponent for schedule. The amount of calendar time, $TDEV_{NS}$, it will take to develop the product is estimated by the formula:

$$TDEV_{NS} = [C \times (PM_{NS})^F] \times SCED\% /100, \quad (2.4)$$

$$F = D + 0.2 \times [E - B],$$

where the values of $A,B,C,$ and D are 2.94, 0.91, 3.67 and 0.28, respectively.

Reliable effort prediction is dependent on good size estimation. Software projects are generally composed of new code, code reused from other sources - with or without modifications - and automatically translated code. Size attributes are used to describe physical magnitude, extent or bulk. Software size measures are classified as volume, structure, and rework, and can represent relative or proportionate dimensions. The amount of effort required to produce a software product, the defects remaining in a software product, and time required to create a software product are all estimated using Volume attributes. Of the existing size measures, the SLOC (Source Lines Of Code) attribute is the most commonly accepted because it is [107]:

- Relatively easy to define and discuss unambiguously,
- Easy to objectively measure,
- Conceptually familiar to software developers,
- Used directly or indirectly by most cost estimation models and rules of thumb for productivity estimation, and

- Is available directly from many organization's project databases.

However, there are a number of problems associated with the SLOC size metric as noted by Jones [95]:

- Cross-language comparisons for productivity or quality for the more than 500 programming languages in current use are not accurately supported.
- There is no national or international standard for a source line of code.
- Paradoxically, as the level of language gets higher, the most powerful and advanced languages appear to be less productive than the lower level languages.

Additionally, the SLOC attribute is available after the product has been implemented and thus its use in effort prediction is limited.

Researchers have introduced metrics to address some of the flaws associated with the SLOC measure. The Function Point metric is based on the functional requirements of the software product and can be estimated and counted much earlier than lines of code. Function points let organizations normalize data such as cost, effort, duration, and defects. Even though function points are a popular measure, they too have drawbacks:

- They are based on a subjective measure, which have resulted in a 30% variance within an organization and more than 30% across organizations [77].
- Function points behave well when used within a specific organization, but they do not work well for cross-company bench marking.

Object points are similar to function points. They have the same advantages and disadvantages, but can be estimated and counted earlier than function points. Simmons et al. [52] introduced the Chunk metric, which is a size measure at the cognitive level.

Chunks can be applied to objects, scripts, spreadsheets, graphic icons, application generators, etc.

2.2.2 Empirical nonparametric models

Nonparametric models typically involve the use of artificial intelligence techniques in producing an effort estimate. Briand et al. [78] demonstrate the use of the optimized set reduction (OSR) method, which is a pattern recognition model for analyzing data sets based on classification trees. In [78] the authors contrast the accuracy of the OSR method to a COCOMO model calibrated for the combined COCOMO and Kemerer data sets and to a stepwise regression model. In the comparison, the OSR methodology produced a lower mean absolute relative error than both the two parametric models, with the COCOMO model performing least favorably.

An advantage of OSR is that nominal or ordinal cost driver values can be used as inputs without being mapped to numeric multiplier values. Another advantage is that it can be applied with incomplete input data; especially where only a subset of the cost driver values is known [107]. Srinivasan and Fisher [79] describe two nonparametric methods for generating effort models. One method uses back-propagation to train an artificial neural network and another other uses a learning algorithm to derive a decision tree. Even though they may provide better effort estimates, empirical nonparametric methods such as a neural network are hard to set up and they typically require more work than preparing a statistical regression model [91].

Kemerer [74] tested the artificial intelligence methods on the COCOMO data sets. The effort estimates from the artificial neural network had a lower mean absolute relative error than the decision tree. Discrepancies in the sampling methodologies mean that the results presented by Srinivasan and Fisher [79] are not directly comparable with those of Briand et al. [78], although the same data sets are used. It is probable that the accuracy of both the artificial neural network and the decision tree is comparable with that of OSR and the stepwise regression model. However, Srinivasan and Fisher [79] indicate that the computational cost of training the artificial neural network is high in comparison to the cost of deriving the decision tree.

Most of the research into software project effort estimation has adopted the aforementioned approaches. Some other explorations of the use of artificial intelligence techniques worth mentioning: Wittig et al. [81] described the use of back propagation learning algorithms on a multilayer perception in order to predict development effort. A study [80] on the use of neural nets for predicting software reliability concluded that both feed forward and Jordan networks with a cascade correlation-learning algorithm outperform traditional statistical models. An Albus multilayer perception was used by Samson et al. [82] in order to estimate software development effort. The work compares a neural net approach to linear regression using the COCOMO data set. There have been several attempts to use regression and decision trees to estimate aspects of software engineering. They found that the results were not as conclusive compared to either a statistical model derived from function points or a neural net.

In order to be applied with assurance, both the empirical parametric and nonparametric models that have been described require a large number of data points due to the large number of independent variables and value ranges covered by the models. Both sets of researchers comment on the small size of the COCOMO data set (63 projects) for applying their techniques and on the desirability of all projects in the data set coming from the same environment. Even though the COCOMO data set may be small, it is significantly larger than many organizations could hope to gather. A single organization can provide a large enough data set but it is hard to believe that all the projects would come from the same environment.

The empirical nonparametric models such as the decision tree, artificial neural network, and OSR techniques can still be applied where the number of independent variables is reduced to complement the size of the available data set, for example, lines of code as the single independent variable. However, it is uncertain whether these methods are more accurate than simple regression techniques under those circumstances.

2.2.3 Analogical models

Effort estimation by analogy (EBA) is an established method for software effort prediction. In EBA, the estimated effort of the project under consideration (target project) is a function of the known effort values from analogous historical projects. This is primarily a data-driven process where the attributes common to both the target project and similar historical projects are compared [114] to find a set of 'analogous' projects.

Atkison and Shepperd [86] describe an EBA methodology for estimating development effort where the common attribute used in the comparison is the function

point [30] count of each component. The vector distance of the function point counts from the target project to the historical projects is calculated in order to identify the neighbors of the target project. The estimated effort for the target project is then determined as the weighted mean of the effort values of its neighbors.

Shepperd et al. describe the tool ANGEL, which is based on a generalization of the approach of Atkison and Shepperd [86]. ANGEL affords the user the ability to specify the attributes that are used in the search for analogous projects; it can be directed to search for one, two, or three analogous projects and then calculate an unweighted mean of their effort values to estimate effort for the new project. It also includes a feature to automatically determine an optimal subset of measures for a particular data set of historical projects.

Estimation by analogy is a form of Case Based Reasoning (CBR), which is the process of solving new problems based on the solutions of similar past problems. CBR employs the five basic processes [84]:

- Construction of a representation of the target problem
- Retrieval of a suitable case to act as source analog
- Transfer of the solution from the source case to target
- Mapping the differences between source and target cases
- Adjusting the initial solution to take account of these differences

ESTOR [85] is another example of a case-based reasoning model that is used for software development effort estimation. The cases (software projects) are represented by function point counts and Intermediate COCOMO model inputs. ESTOR retrieves one

case to act as a source analog based on a vector distance calculation of the function point counts from the target case (new project). The initial solution or effort estimate for the target project is the effort value for the analogous project; this is then adjusted to take into consideration the differences between the analog and new project. Adjustments are determined by comparing the values of the measures and applying a set of rules / heuristics determined by an expert. The rules amend the effort value by a multiplier when certain preconditions on the target and source project values are met.

The data set used to develop ESTOR is a subset of 10 projects from the Kemerer [74] data set. On the Kemerer [74] data set, the reported mean absolute relative error for ANGEL is 62%, which compares with more than 100% for the regression models and 53% for ESTOR. ANGEL performed as well as or better than linear and stepwise regression models for effort estimation. The regression models were based on the measures in the data set that displayed the highest correlations with effort. Even though ESTOR appears to outperform ANGEL on this data set, the adjustment rules for ESTOR were developed based on 10 of the 15 projects in the set, and these rules may not be as successful when applied to projects from difference data sets.

It is argued that estimation by analogy offers some distinct advantages to parametric and non-parametric estimation methodologies [115]:

- It avoids the problems associated both with knowledge elicitation and extracting and codifying the knowledge.

- Analogy-based systems only need deal with those problems that actually occur in practice, while generative (i.e., algorithmic) systems must handle all possible problems.
- Analogy-based systems can also handle failed cases (i.e., those cases for which an accurate prediction was not made). This is useful as it enables users to identify potentially high-risk situations.
- Analogy is able to deal with poorly understood domains (such as software projects) since solutions are based upon what has actually happened as opposed to chains of rules in the case of rule-based systems.
- Users may be more willing to accept solutions from analogy based systems since they are derived from a form of reasoning more akin to human problem solving, as opposed to the somewhat arcane chains of rules or neural nets. This final advantage is particularly important if systems are to be not only deployed but also have reliance placed upon them.

2.2.4 Theoretical models

In comparison to the algorithmic (parametric and non-parametric) and analogical models, there is less research on the development of theoretical models for software effort estimation. Wang and Yuan [113] have developed a 'coherent' theory on the nature of collaborative work and their mathematical models in software engineering. This is modeled in the form of the formal economic model of software engineering costs (FEMSEC). The FEMSEC model provides a theoretical foundation for software engineering decision optimizations on the optimal labor allocation, the shortest duration

determination, and the lowest workload/effort and costs estimation. The experiments conducted indicate that the strategy for the optimization of a software engineering project for the lowest cost is to set the project at W (T_{min}, L_0), where L_0 is the optimal labor allocation for a given project and T_{min} is the corresponding shortest duration of the project with L_0. The FEMSEC can be derived by the following steps:

1. Estimate the project size
2. Determine the ideal workload W
3. Allocate the optimal labor L_0
4. Determine the shortest duration T_{min}
5. Minimize the project effort W_{min}
6. Optimize the project cost C_{min}

Abdel-Hamid and Madnick [87]-[89] use dynamic feedback relationships among staff management, software production, planning, and control modeled via a simulation language. The model works from an initial estimate for overall effort and then explores how the actual effort is influenced by the model's assumptions about the interactions and feedback between project and decisions. Simulations of project management scenarios can be run to investigate the effects of management policies and decisions. Their overall contribution is to demonstrate how both underestimates and overestimates of project effort can lead to lower average productivity and increased overall effort.

2.2.5 Heuristic models

Heuristics are rules of thumb, developed through experience that capture knowledge about relationships between attributes of the empirical model. Heuristics can be used to

adjust estimations made by other methods. An example is the COSEEKMO [116] effort-modeling workbench, which applies a set of heuristic rejection rules to comparatively assess results from alternative models. It is intended to provide a solution to the large deviation problem that is seen in model-based estimation methodologies. An experiment was conducted [116] to compare the "before" and "after" effects of applying COSEEKMO to COCOMO and COCOMO II effort estimation models. Reduction in errors of more than 100 % is reported. The COSEEKMO process is also fully automated.

In addition to the formal aforementioned methodologies, expert judgment is also recognized as an estimation approach [43], [66]. Expert judgment is likely to be utilized whenever an expert is available. Experts typically employ one or more of the other methods in making estimations, either informally or formally. The framework for selecting estimation methods does not formally include expert judgment because the method cannot easily be characterized, and it is assumed that it is used whenever experts are available.

It is hard to evaluate which of the above methodologies would be most fitting for a software development project on hand. The easiest to apply are the empirical parametric models; the popular COCOMO II [90] is based on this method. Analogy based estimation models are also straightforward, provided that only a small data set needs to be searched for analogs, and the number of variables to consider is no more than half a dozen. Specific tools are needed to help build analogous models when the number of cases and variables increases above this threshold [41], [81]. Furthermore, the level of similarity of historical projects to the new project development project exerts

great influence on the estimation. Ideally, an estimate should take this difference into account when the new project differs notably from the historical projects.

To an organization's management team and also to the customer, the most interesting software cost estimation measures are total effort and total duration. Once development commences, the remaining totals needed to complete the project also becomes of interest. However, the individual developers are less likely to be interested in total effort estimates. They might prefer to track their own productivity in order to make effort estimates for the activities that they have ownership. For example, in some organizations, developers are expected to sign up to meet the target duration for a particular activity [107]. Estimates based on group productivity figures generally will not be satisfactory, because of the significant variations commonly found between individual developers [92].

Total effort estimates are undoubtedly very desirable at the start of system development. Ironically, this is the time relative to system development activities when there is the least information available on which to base an estimate. Models that predict total effort based on lines of code particularly cannot give an accurate effort estimate because of the lack of detailed information at this time. Models based on function points [30] offer some improvement over lines of code, as it appears that function points can be estimated more consistently from specification and design descriptions than lines of code [74]. On the other hand, considerably more experience and effort is involved in counting function points than lines of code, so data pairs of total effort and function points are likely to be harder to obtain.

Since initial software cost estimates are made based on preliminary data, re-estimating is desirable when additional information is available. Additionally, there is a shift of interest from total effort to total effort to complete development once system development is under way. The computed re-estimate of total effort needs to take into consideration the actual progress that has been made so far, as well as the effort that has been expended. For example, the function point methodology might be used to give a rough preliminary initial estimate of the total effort in the requirements gathering phase of development and a new estimate can be calculated from a re-estimate of the function points, which is made after a high level design is complete.

However, when a re-estimate is calculated the model should not assume the same average productivity for system development for both estimates. For instance, the productivity of the teams involved in differing activities could be substantially different, and thus the new total effort estimate should incorporate this knowledge. The process of re-estimation is made more complicated by such issues, but in order to successfully estimate the total effort or time to complete successfully, effort estimation models need to incorporate these measures.

The environment (including the targeted market) in which a software development project is undertaken has great impact on the software product and the project has a whole. It contributes important factors such as constraints on the availability of staff members that are able to work on the project and it affects the target release date. Estimates of the constraints and the actual estimated feasible release date based on these constraints are needed in order to plan system development or check

whether it is possible to deliver the product within the desired time. Total effort, duration, and staffing are closely related and interdependent, but there may be independent constraints on all three. Due to the constraints and their interdependency of the variables, the complexity of estimating any one or two is increased.

Current parametric models such as COCOMO [43] and Putnam [93] have not proved generally successful in clarifying the relationships among effort, duration, and staffing across a range of organizational settings. Theoretical models such as the dynamic model of Abdel-Hamid and Madnick [89] appear able to better explain interrelationships among duration, staffing, and overall cost in a qualitative way, but they are typically not easy to apply. For instance, the dynamic model of Abdel-Hamid and Madnick [89] requires a specialized simulation tool.

An organization's historical data is very important in model development. The estimated effort is produced by inputting data gathered into the model; in addition, some models require re-calibration using previously collected data before they can be put to use. In order to realize the benefits of collecting local data, experience in developing and applying measures and models must be cultivated within the organization [107]. The simplest models to construct and utilize are empirical parametric models, with few variables, and analogical models. Models that are more difficult to develop and apply are typically based on a large number of variables such as Abdel-Hamid and Madnick [89].

The effort estimation models described thus far like COCOMO II were not defined specifically for the object-oriented development paradigm. In the following

sections we explore Function Point analysis and the variations to the Function Point method that were designed to suite the object-oriented development model.

2.3 Function Point Analysis

The *Function Point* method was introduced in 1979 by Albrecht [30] to measure the size of a data-processing system from the end-user's point of view. The initial counting procedure has been modified several times, resulting in different versions. Since 1986, the International Function Point User Group (IFPUG) has operated as the standard definition body for the function point methodology. The current standard version is reported in the IFPUG Counting Practices Manual [41].

The FP method is applicable at multiple stages of a typical development process, starting from the early requirements definition phase. It measures the size of a system by performing a sequence of steps. The first step is the identification of all functions - each function is classified as belonging to one of the following function types: external input (EI), external output (EO), external inquiry (EQ), internal logical file (ILF), and external interface file (EIF). The function point metric is the weighted totals of five external aspects of the software application:

1. External Input (EI): the types of inputs to application.
2. External Output (EO): the types of outputs to leave the application.
3. External Inquiry (EQ): the types of inquiries that users can make.
4. Internal Logical File (ILF): the types of logical files that the application maintains.
5. External Interface File (EIF): the types of interfaces to other applications.

The first three classes of functions fall within the transaction function typology, while the last two, referring to the logical files, are considered data function types. Each function is then weighted based on its type and on the level of its complexity, in agreement with standard values as specified in the Counting Practices Manual. As an example, for transactions (EI, EO, and EQ), the rating is based on the number of Data Element Types (DETs) and Referenced File Types (FTRs). A DET is a unique, non-repeated field recognized by the user. An FTR can be an ILF referenced or maintained by the transaction or an EIF read by the transaction. Thus, if an external inquiry has more than 16 DETs and at least two FTRs, it is assigned a high complexity level and a weight value equal to 6. The weighted total of the five components of the application is then multiplied by the Value Adjustment Factor, which is computed on the basis of the degree of influence that each of 14 general system characteristics is likely to have on the application. The adjustment factor causes the FP count to vary with respect to the unadjusted count from -35 percent (corresponding to a null degree of influence for all the 14 characteristics) to +35 percent (corresponding to all degrees set to 5) [29].

The FP measure has been used by application developers to estimate productivity, in terms of Function Points per person-month, and quality, in terms of the number of defects per Function Point with respect to requirements, design, coding, and user documentation phases. The success of Function Point method, in part, can be attributed to its early applicability but also to its independence from the language and the tools used throughout the lifecycle. The attractive features of the Function Point approach have motivated several proposals meant to exploit the main ideas of the

method in order to predict the size of object-oriented systems. In the following sections, several adaptations of the FP method to OO systems, besides other size measures conceived for object-orientation are examined.

2.4 Use Case Points (UCP) Model

The Use Case Points (UCP) model [9] is a software sizing estimation method based on use case counts called use case points. A use case is a description of a system's behavior as it responds to a request that originates from outside of that system.

The *use case* technique is used in software and systems engineering to capture the functional requirements of a system. Use cases describe the interaction between a primary actor—the initiator of the interaction—and the system itself, represented as a sequence of simple steps. Actors are something or someone which exist outside the system under study, and that take part in a sequence of activities in a dialogue with the system, to achieve some goal: they may be end users, other systems, or hardware devices. Each use case is a complete series of events or transactions, described from the point of view of the actor [97].

An estimate of effort based on use cases can be made early in a development project as soon as there is some understanding of the problem domain, system size and architecture. Use case modeling is part of the UML 2.0 and is therefore applicable in the early estimation of an object oriented software development project. Below is the UCP estimation method.

Use Case Point (UCP) estimation model:

a) Weighting actors for complexity: a *simple* actor represents another system with a defined API; an *average* actor is either another system that interacts through a protocol such TCP/IP; or a person interacting through a text-based interface and a *complex* actor is a person interacting through a GUI interface.

b) Weight use cases for complexity: a *simple* use case has 3 or fewer transactions; an *average* use case has 4 to 7 transactions; and a *complex* use case has greater than 7 transactions.

c) UUCP (Unadjusted UCP) = Weighted Actors + Weighted Use Cases.

d) Weighing Technical Factor is an exercise to calculate a Use Case Point modifier which will modify the UUCP by the weight of the technical factors.

   SzUC = UUCP * ((0.01 * Tfactor) +0.6).

e) Weighing Environment Factor is an exercise to calculate a Use Case Point modifier which will modify the UUCP by the weight of the Environment factors.

   UCP = SzUC *((-0.03 * Efactor) +1.4).

f) Translating Man-hours from UCP is a matter of calculating a standard usage or effort rate (ER) and multiplying that value by the number of UCPs.

The main limitation of the UCP is that it can only be used early on in an object-oriented project, and a comprehensive methodology that will be able to produce more accurate estimates as the use cases are further realized and refined is needed.

2.5 Object-Oriented Function Point (OOFP) Model

Another related work in the sizing of OOP is the Object-Oriented Function Point (OOFP) model. Most traditional methods for estimating software project effort require an estimate of a size metric (volume + complexity) of the software. For example, the function point size metric uses functional, logical entities such as inputs, outputs, and inquiries that tend to relate more closely to the functions performed by the software as compared to other measures, such as lines of code. The object oriented function point is an adaptation of the traditional function points to the object-oriented paradigm. The *OOFP* as proposed by Antoniol et al. [10] follows the function point counting procedure. Inputs, Outputs and Inquiries are all treated in the same way: they are generically called "service requests" and correspond to class methods. The complexity of service requests depends on the number and type of method parameters. Classes within the application boundary correspond to *ILFS*, while classes outside the application boundary (including libraries) correspond to *EIFS*. The complexity of *ILFs* and *EIFs* depends on the number and type of attributes and associations. Function types contribute to the FPs according to the weights defined by Albrecht [30]. *OOFP* are based on counts of classes, weighted methods per class and data attributes with adjustments for the depth of the inheritance tree, number of children per class and aggregation.

The OOFP is an adaptation of the original FP and although it attempts to use Object Oriented metrics, the framework itself is not very well suited to the object-oriented paradigm. It is simply trying to retrofit a model designed for an earlier development paradigm to the OO paradigm.

2.6 Class Point (CP) Model

The *Class Point* model as defined by Costagliola et al, 2005 [29], is similar to the *OOFP* approach in that it attempts to give an estimate of the size metric based on design/structural artifacts. There are two forms of the Class point metric, named CP_1 and CP_2 respectively. The former is used later in the design stage as more information is available where as CP_1 is meant to be used a bit earlier at the beginning of the design process to carry out a preliminary size estimate.

The process of Class Point size estimation is composed of three main phases:

a) Identification and classification of User Classes: design specifications are analyzed to classify systems components into four types. These are the problem domain type (PDT)/entity classes, the human interaction type (HIT)/boundary classes, the data management type (DMT)/data classes, and the task management type (TMT)/ control classes.

b) Evaluation of a Class Complexity Level: each of the identified classes in the previous step's behavior is taken into account to evaluate its complexity level. In CP_1, the number of external methods and the number of services requested are taken into account; whereas, in CP_2, the number of attributes is also exploited.

c) Estimating the Total Unadjusted Class Point: this consists of computing a weighted total of the classes with their complexity levels determined.

d) Technical Complexity Factor Estimation: this is similar to the UCP where the Total Unadjusted Points are adjusted based on system characteristics such as multiple sites, operational ease, multi-user interactivity etc.

The CP estimation is done only on design artifacts and although it provides a mechanism to accommodate for the refinement of the design artifacts, it does not provide a mechanism to make predictions at the earlier phases, nor a bridge or a means of conversion to relate to earlier metrics.

2.7 SysML Point Overview

In object-oriented development projects, it is desirable to have an estimation model that imitates the continuous realization and refinement of the same system artifacts through the pre-implementation activities of the project development. For example, use cases models are realized into object interaction diagrams and analysis classes, and these are further refined into the class structures that will be coded. The Pattern Point model is a constituent of the proposed SysML point approach (Figure 3). The remainder of this dissertation defines and validates the Pattern Point estimation model.

3. PATTERN POINT ESTIMATION

3.1 Design Patterns

In software engineering, a design pattern is a common reusable solution to a frequently-occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Typically, object-oriented design patterns display relationships and interactions between classes or objects without specifying the final application classes or objects that are involved. Algorithms are not considered design patterns because they solve computational problems and not design problems.

3.1.1 History

Although the practical employment of design patterns is widespread, the concept of a design pattern was not formalized for several years. Patterns, in general, emerged as an architectural concept by Christopher Alexander in 1977. In 1987, Kent Beck and Ward Cunningham began experimenting with the concept of applying patterns to computer programming and presented their results at the OOPSLA conference that year [20], [21]. In the following years, Beck, Cunningham and others followed up on this work.

In the field of computer science design patterns gained popularity after the book *Design Patterns: Elements of Reusable Object-Oriented Software* was published in 1994 (Gamma et al.)[3]. That same year, the maiden Pattern Languages of Programming

Conference was held and the following year, the Portland Pattern Repository was created for documentation of design patterns.

3.1.2 Uses

Design patterns provide tested, proven development paradigms and can thus speed up the development process. Effective software design demands the consideration of issues that may not come to light until later in the implementation stage. The reuse of design patterns helps to avert subtle issues that can cause major problems, and it also improves code readability for developers who are conversant with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem. Moreover, patterns enable developers to communicate using established names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

3.1.3 Classification

Object-oriented design patterns are classified into the categories: *Creational Patterns*, *Structural Patterns*, and *Behavioral Patterns*, and described using the concepts of aggregation, delegation, and consultation [21]. *Creational Patterns* are design patterns that are concerned with object creation mechanisms; trying to create objects in a manner suitable to the situation. *Structural Patterns* are design patterns that ease the design by identifying a simple way to realize relationships between entities. Lastly, *Behavioral*

Patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication. Table 1 lists design patterns classified into the three categories.

3.2 The Pattern Point Model

The Pattern Points (PP) model is an empirical parametric estimation method that utilizes UML sequence diagrams (object interactions) to predict development effort in the analysis phase of an object-oriented development project. It relies on a sizing of each of the 23 object oriented design patterns as defined in the seminal book *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994 (Gamma et al) [3]. Each pattern is sized based on a pattern ranking and an implementation ranking. The pattern ranking metric is a function of the degree of difficulty and the structural complexity of the design pattern; where as the implementation ranking is a function of the ease of applicability of the pattern to the problem type.

The *PP* model focuses on UML sequence diagrams as the modeling representation for object interactions. At the earliest stages, a practitioner is able to compute a range of estimates for a component size using the Pattern Points of the design patterns that might be used in the implementation of each object interaction. As the interaction model is refined and designers have identified which patterns to use in the construction of each object interaction, a single unadjusted component size estimate can be attained. Size estimates are then adjusted to accommodate for technical and

environmental factors such as the lead programmer experience and requirements volatility.

At the late analysis stage where the object interactions have been further refined to reflect some initial design elements, the *PP* metric is computed a little differently. The design pattern implementation ranking together with class metrics of the number of children of abstract classes and interfaces to be implemented, are used in a formulation to compute the PP size estimate.

3.2.1 The pattern point method

The *Pattern Point* size estimation process is composed of three main phases, corresponding to analogous phases in the FP approach [30]. There are two size metrics: PP_1 and PP_2. The former is applicable at the beginning of the analysis phase where a majority of the design constructs have not been formalized, where as the latter takes into account the structural constructs that have been identified in the late analysis phase. Following are the three main steps in estimating the *Pattern Point* size.

3.3 Identification and Classification of User Objects

The user objects that form the design patterns are classified into 4 groups. Table 1 shows a default grouping as defined for the objects that comprise the 23 design patterns as defined by Gamma et al [3]. These are default groupings that are based on the type of components in which the design patterns are typically found.

a. Problem domain type (PDT) – The PDT component contains patterns comprised of objects/classes representing real-world entities in the application domain of

the system. Examples taken from Bruegge et al [9], which describes a distributed information system for accident management, includes objects such as *Incident*, *FieldOfficer*, and EmergencyReport.

b. Human interaction type (HIT) – The objects of HIT type are created to accomplish the need for information visualization and human-computer interaction. With regard to the previous example, the objects *EmergencyReportForm* and *ReportEmergencyButton* belong to HIT.

c. Data management type (DMT) - The objects that belong to the DMT component offer functionality for data storage and retrieval. In the example [3], a DMT component is the *IncidentManagement* subsystem containing classes responsible for issuing SQL queries in order to store and retrieve records representing Incidents in the database.

d. Task management type (TMT) - TMT objects are responsible for the definition and control of tasks. In the example, *Manage-EmergencyControl* and *ReportEmergencyControl* are two objects designed for this purpose. Additionally, a task management type also includes objects responsible for the network communications between subsystems on different hosts. As a matter of fact, *Message* and *Connection* are typical classes falling within this component.

3.4 Evaluation of a Pattern Complexity Level

The second step is to evaluate the complexity level of the design patterns that are found in the object interaction analysis of the system. Two metrics have been formulated which are the *Degree of Difficulty (DD)* and *Structural Complexity (SC)*, for each of the 23

design patterns as identified by Gamma et al [3]. The former is a function of the # of objects and # of messages identified in each design pattern according to its sequence diagram. For example, the degree of difficulty of the *Command* pattern in Figure 6 is 7, as there are 4 objects and 3 messages passed between the objects in the diagram.

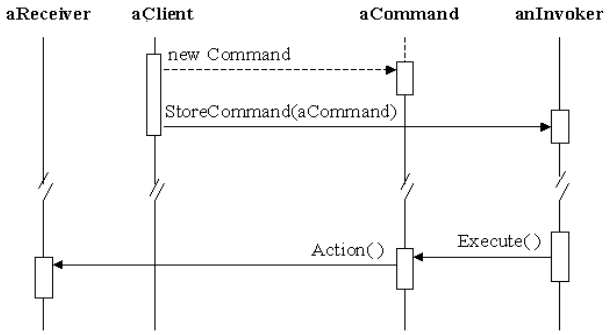


Figure 6: Sequence diagram for the *Command* design pattern

The structural complexity is a function of the # of classes and # of associations that are identified in the structure of the design pattern. For example, the structural complexity of the *Abstract Factory* design pattern in Figure 7 is 7 because there are 3 classes and 4 associations (concrete classes are not counted). Concrete class implementations of interfaces and abstract classes are more readily available in the late analysis stage and are included the PP_2 metric. Table 1 lists the degree of difficulty and structural complexity of the 23 design patterns identified by Gamma et al [3] and in addition two common object oriented design patterns not listed in [3] that are in blue. These are the *Interface* pattern and the *Filter* pattern as defined in [4]. The PP_1 metric is a function of the *Degree of Difficulty (DD)* and *Structural Complexity (SC)* of the design

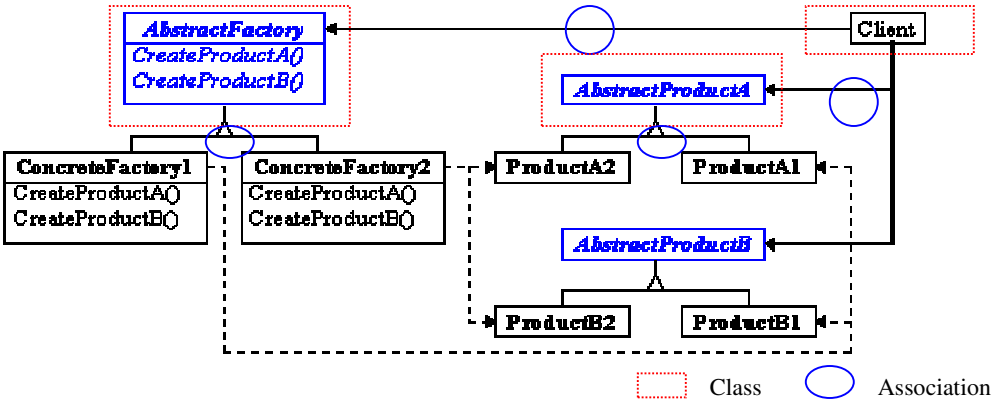


Figure 7: Structural diagram of the *Abstract Factory* design pattern

pattern, and PP_2 takes the number of implemented concrete classes in the pattern also into consideration. For example, in Fig. 3, *ProductA1* and *ProductB1* are examples of *Pattern Concrete (PC)* classes.

$$PP_2 = PP_1 + \#\text{ of Pattern concrete classes } (PC). \tag{3.1}$$

The PP_2 metric is applicable at the late analysis early design stages where more of the concrete implementations have been identified.

Table 1: The 23 design patterns categorized by design pattern type and the corresponding *DD, SC and Complexity* values

Type	Pattern	objs.	mesgs.	DD	classes	assoc.	SC	Group	Complexity
Creational	Abstract Factory	2	1	3	3	4	7	PDT	High
	Builder	3	4	7	3	4	7	PDT	High
	Factory Method	1	1	2	2	2	4	PDT	Low
	Prototype	2	1	3	2	1	3	PDT	Low
	Singleton	1	1	2	2	1	3	PDT	Low
Structural	Adapter	2	1	3	4	3	7	PDT	Average
	Bridge	2	2	4	5	5	10	PDT	High
	Composite	2	1	3	4	5	9	PDT	High
	Decorator	2	1	3	4	3	7	HIT	Average
	Façade	1	0	1	1	1	2	PDT	Low
	Flyweight	2	1	3	5	5	10	PDT	High
	Proxy	2	1	3	1	1	2	PDT	Low
Behavioral	Chain of Responsibility	2	1	3	3	3	6	TMT	Average
	Command	4	3	7	5	3	8	TMT	High
	Interpreter	3	1	4	4	4	8	TMT	High

Table 1: Continued,

Type	Pattern	objs.	mesgs.	DD	classes	assoc.	SC	Group	Complexity
Behavioral	Iterator	1	1	2	3	2	5	DMT	Low
	Mediator	3	3	6	4	4	8	TMT	High
	Memento	3	4	7	3	2	5	DMT	High
	Observer	2	5	7	4	3	7	TMT	High
	State	2	3	5	3	2	5	TMT	Average
	Strategy	2	3	5	3	2	5	PDT	Average
	Template Method	1	2	3	2	1	3	PDT	Low
	Visitor	3	3	6	5	5	10	TMT	High
	Filter	2	1	3			2	PDT	Low
	Interface	1	0	1	2	1	3	PDT	Low

The Complexity Level as identified in the *Complexity* column in Table 1 is based on an entry mapping of the DD and SC metric in Table 2. Each design pattern is assigned a complexity level of Low, Average or High depending on the size of the corresponding DD and SC metrics.

Table 2: Evaluation of the complexity level of a design pattern

	0 – 4 SC	5 – 8 SC	>= 9 SC
0 – 2 DD	LOW	LOW	AVERAGE
3 – 5 DD	LOW	AVERAGE	HIGH
>= 6 DD	AVERAGE	HIGH	HIGH

3.5 Estimating the Total Unadjusted Pattern Point

After estimating the complexity of each of the design patterns found in the object interaction analysis of the system according to Table 2, we can now compute the *Total Unadjusted Pattern Point (TUPP)*. To achieve this, Table 3 below, as defined in the Class Point estimation [29] is completed for Pattern Point estimation.

Table 3: Evaluating the *TUPP*

System Component Type	Description	Complexity			
		Low	Average	High	Total
PDT	Problem Domain	…*3=…	…*6=…	…*10=…	…
HIT	Human Interaction	…*4=…	…*7=…	…*12=…	…
DMT	Data Management	…*5=…	…*8=…	…*13=…	…
TMT	Task Management	…*4=…	…*6=…	…*9=…	…
TUPP		Total Unadjusted Pattern Point			

The entries in the table above express the weighted number of patterns whose typology and complexity level are given by the corresponding row and column, respectively. In summary, the *TUPP* is computed as the weighted total of the four components of the application:

$$TUPP = \sum\sum w_{ij} \times x_{ij}, \tag{3.2}$$

where x_{ij} is the number of patterns of component type i (problem domain, human interaction, etc.) with complexity level j (low, average, or high), and w_{ij} is the weighting value for type i, and complexity level j.

3.6 Technical Complexity and Environmental Factor Estimation

The *Technical Complexity Factor (TCF)* [9] is determined by assigning the degree of influence (ranging from 0 to 5) that 13 general system characteristics have on the application, from the designer's point of view. The estimates given for the degrees of influence are recorded in the Technical factors table illustrated in Table 4. The sum of the influence degrees related to such general system characteristics forms the *Technical Factor (TFactor),* which is used to determine the *TCF* according to the following formula:

$$TCF = 0.6 + (0.01 * TFactor). \tag{3.3}$$

The *Environmental Adjustment Factor (EAF)* [9] is determined by factors that represent some characteristics existent at the development environment that could influence the software cost. Each factor from Table 5 receives a value and the Environmental Adjustment Factor (FAA) is given by

$$EAF = 1.4 + (-0.03 * EFactor) \tag{3.4}$$

The final value of the Adjusted *Pattern Point (PP*) is obtained by multiplying the *Total Unadjusted Pattern Point* value by the *TCF* and *EAF*

$$PP = TUPP * TCF * EAF \tag{3.5}$$

The *PP* count can vary with respect to the unadjusted count from -45 percent (corresponding to a null *TDI*) to +45 percent (corresponding to all degrees set to 5), due

to the adjustment factor. It is worth mentioning that the *Technical Complexity Factor* and *Environmental Adjustment Factor* are determined by taking into account the characteristics that are considered in the FP.

Table 4: Technical factors

Factor	Description	Weight
T1	Distributed system	2
T2	Response or throughput performance objectives	2
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Includes security features	1
T12	Provide access for third parties	1
T13	Special user training facilities are required	1

Table 5: Environmental factors

Factor	Description	Weight
F1	Familiar with Rational Unified Process	1.5
F2	Application experience	0.5
F3	Object-oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	-1
F8	Difficult programming language	-1

4. THEORETICAL VALIDATION

The *PP* metric as well as its composite metrics: *DD*, *SC* and *PC* have been defined so far, but a software measure can be acceptable and effectively usable only if its usefulness has been proven by means of a validation process. The goal of such a process is to convey that a measure really measures the attribute that it is supposed to and it is practically useful [29]. It is widely accepted that two forms of validation are required: theoretical and empirical validation. Theoretical validation is a fundamental step in the validation process and should allow one to demonstrate that a measure satisfies properties characterizing the concept (e.g., size, complexity, coupling, etc.) it is intended to [5]. Once the measure satisfies the properties, its usefulness can be verified by carrying out an empirical validation process, which usually employs the use of statistical analysis techniques.

Several authors have defined measurement theoretical principles that software measures should adhere to in order to be valid [6]-[8]. Nonetheless, as indicated by Briand et al. in [12], the software engineering community might gain from the adoption of a more pragmatic approach that is able to provide more practical results. Such concerns have led several researchers [13]-[16] to provide guidelines on frameworks for theoretical and empirical validation of measures. Specifically, it is suggested that the measures should conform to certain fundamental properties. In addition, measures of internal attributes should be validated empirically against external attributes.

In this section, the general framework proposed by Briand et al. [5] is applied in the theoretical validation. The framework contributes to the definition of a stronger

theoretical ground of software measurement by providing convenient and intuitive properties for several measurement concepts, such as complexity, cohesion, length, coupling and size. The generality of the approach is due to the fact that the properties characterizing these concepts are independent of the software artifacts (e.g., software specification, design, code) the concepts are applied to [29].

The theoretical validation is conducted by evaluating the *Pattern Point* approach against the properties proposed by Briand et al [5] that are specific to size measures. Consideration is given only to the PP_2 measure since the corresponding theoretical validation process can also be applied to PP_1 as a special case.

In [5], three properties are defined that are specific to the size measures, namely, *Nonnegativity*, *Null Value*, and *Module Additivity*. It is important to mention that such properties are requisite but not sufficient. Still, they constrain the search for measures and "make the measure definition process more rigorous and less exploratory." Before describing the analysis process that was performed on the PP_2 measure, the definitions of the general framework and the properties that the size measures are supposed to verify are detailed in the following section. Within the framework, a system is characterized as a set of elements and a set of relationships between those elements, as formalized in the following definition.

4.1 Representation of Systems and Modules

A system S will be represented as a pair $< E, R >$, where E represents the set of elements of S and R is a binary relation on E ($R \subseteq E \times E$) representing the relationships between S's elements. Given a system $S = < E, R >$, a system $m = < E_m, R_m >$ is a module of S if

and only *if* $E_m \subseteq E$, $R_m \subseteq E_m X E_m$, and $R_m \subseteq R$. The elements of a module are connected to the elements of the rest of the system by incoming and outgoing relationships. The set *InputR(m)* of relationships from elements outside module *m* to those of module *m* is defined as:

$$InputR(m) = \{< e_1, e_2 > \in R \mid e_2 \in E_m \text{ and } e_1 \in E - E_m\}$$

The set of *OutputR(m)* of relationships from the elements of a module m to those of the rest of the system is defined as:

$$OutputR(m) = \{< e_1, e_2 > e_1 \in R \mid \in E_m \text{ and } e_2 \in E - E_m\}$$

The basic properties of size measures are very intuitive; they ensure that the size cannot be negative, it is null when the system has no element, and it can be obtained as the sum of the size of its modules when they are disjoint. More formally:

Property Size 1: (Nonnegativity). The size of a system $S = < E, R >$ is nonnegative

$$Size \geq 0$$

Property Size 2: (Null Value). The size of a system $S = < E, R >$ is null if E is empty

$$E = \phi \Rightarrow Size(S) = 0$$

Property Size 3: (Module Additivity) The size of a system $S = < E, R >$ is equal to the sum of the sizes of two of its modules $m_1 = <E_{m1}, R_{m1}>$ and $m_2 = <E_{m2}, R_{m2}>$ such that any element of *S* is an element of either m_1 *or* m_2

$$(m_1 \subseteq S \text{ and } m_2 \subseteq S \text{ and } E = E_{m1} \cup E_{m2} \text{ and } E_{m1} \cap E_{m2} = \phi)$$

$$\Rightarrow Size(S) = Size(m_1) + Size(m_2).$$

4.2 Theorem

The Nonnegativity, Null Value, and Module Additivity properties hold for the Pattern Point measure.

4.3 Proof

Since the *PP* value is obtained as a weighted sum of nonnegative numbers, the *Nonnegativity* property holds. If no design pattern (i.e. classes/objects, associations/calls) is present in the system analysis the *PP* value is trivially null and the Null Value property is also verified.

In order to prove the Module *Additivity* property, let S = < E, R > be the system, and let $m_1 = <E_{m1}, R_{m1}>$ and $m_2 = <E_{m2}, R_{m2}>$ be its modules, such that the following condition holds: $m_1 \subseteq S$, $m_2 \subseteq S$, $E = E_{m1} \cup E_{m2}$, *and* $E_{m1} \cap E_{m2} = \phi$).

Let us observe that the conditions $m_1 \subseteq S$, $m_2 \subseteq S$ and $E = E_{m1} \cup E_{m2}$ imply that no modification is made to the design patterns of S when the system is partitioned into modules m1 and m2. This means that for each pattern, the values for *DD* and *SC* will be unchanged after the partitioning. Indeed, the *DD*, *SC* and *PC* values in a design pattern are the same no matter how the design pattern is used in the system, i.e., regardless of the actual connections among modules.

5. EMPIRICAL VALIDATION

In the literature, it is largely accepted that system size is strongly correlated with development effort [17]-[20]. The theoretical validation conducted in the previous section illustrates that the *Pattern Point* measures satisfy properties that are considered requisite for size measures. However, a theoretical validation alone does not guarantee the usefulness of the measures as predictors of effort and cost. Moreover, an empirical evaluation process is needed to verify their predictive power. Thus, the author has performed an empirical study purposed to determine whether the *Pattern Point* measures can be used to predict the development effort of OO systems in terms of person-days (8 hours per day).

The subject of the study was the initial release of the IBM Lotus Quickr software product. This particular product was chosen as the subject because of its extensive use of design patterns and the ample documentation on the effort expended per use case that existed. The experimentation on the data from the Quickr 8.0 release has provided initial evidence of the effectiveness of the *Pattern Point* approach.

5.1 IBM Lotus Quickr 8.0

Lotus Quickr is IBM's team collaboration and content sharing software that helps users access and interact with the people, information and project materials that they need to get their work done. Lotus Quickr has a rich set of features such as content libraries to share everyday business files, team discussion forums and blogs to facilitate communications, wikis that let you author and edit content in place together, and

connectors that help make sharing easier than ever right from your favorite desktop software such as Lotus Notes, Lotus Sametime, Lotus Symphony, Microsoft Office and Microsoft Outlook.

The IBM Lotus Quickr 8.0 release was selected as the subject of this study because of the extensive use of design patterns in its implementation, the careful recording of the effort expended in completing each project task, which was also grouped by use case, and also because the 8.0 release was the first release and thus there was more emphasis on new code development. The software was released June 2007 and the Pattern Point method was applied retroactively on the recorded data.

5.2 Applying the Pattern Point Method to Lotus Quickr

Like many software development projects, there was incomplete documentation particularly with respect to the artifacts from the analysis phase of the software product i.e. there was little or no documentation of object interaction analyses including sequence diagrams. However, there was ample data on implemented use case scenarios, and also the package structure of the code was designed for easy identification of the design patterns in play, which helped in the reverse engineering of the object interaction diagrams in the following section. For instance, the package structure below identifies a *Mediator* design pattern and concrete class implementations of the classes involved in the design pattern:

com.ibm.content.clb.mediator.handler.OperationHandlerImpl

com.ibm.content.clb.mediator.handler.UserOperationHandlerImpl

com.ibm.content.clb.mediator.handler.ChildMetadataHandler

com.ibm.content.clb.mediator.cache.CustomDataCacheEventListener

Where as, the next set of package names identify the *Visitor* design pattern and the concrete class implementations of the classes involved in the design pattern:

com.ibm.content.service.clb.serialization.visitor.ImportItemVisitorContainer

com.ibm.content.service.clb.serialization.visitor.impl.ImportCopyObjectVisitor

com.ibm.content.service.clb.serialization.visitor.impl.ImportRemoveControlData Visitor

The reverse engineering tool MaintainJ was employed to reverse engineer the object interaction diagrams involved in a particular use case.

5.3 Reverse Engineering Using MaintainJ

MaintainJ is an Eclipse plug-in that generates runtime UML sequence and class diagrams for a use case. This occurs in three steps:

1. Instrument the Quickr server
2. Perform the use case scenario in the application
3. Generate trace files

In the first step, *Instrument the Quickr server*, MaintainJ uses AspectJ technology [28], which allows for byte-code weaving directly in the Virtual Machine so that MaintainJ developers can write *aspects* (which insert instrumentation software) for code in binary (.class) form. With respect to Quickr, which runs off a WebSphere Portal server, this

step involves deploying the MaintainJ.war application onto the Portal server. In Step 2, the user can now log in to the Quickr application and perform use case scenarios with the MaintainJ application running. With the MaintainJ application started we are able to capture the flow of message calls between the objects that are involved in the actualization of the use case. Once this is complete, in step 3, the user can now output a trace file of the object interactions, which can then be imported into the Eclipse IDE. With the MaintainJ eclipse plug-in installed, the user can view run-time sequence and class diagrams. The author has written a separate tool that takes as input the trace file and it outputs class and method names involved in the object interactions to a text file.

In order to calculate *the Pattern Point* metrics for the use cases, the author has implemented an algorithm that identifies the design patterns based on the package name and structure of the outputted trace for each use case scenario. The program then performs the Complexity Level calculations in section 3.4 and generates corresponding $TUPP_1$ and $TUPP_2$ metrics for each use case.

The *Technical Complexity Factors (TCF)* and *Environmental Adjustment Factors (EAF)* were not included in the experiment. The first reason is because the use cases were all taken from the same project and thus the factors would not vary much between the use cases. Furthermore from the literature, some studies have revealed that the application of the Processing Complexity Factor to the raw Function Point measure can have little impact on the performance of Function Points in the cost estimation process [38], [43], [68]. The same was also verified in the Class Point approach, i.e. whether or

not the 14 Function Point factors are useful in our context, and also if the four additional Class Point factors enhance the prediction accuracy [29].

As a side effect, the experiment has ascertained the author's intuition that the *Pattern Point* measures do not require a long training, and is not labor intensive. As a matter of fact, one of the objectives when the measures were conceived was to overcome one of the criticisms to *Function Point Analysis* approach [21]-[23]. Additionally, in order to arrive at a more realistic accuracy of the derived models and have more reliable nonbiased results, an 8-fold cross-validation approach was utilized in the empirical validation. Multiple-fold cross validation has been successfully used in the literature in order to validate cost estimation models (see references [24]-[27]). The process is especially recommended to increase the accuracy of prediction models when dealing with small data sets. In the remainder of section 6, the cross validation process applied to PP_1 and PP_2 is described. Subsequently, a comparison of the results gained for PP_1 and PP_2 with other measures is performed in Section 7.1.

5.4 The Cross Validation Process

To carry out the cross validation process on the 78 selected use cases from the Lotus Quickr, the following steps were performed:

1. The whole data set was partitioned into eight randomly selected test sets; seven of equal size (10) and the last test set had two less data elements (8). For each data set, the remaining use cases were analyzed to identify the corresponding training set obtained by removing influential outliers.

2. An Ordinary Least-Squares (OLS) regression analysis was performed on each training set to derive the effort prediction model.
3. Accuracy was separately calculated for each test set and the resulting values have been aggregated across all 8 test sets.

In what follows, we describe each of the above steps.

5.5 Partitioning the Data Set

Table 6 reports the data of the 78 use cases, following the order resulting from the random partition performed. Thus, the first ten use cases form the first test set, the subsequent ten use cases form the second one, and so on.

Table 6: The data for the 78 use cases

Use case	DD	SC	PC	EFD	TUPP1	TUPP2
1	27	27	11	13	33	44
2	0	0	0	4	0	0
3	48	61	29	20.5	78	109
4	97	147	122	39	165	287
5	51	74	79	18	81	160
6	37	51	46	16	63	112
7	58	80	88	28	93	181
8	11	13	3	7	15	18

Table 6: Continued,

Use case	**DD**	**SC**	**PC**	**EFD**	**TUPP1**	**TUPP2**
9	62	87	89	28	96	185
10	35	41	18	13	48	66
11	72	101	62	35	124	280
12	57	76	50	46	90	140
13	11	13	3	4	15	18
14	34	38	14	15	45	59
15	11	13	3	4	15	18
16	45	73	34	24	78	112
17	49	68	30	22	75	105
18	34	38	14	13	45	59
19	31	34	7	18	48	55
20	11	13	3	10	15	18
21	11	13	1	4	15	16
22	0	0	0	7	0	0
23	11	13	1	9	15	16
24	43	64	45	17	69	114
25	43	82	30	22	87	117
26	11	13	3	10	15	18
27	59	80	116	30.5	96	212
28	0	0	0	7	0	0

Table 6: Continued,

Use case	**DD**	**SC**	**PC**	**EFD**	**TUPP1**	**TUPP2**
29	35	51	60	27	60	123
30	37	48	8	18	62	70
31	27	38	4	10	45	49
32	72	111	78	55	125	203
33	73	114	66	31	128	194
34	29	30	4	7	39	43
35	11	13	3	4	15	18
36	45	61	30	17	75	108
37	35	41	18	18	48	66
38	11	13	1	7	15	16
39	83	130	70	28.5	145	215
40	0	0	0	3	0	0
41	36	48	20	20	60	83
42	0	0	0	13	0	0
43	56	79	81	22	87	168
44	35	41	18	16	48	66
45	94	138	118	39	156	274
46	57	83	81	31	103	187
47	42	61	15	20	66	81
48	48	61	29	17	78	109

Table 6: Continued,

Use case	DD	SC	PC	EFD	TUPP1	TUPP2
49	32	39	12	7	48	60
50	0	0	0	6	0	0
51	40	66	33	23	77	110
52	47	67	47	19.5	81	130
53	0	0	0	8	0	0
54	42	55	34	16	66	100
55	62	84	52	21	96	148
56	48	69	67	21	75	142
57	11	13	3	7	15	18
58	56	79	31	24	87	118
59	54	77	79	27	91	170
60	37	48	8	19	62	70
61	69	77	34	27	108	142
62	11	13	3	10	15	18
63	0	0	0	7	0	0
64	63	90	103	31	106	209
65	47	61	16	20	72	88
66	58	85	84	36	93	177
67	0	0	0	5	0	0
68	0	0	0	4	0	0

Table 6: Continued,

Use case	DD	SC	PC	EFD	TUPP1	TUPP2
69	53	63	30	25	84	117
70	51	74	76	28	91	169
71	51	72	47	22	81	128
72	51	76	51	25	81	132
73	11	13	2	0	15	17
74	44	38	8	15	66	74
75	0	0	0	5	0	0
76	42	61	64	22	66	130
77	100	144	70	39	163	233
78	58	74	46	27	100	146

Descriptive statistics have been computed both for the variable Effort (denoted by *EFD*), expressed in terms of person-days (8 hours/day), and the variables PP_1 and PP_2, related to the 78 use cases. The summary statistics of those variables are given in Table 7.

Table 7: Descriptive statistics: *EFD,* PP_1, PP_2

Variable	Obs.	Min	Max	Median	Mean	Std Dev.
Days	78	0.00	55.00	18.00	18.38	11.13
PP_1	78	0.00	165.00	66.00	60.42	42.80
PP_2	78	0.00	287.00	94.00	95.36	76.10

A careful outlier analysis was performed in order to remove possible extreme values, which may unduly influence the models obtained from the regression analysis. For each data point, the Cook's distance was calculated to verify its influence on the generated model. In regression models, Cook's distance is the standard statistic to detect influential observations—it measures the overall effect that omitting a given observation would have in the model. Figure 8 shows example outliers in the training set. Thus, any influential outlier has been omitted from the corresponding initial training set, resulting in the sets used to derive the models.

Influential observation for the training sets are determined based on Cook's distances greater than the threshold value 0.10, corresponding to the ratio 8 and the sample size 78. The other data points have been retained since they are not influential observations and cannot prejudice the results. Table 8 shows that for PP_1 use case 12 represents outliers for training sets 1, 3, 4, 5, 6, 7 and 8, and that the use cases 32 and 39 represent outliers for training sets 1, 2, 3, 5, 6, 7 and 8. Where as for $PP_{2,}$ use case 11 is also detected and represents outliers for training sets 1, 3 and 4; use case 12 represents outlier for training sets 1, 3, 4, 5, 6, 7 and 8; use case 32 represents outliers for training sets 1, 2, 3, 5, 6, 7 and 8; and lastly use case 39 represent outliers for training sets 5, 6 and 7.

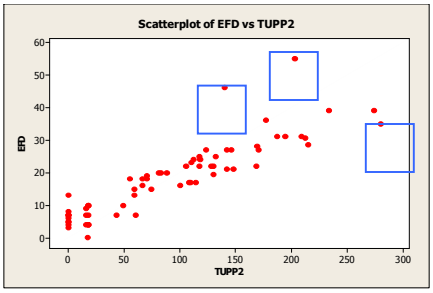
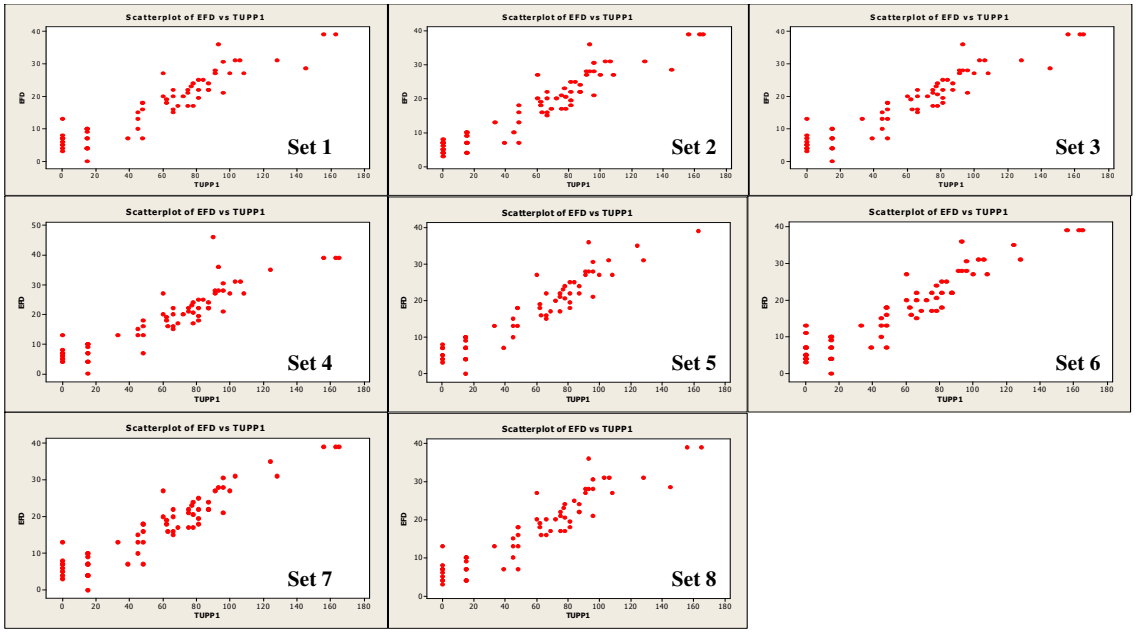


Figure: 8 Outliers in the training set

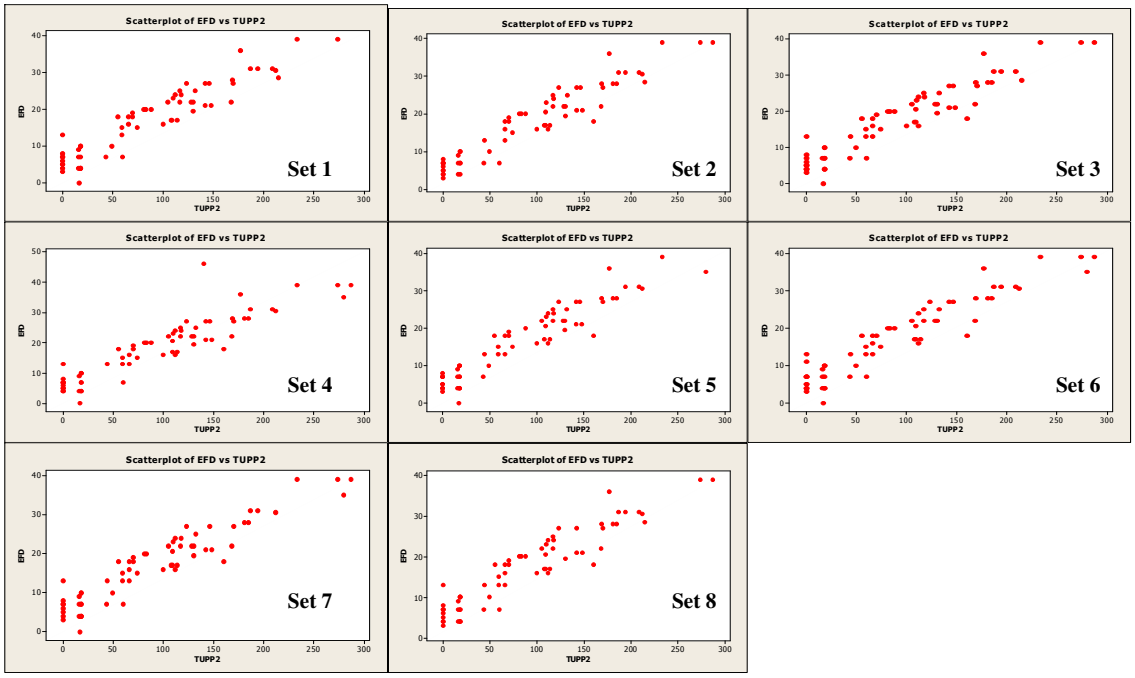
Table 8: The values of Cook's distance for outliers of PP_1 and PP_2

PP_1	**Set 1**	**Set2**	**Set3**	**Set4**	**Set5**	**Set6**	**Set7**	**Set8**
12	0.175		0.172	0.271	0.192	0.170	0.184	0.182
32	0.450	0.591	0.433		0.484	0.421	0.478	0.475
39	0.154	0.134	0.131		0.176	0.142	0.131	0.153
PP_2	**Set 1**	**Set2**	**Set3**	**Set4**	**Set5**	**Set6**	**Set7**	**Set8**
11	0.195		0.126	0.126	0.002	0.002	0.004	0.112
12	0.184		0.168	0.261	0.192	0.170	0.184	0.181
32	0.469	0.642	0.420		0.484	0.420	0.478	0.466
39	0.048	0.039	0.030		0.176	0.142	0.131	0.028

For each test set, the remaining 68 projects (70 for the last test set) are considered as an initial training set. In order to verify the presence of a positive linear relationship between each of the measures PP_1 and PP_2 and the effort, a scatter plot for each training set was produced. As illustrated in Figures 9a and 9b, for either measure, each scatter plot shows a positive linear relationship between the variables involved. This suggests that a linear regression analysis of *EFD* and PP_1 (respectively, *EFD* and PP_2) can be performed.



(a)



(b)

Figure 9: The scatter plots for (a) *EFD* and PP_1, and (b) *EFD* and PP_2, resulting from the OLS regression applied to the four training sets

5.6 OLS Regression Analysis to Derive Effort Prediction Models

An Ordinary Least-Squares regression analysis was applied in order to perform an empirical validation of the PP_1 and PP_2 measures. OLS allows the analyst to determine the equation of a line that can be used to predict the development effort in terms of the number of person-days required. This modeling technique has often been used for validation purposes due to its useful predictive capability and to the mature statistical packages supporting it. For this experiment, the statistics package *Minitab 15 Statistical Software English* was employed in deriving the models.

When applying the OLS regression, a number of determinative indicators have been taken into account to establish the quality of the prediction. An important measure is the goodness of fit of a regression model, which is determined by the Coefficient of Determination, R^2. The Coefficient of Determination measures the percentage of variation in the dependent variable explained by the independent variable. Furthermore, to evaluate the statistical significance a t-test was performed and the p-value, t-value of the coefficient and intercept for each model was determined. Specifically, a significance threshold of 0.05 for the p-value is generally used to establish whether a variable is a significant predictor. When it is less than 0.05, we can reject the hypothesis that the coefficient is zero; the reliability of the predictor is then given by the t-value of the coefficient. The commonly used threshold is 1.5, so a t-value greater than 1.5 indicates that the predictor is reliable at a risk level of 5 percent or less and hence it is a reliable predictor. It is important to note that the p-value of the coefficient coincides with the overall p-value of the model - Significant F - which is related to the probability that the

independent variable impacts the dependent variable, i.e., that the regression equation is significant. Furthermore, by squaring the t-value we can obtain the F value, due to the relation between Student's t distribution and Fisher-Snedecor's F distribution. As for the intercept, the corresponding p-value provides the probability that it is zero. Thus, also for the intercept a high t-value, together with a low p-value indicate that the null hypothesis can be rejected.

In tables 9 to 16, the results of the OLS regression carried out with each training set are presented. It can be observed that for each training set, the linear regression analysis shows a high R^2 value, for both PP_1 and PP_2, with a slightly higher value for PP_2 in training sets 1, 2, 6, 7, and 8, and a higher R^2 for PP_1 in training sets 3, 4, and 5. As an example, consider training set 1. For PP_1, we have $R^2 = 0.86$, which indicates that 86 percent is the amount of variance of the dependent variable EFD that is explained by the model related to PP_1, whereas for PP_2, we have $R^2 = 0.88$ indicating that 88 percent is the amount that is explained by the model related to PP_2. For this training set, the equation of the regression model for PP_1 is:

$$EFD = 4.486 + 0.223\ PP_1, \tag{5.1}$$

where the coefficient 0.223 and the intercept 4.486 are significant at level 0.05, as from the t-test. The equation of the regression model for PP_2 on the other hand, is:

$$EFD = 5.84 + 0.129\ PP_2 \tag{5.2}$$

where the coefficient 0.129 and the intercept 5.84 are again significant at level 0.05. In table 17, the mean and median R^2 values have also been calculated for $PP_1$and PP_2. for PP_1, both values are 0.881, and for PP_2 the mean and median R^2 values are 0.879 and

0.884 for PP_1 and PP_2 respectively. These set of results have led to the conclusion that the R^2 values for PP_1 are comparable to that PP_2, but the median PP_2 slightly outperforms PP_1.

Table 9: The results of the OLS regression analysis for training set no. 1

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.486 + 0.223 PP_1	0.932	0.869	3.59	418.879	0.000
PP_2	EFD = 5.84 + 0.129 PP_2	0.940	0.884	3.33	479.930	0.000

(b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.224	0.010	20.467	0.000
	Intercept	4.486	0.764	5.867	0.000
Model for PP_2	Coefficient	0.129	0.006	21.907	0.000
	Intercept	5.843	0.660	8.860	0.000

Table 10: The results of the OLS regression analysis for training set no. 2

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.647 + 0.219 PP_1	0.935	0.875	3.552	446.216	0.000
PP_2	EFD = 5.873 + 0.125 PP_2	0.950	0.903	3.09	588.286	0.000

(b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.219	0.010	21.124	0.000
	Intercept	4.647	0.752	6.180	0.000
Model for PP_2	Coefficient	0.125	0.005	24.255	0.000
	Intercept	5.873	0.632	9.286	0.000

Table 11: The results of the OLS regression analysis for training set no. 3

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.13 + 0.225 PP_1	0.943	0.889	3.365	505.711	0.000
PP_2	EFD = 5.709 + 0.127 PP_2	0.940	0.884	3.397	479.433	0.000

Table 11: Continued, (b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.225	0.010	22.488	0.000
	Intercept	4.13	0.735	5.623	0.000
Model for PP_2	Coefficient	0.127	0.006	21.896	0.000
	Intercept	5.709	0.689	8.289	0.000

Table 12: The results of the OLS regression analysis for training set no. 4

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.877 + 0.221 PP_1	0.938	0.88	3.411	483.639	0.000
PP_2	EFD = 6.512 + 0.124 PP_2	0.909	0.823	4.347	312.776	0.000

(b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.221	0.010	21.992	0.000
	Intercept	4.877	0.729	6.693	0.000
Model for PP_2	Coefficient	0.124	0.007	17.685	0.000
	Intercept	6.512	0.856	7.609	0.000

Table 13: The results of the OLS regression analysis for training set no. 5

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.144 + 0.228 PP_1	0.945	0.893	3.279	523.493	0.000
PP_2	EFD = 5.572 + 0.129 PP_2	0.942	0.888	3.249	489.266	0.000

(b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.228	0.010	22.880	0.000
	Intercept	4.144	0.698	5.935	0.000
Model for PP_2	Coefficient	0.129	0.006	22.119	0.000
	Intercept	5.572	0.650	8.576	0.000

Table 14: The results of the OLS regression analysis for training set no. 6

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.343 + 0.226 PP_1	0.940	0.883	3.570	474.896	0.000
PP_2	EFD = 6.063 + 0.124 PP_2	0.941	0.886	3.503	488.507	0.000

Table 14: Continued, (b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.226	0.010	21.792	0.000
	Intercept	4.343	0.738	5.884	0.000
Model for PP_2	Coefficient	0.124	0.006	22.102	0.000
	Intercept	6.063	0.668	9.070	0.000

Table 15: The results of the OLS regression analysis for training set no. 7

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.299 + 0.220 PP_1	0.939	0.881	3.347	468.486	0.000
PP_2	EFD = 5.970 + 0.121 PP_2	0.939	0.882	3.338	471.409	0.000

(b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.220	0.010	21.644	0.000
	Intercept	4.299	0.723	5.949	0.000
Model for PP_2	Coefficient	0.121	0.006	21.712	0.000
	Intercept	5.970	0.659	9.053	0.000

Table 16: The results of the OLS regression analysis for training set no. 8

(a)

	Prediction model	**R**	**R^2**	**Std. Err**	**F**	**Significant F**
PP_1	EFD = 4.648 + 0.221 PP_1	0.935	0.874	3.442	452.336	0.000
PP_2	EFD = 6.265 + 0.121 PP_2	0.938	0.881	3.303	479.402	0.000

(b)

		Value	**Std. Err**	**t-value**	**p-value**
Model for PP_1	Coefficient	0.221	0.010	21.268	0.000
	Intercept	4.648	0.723	6.429	0.000
Model for PP_2	Coefficient	0.121	0.006	21.895	0.000
	Intercept	6.265	0.637	9.834	0.000

Table 17: The mean and median R^2 values for PP_1 and PP_2

	Mean R^2	**Median R^2**
PP_1	0.881	0.881
PP_2	0.879	0.884

5.7 Accuracy Evaluation of the Prediction Models

In order to assess the acceptability of the effort prediction models, the criteria suggested by Conte et al. [31] were adopted. In particular, the author applied the *Magnitude of Relative Error*, which is defined as

$$MRE = |EFD_{real} - EFD_{pred}| / EFD_{real}, \quad (5.3)$$

where EFD_{real} and EFD_{pred} are the actual and predicted efforts, respectively. The rationale behind this measure is that the gravity of the absolute error is proportional to the size of the observations. Such value has been calculated for each of the 10 observations in any test set, using the models derived for both PP_1 and PP_2. For each test set, the prediction accuracy has been evaluated by taking into account a summary measure, given by the *Mean of MRE (MMRE)*, to measure the aggregation of *MRE* over the 10 observations.

An acceptable threshold for an effort prediction model, as suggested by Conte et al is an *MMRE* value ≤ 0.25. The values of such measures are reported in Tables 18 to 25. It can be observed that the model derived from training sets 2 and 4 for both PP_1 and PP_2 exhibit *MMRE* values greater than 0.25; in addition, the model derived from training sets 3 and 5 exhibit an *MMRE* value greater than 0.25 for in PP_1. All other models satisfy that condition. This represents an acceptable threshold for an effort prediction model, as suggested by Conte et al [31], which is confirmed by the aggregate (mean) and median *MMRE* values for PP_1 and PP_2 in Table 17, which are both ≤ 0.25.

Another meaningful measure of accuracy namely, the *prediction at level l,* was assessed. The *prediction at level l* is defined as

$$PRED\ (l) = k/N \tag{5.4}$$

where k is the number of observations whose *MRE* is less than or equal to l, and N is the total number of observations. Again, according to Conte et al., at least 75 percent of the predicted values should fall within 25 percent of their actual values. In other words, a good effort prediction model should have PRED (0.25) ≥ 0.75. As we can observe from the results shown in Tables 9 to 16, the required condition is satisfied for most of the derived models. PP_2 clearly outperforms PP_1. The exceptions where *PRED* (0.25) < 0.75 lie in the models derived from training set 2 and 4 in both PP_1 and PP_2, and in the models derived from training sets 1 and 3 for PP_1.

Once accuracy has been separately calculated for each test set, the resulting values have been aggregated across all eight sets. Table 26 reports the results of such analysis. The aggregate *MMRE* and aggregate *PRED* (0.25) suggests that PP_2 is good for estimating the development effort but PP_1 falls just short at a mean *PRED* (0.25) value of 0.71. PP_2 exhibits a better performance, thus confirming our intuition that the *PC* metric may contribute, together with the *DD* and *SC* measures, to predict the development effort of object-oriented systems. Nevertheless, the knowledge of the # of *Pattern Concrete* classes may not be very accurate early in the development process, whereas the *DD* and *SC* metrics are usually available earlier than the *PC* metric. This suggests the use of the PP_1 measure at the beginning of the development process, in order to obtain a preliminary effort estimation, which can be refined by employing PP_2 when the number of *Pattern Concrete* classes is known. As a matter of fact, PP_2 is

strongly correlated to PP_1 (Figure 10), as shown by the OLS regression carried out on the 78 projects.

Table 18: The validation results for test set 1

	EFD_{real}	EFD = 4.486 + 0.223 PP_1			EFD = 5.84 + 0.129 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	13	25.938	11.845	0.089	34.584	11.519	0.114
2	4	0	4.486	0.122	0	5.843	0.461
3	20.5	61.308	21.88	0.067	85.674	19.904	0.029
4	39	129.69	41.281	0.058	225.582	42.866	0.099
5	18	63.666	22.549	0.253	125.76	26.483	0.471
6	16	49.518	18.535	0.158	88.032	20.291	0.268
7	28	73.098	25.225	0.099	142.266	29.192	0.043
8	7	11.790	7.831	0.119	14.148	8.165	0.166
9	28	75.456	25.894	0.075	145.41	29.708	0.061
10	13	37.728	15.19	0.168	51.876	14.357	0.104
		MMRE		0.121	**MMRE**		0.182
		PRED (0.25)		0.9	**PRED (0.25)**		0.8

Table 19: The validation results for test set 2

	EFD_{real}	EFD = 4.647 + 0.219 PP_1			EFD = 5.873 + 0.125 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	35	97.464	31.679	0.095	220.080	40.873	0.168
2	46	70.740	24.267	0.472	110.040	23.373	0.492
3	4	11.790	7.917	0.979	14.148	8.123	1.031
4	15	35.370	14.457	0.036	46.374	13.248	0.117
5	4	11.790	7.917	0.979	14.148	8.123	1.031
6	24	61.308	21.651	0.098	88.032	19.873	0.172
7	22	58.950	20.997	0.046	82.530	18.998	0.136
8	13	35.370	14.457	0.112	46.374	13.248	0.019
9	18	37.728	15.111	0.161	43.230	12.748	0.292
10	10	11.790	7.917	0.208	14.148	8.123	0.188
		MMRE		0.319	**MMRE**		0.365
		PRED (0.25)		0.7	**PRED (0.25)**		0.6

Table 20: The validation results for test set 3

	EFD_{real}	EFD = 4.13 + 0.225 PP_1			EFD = 5.709 + 0.127 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	4	11.790	7.505	0.876	12.576	7.741	0.935
2	7	0.000	4.130	0.410	0.000	5.709	0.184
3	9	11.790	7.505	0.166	12.576	7.741	0.140
4	17	54.234	19.655	0.156	89.604	20.187	0.187
5	22	68.382	23.705	0.078	91.962	20.568	0.065
6	10	11.790	7.505	0.250	14.148	7.995	0.201
7	30.5	75.456	25.730	0.156	166.632	32.633	0.070
8	7	0.000	4.130	0.410	0.000	5.709	0.184
9	27	47.160	17.630	0.347	96.678	21.330	0.210
10	18	48.732	18.080	0.004	55.020	14.599	0.189
		MMRE		0.285	**MMRE**		0.237
		PRED (0.25)		0.6	**PRED (0.25)**		0.9

Table 21: The validation results for test set 4

	EFD_{real}	EFD = 4.877 + 0.221 PP_1			EFD = 6.512 + 0.124 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	10	35.370	14.822	0.482	38.514	12.588	0.2588
2	55	98.250	32.502	0.409	159.558	31.684	0.424
3	31	100.608	33.165	0.070	152.484	30.568	0.014
4	7	30.654	13.496	0.928	33.798	11.844	0.692
5	4	11.790	8.192	1.048	14.148	8.744	1.186
6	17	58.950	21.452	0.262	84.888	19.904	0.171
7	18	37.728	15.485	0.140	51.876	14.696	0.184
8	7	11.790	8.192	0.170	12.576	8.496	0.214
9	28.5	113.970	36.922	0.296	168.99	33.172	0.164
10	3	0.000	4.877	0.626	0	6.512	1.171
		MMRE		0.443	**MMRE**		0.448
		PRED (0.25)		0.3	**PRED (0.25)**		0.6

Table 22: The validation results for test set 5

	EFD_{real}	EFD = 4.144 + 0.228 PP_1			EFD = 5.572 + 0.129 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	20	47.160	17.824	0.109	65.238	16.279	0.186
2	13	0.000	4.144	0.681	0.000	5.572	0.571
3	22	68.382	23.980	0.090	132.048	27.244	0.238
4	16	37.728	15.088	0.057	51.876	14.086	0.120
5	39	122.616	39.712	0.018	215.364	40.918	0.049
6	31	80.958	27.628	0.109	146.982	29.695	0.042
7	20	51.876	19.192	0.040	63.666	16.021	0.199
8	17	61.308	21.928	0.290	85.674	19.633	0.155
9	7	37.728	15.088	1.155	47.160	13.312	0.902
10	6	0.000	4.144	0.623	0.000	5.572	0.071
		MMRE		0.317	**MMRE**		0.253
		PRED (0.25)		0.6	**PRED (0.25)**		0.8

Table 23: The validation results for test set 6

	EFD_{real}	$EFD = 4.343 + 0.226\ PP_1$			$EFD = 6.063 + 0.124\ PP_2$		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	23	60.522	21.745	0.055	86.460	19.703	0.143
2	19.5	63.666	22.649	0.161	102.180	22.183	0.138
3	8	0.000	4.343	0.457	0.000	6.063	0.242
4	16	51.876	19.259	0.204	78.600	18.463	0.154
5	21	75.456	26.039	0.240	116.328	24.415	0.163
6	21	58.950	21.293	0.014	111.612	23.671	0.127
7	7	11.790	7.733	0.105	14.148	8.295	0.185
8	24	68.382	24.005	0.000208	92.748	20.695	0.138
9	27	71.526	24.909	0.077	133.620	27.143	0.005
10	19	48.732	18.355	0.034	55.020	14.743	0.224
		MMRE		0.135	**MMRE**		0.152
		PRED (0.25)		0.9	**PRED (0.25)**		1

Table 24: The validation results for test set 7

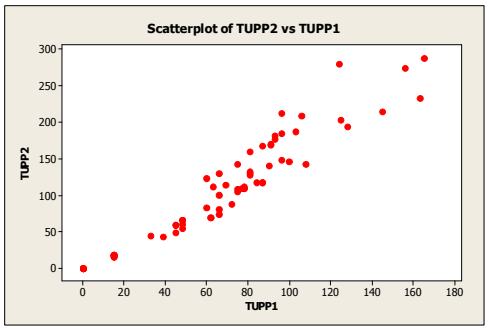
	EFD_{real}	EFD = 4.299 + 0.220 PP_1			EFD = 5.970 + 0.121 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	27	84.888	28.059	0.039	111.612	23.152	0.143
2	10	11.790	7.599	0.240	14.148	8.148	0.185
3	7	0.000	4.299	0.386	0.000	5.970	0.147
4	31	83.316	27.619	0.109	164.274	31.259	0.008
5	20	56.592	20.139	0.007	69.168	16.618	0.169
6	36	73.098	24.759	0.312	139.122	27.387	0.239
7	5	0.000	4.299	0.140	0.000	5.970	0.194
8	4	0.000	4.299	0.075	0.000	5.970	0.493
9	25	66.024	22.779	0.089	91.962	20.127	0.195
10	28	71.526	24.319	0.131	132.834	26.419	0.056
		MMRE		0.153	**MMRE**		0.183
		PRED (0.25)		0.8	**PRED (0.25)**		1

Table 25: The validation results for test set 8

	EFD_{real}	EFD = 4.648 + 0.221 PP_1			EFD = 6.265 + 0.121 PP_2		
		PP_1	**EFD_{pred}**	**MRE**	**PP_2**	**EFD_{pred}**	**MRE**
1	22	63.666	22.549	0.025	100.608	21.758	0.011
2	25	63.666	22.549	0.098	103.752	22.242	0.110
3	0	11.790	7.963	0.000	13.362	8.327	0.000
4	15	51.876	19.234	0.282	58.164	15.224	0.015
5	5	0.000	4.648	0.070	0.000	6.270	0.254
6	22	51.876	19.234	0.126	102.180	22.000	0.000
7	39	128.118	40.671	0.043	183.138	34.463	0.116
8	27	78.600	26.748	0.009	114.756	23.936	0.113
		MMRE		0.082	**MMRE**		0.078
		PRED (0.25)		0.875	**PRED (0.25)**		1

Table 26: Aggregate accuracy evaluation

	Aggregrate *MMRE*	**Aggregrate *PRED (0.25)***
PP_1	0.232	0.709
PP_2	0.237	0.874



R^2	R	Std Err	F	Significant F
0.931	0.965	20.178	1019.174	0.000

Figure 10: Results of the OLS regression analysis with PP_1 as independent variable and PP_2 as dependent variable

6. COMPARISON ANALYSIS

6.1 Single Measures and Their Sums

Courtney et al. [71] report that researchers who set out to learn empirical relationships by experimenting with different combinations of measures and functional forms before choosing the one with the highest correlation tend to make a good model with small data sets. A comparative study was conducted with respect to each of the single measures employed in the *Pattern Point* approach (i.e., *DD*, *SC*, and *PC*), and with respect to the measures obtained by summing them. The results of the study are presented in this section. An 8-fold cross validation technique similar to that used in the empirical validation process of the PP_1 and PP_2 measures was again used in this study. An Ordinary Least Squares regression was carried out on the training sets after removing the influential outliers. Then, the performance of the derived models for all considered measures was evaluated using the data coming from the corresponding testing sets. Table 27 shows a summary descriptive statistics of the measures considered. We recall that data about *DD*, *SC*, and *PC* for the 78 use cases are listed in Table 6. For the sake of simplicity, only the aggregate *MMRE* and *PRED* (0.25) resulting from the cross validation, for each of the considered measures is shown in Table 28.

Among the single measures, the best performance is gained by *SC+PC*, for which we have acceptable *MMRE* values, and *PRED* (0.25) values that surpass the indicated threshold. An acceptable *MMRE* value and corresponding *PRED* (0.25) value also results for *SC*, *DD+SC*, *DD+SC+PC*, which shows that the *SC* metric is very strongly correlated with effort. In fact, all the measures with *SC* fair slightly better than

the PP_1 metric. Even though the *SC+PC* and *DD+SC+PC* measures are expected to perform better because the *PC* metric is a part of these measures, *SC* by itself also faired slightly better: an aggregate *PRED (0.25)* of 0.75 VS an aggregate *PRED (0.25)* of 0.71 for PP_1. This leads to the conclusion that the *SC* metric can be used interchangeably with the PP_1 metric at the early analysis stage. The PP_2 metric outperformed all the other measures.

Table 27: Descriptive statistics of the measures considered for the comparison analysis

	Obs.	**Min**	**Max**	**Mean**	**Median**	**Mode**	**Std Dev.**
DD	78	0	100	37.641	41	11	25.265
SC	78	0	147	51.436	53	31	37.374
PC	78	0	122	33.397	24.5	0	33.880
DD + SC	78	0	244	88.910	92.5	24	62.332
DD + PC	78	0	219	70.872	68	0	56.647
SC + PC	78	0	269	84.833	83	0	68.750
DD + SC + PC	78	0	366	122.308	127.5	0	92.961

Table 28: Aggregate accuracy evaluation of the prediction models derived from basic and combined size measures

	Aggregate MMRE	**Aggregate PRED (0.25)**
DD	0.37	0.23
SC	0.23	0.75
PC	0.36	0.56
DD+SC	0.23	0.74
DD+PC	0.26	0.79
SC+PC	0.25	0.80
DD+SC+PC	0.24	0.78

Comparing the above results with the ones of Section 6.7, we may derive two main conclusions. First, the PP_2 metric is better correlated to effort than any single measure composing it. Second, the mere sum of the single measures is sufficient to enhance the performance of its prediction model as can been seen in *DD+PC*. Lastly, the PP_1 metric can be used interchangeably with *SC* metric at the early analysis stage.

6.2 Multivariate OLS Regression

In order to complete the analysis, a multivariate OLS regression using as independent variables the basic measures of the *Pattern Point* approach, was carried out. Again, the 8-fold cross validation technique was applied by carrying out a multivariate OLS regression on the eight training sets, and then evaluating the performance of the derived

models, using the data coming from the corresponding testing sets. Table 29 reports the aggregate *MMRE* and *PRED (0.25)* resulting from this analysis. Compared with the values reported in Table 26, it can be deduced that the PP_2 measure exhibits a more accurate predictive capability. In any case, this study has confirmed once again that the use of the PP_2 measure may yield a better predictive accuracy in models, which are based on a multivariate regression as well.

Table 29: Aggregate accuracy evaluation of the prediction models derived from multivariate OLS regression analyses

	Aggregate MMRE	**Aggregate PRED (0.25)**
Multivariate DD_SC	0.24	0.74
Multivariate DD_SC_PC	0.23	0.78

7. CONCLUSIONS AND FUTURE EXTENSIONS

7.1 Conclusions

There are several models in existence that are used to estimate the size of software systems. System-level measures are especially important for project managers who could benefit from an overall view of the system [32]. As a matter of fact, activities such as software development planning, and particularly the tasks of estimating cost and effort, are more effectively performed when a size estimate of the whole system is available.

The object-oriented development paradigm warrants a re-thinking of the way estimation models are contrived because of the unique characteristic of the OO paradigm; namely, that the same artifacts are systematically realized and refined. The definition of a common, structured modeling framework like OMG SysML and the availability of the artifacts in the CASE tools present an opportunity for a holistic modeling approach that can leverage these artifacts. The SysML Point model was presented to take advantage of these factors. Of the estimation techniques that constitute the SysML Point approach, only the Pattern Point model was yet to be defined.

Among system level measures, the Function Point count has achieved international acceptance as a size estimate of business systems and in predicting the effort, cost and duration of projects [30], [33], [34]. The methodology provides an estimate of software size by measuring the functionality of the system to be developed. Despite the fact that the FP method was originally conceived to be independent of the methodology used to develop the system under measurement, the application of the *FP* method turns out to be rather unnatural when applied to object-oriented systems [34, 35].

Kemerer and Porter concluded their empirical study on improving the reliability of *FP* measurement with the statement: "The advent of event driven, object-oriented systems ... may require redefinition of *FP*'s or the development of one or several new measures to identify system size" [34].

Many researchers are in agreement that the *FP* method can be generalized in order to be successfully used for other types of systems (e.g. engineering, scientific and real-time systems) and for different programming paradigms [17], [33], [36]-[39]. This is the case as seen in Object-oriented Function Point [10], Use Case Point [9] and Class Point models [29] in estimating object-oriented projects. Verner and Tate suggest a general *FP*-like model for a more objective and accurate size estimation, which can be tailored to any specific software development environment [117].

The *Pattern Point* approach reflects the main features of the *FP*-like general model proposed by Verner and Tate, namely, the partitioning of a system into different component types with different sizing criteria for each type, the sizing of the individual components, the sum of the component sizes and an overall system adjustment to allow for global factors. The proposed partitioning of the design patterns into different types, which are sized with different rules is not tailored to a specific application environment and this provides a high level of flexibility of the method.

The *Pattern Point* model provides a system-level size measure using the design patterns from object interaction analyses in the late OOA phase of development. Two measures are defined within the *Pattern Point* method; these are the PP_1 and PP_2 metrics. PP_1 is useful as a size measure earlier in than PP_2 because it does not require

the number of *pattern concrete (PC)* classes metric, which is typically available later in OOA.

The empirical study shows that *Pattern Point* measure can be effectively used during the OOA phase to predict the effort values with a high degree of confidence. In particular, the PP_2 measure outperformed PP_1, supporting the intuition that the *PC* measure can be profitably exploited in the estimation of system size. The empirical study presented in the dissertation has suggested that the PP_1 measure may have an equal or lesser predictive capability than its constituent *SC* metric. Moreover, the proposed aggregation and multivariate models turns out to be quite effective; however the PP_2 measure outperformed all others measures in the comparison analyses.

In conclusion, further investigation is needed for assessment of the *Pattern Point* method. A preliminary empirical evaluation, based on data coming from 78 use cases, developed in the IBM Lotus Quickr 8.0 release prove that a model based on the design patterns from object interaction modeling is effective in estimating size and remaining development effort. However, a multi-project study is desired to assess the possible effects of the *Technical Complexity Factors* and *Environmental Factors* in the *Pattern Point* method.

7.2 Future Extensions

In the SysML Point approach, since a Pattern Point Model has been defined, the next step would be for an analysis of the complete methodology to be conducted. It would be particularly beneficial to perform an empirical study in an iterative development project environment such as the Unified Process [99] where OO artifacts would likely be at different levels of realizations.

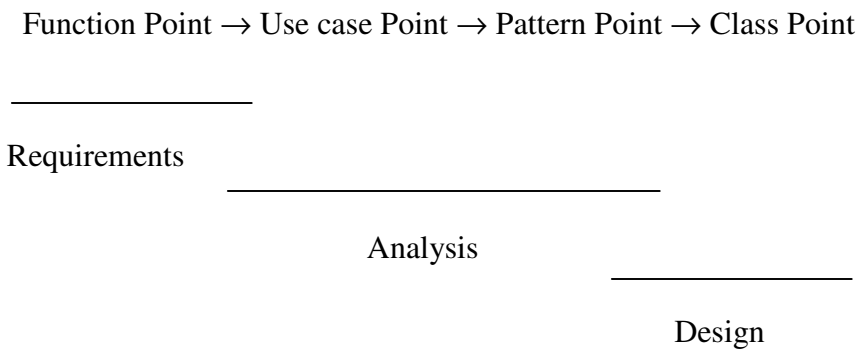


Figure 11: Object-oriented development stages and corresponding effort estimation models

Also of interest is the development of a conversion mechanism from one metric to another. For example, converting Function Points to Pattern Points and vice versa. This might not be completely necessary because each of the measures already have a conversion to a time value such as hours, days or months.

A software architectural pattern expresses a fundamental structural organization schema for a software system, which consists of subsystems, their responsibilities and interrelations. In comparison to design patterns, architectural patterns are larger in scale

and operate at a higher level [103]. Examples include, Model-view-controller, Peer-to-peer and Presentation and Presentation-abstraction-control. A study on the possible effects of these architectural patterns on size and effort estimates would be of interest. However since they operate at a much higher level they are more likely to be included as a Technical Complexity Factor (TCF) or Environmental Adjustment Factor (EAF).

REFERENCES

[1] R. Smith, "Panel on Design Methodology," in *OOPSLA '87*. OOPSLA '87 Addendum to the Proceedings. doi:10.1145/62138.62151, 1987.

[2] K. Beck and W. Cunningham, "Using Pattern Languages for Object-Oriented Program," in *OOPSLA '87*. OOPSLA '87 Workshop on Specification and Design for Object-Oriented Programming.

[3] E. Gamma, R. Helm, R. Johnson and J.M. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," 1994.

[4] M. Grand, "Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML," 1998.

[5] L.C. Briand, S. Morasca, and V.R. Basili, "Property Based Software Engineering Measurement," *IEEE Trans. Software Eng*., vol. 22, no. 1, pp. 68-86, Jan. 1996.

[6] N. Fenton, "Software Measurement: A Necessary Scientific Basis," *IEEE Trans. Software Eng*., vol. 20, no. 3, pp. 199-206, Mar. 1994.

[7] H. Zuse, "Reply to: Property-Based Software Engineering Measurement," *IEEE Trans. Software Eng*., vol. 23, p. 533 Aug. 1997.

[8] H. Zuse, *Software Complexity: Measures and Methods*: Walter de Gruyter, 1990.

[9] E. R. Carroll, *Estimating Software Based on Use Case Points*, OOPSLA, ACM, 2005.

[10] G. Caldiera, G. Antoniol, R. Fiutem and C. Lokan, "Definition and Experimental valuation of Function Points for Object-Oriented Systems," *5th Software Metrics Symposium*, 1998.

[11] C. Gennaro, F. Filomena, T. Genoveffa and V. Guiliana, "Class Point: An Approach for the Size Estimation of Object-Oriented Systems." *IEEE Transactions on Software Engineering*, vol. 31, no. 1, January 2005.

[12] L. Briand, K. El Emam, and S. Morasca, "On the Application of Measurement Theory in Software Engineering," *J. Empirical Software Eng*., vol. 1, pp. 61-68, 1996.

[13] B.A. Kitchenham, N. Fenton, and S.L. Pfleeger, "Towards a Framework for Software Measurement Validation," *IEEE Trans. Software Eng*., vol. 21, no. 12, pp. 929-944, Dec. 1995.

[14] K.B. Lakshmanan, S. Jayaprakash, and P.K. Sinha, "Properties of Control-Flow Complexity Measures," *IEEE Trans. Software Eng*., vol. 17, pp. 1289-1295, Dec. 1991.

[15] E.J. Weyuker, "Evaluating Software Complexity Measures," *IEEE Trans. Software Eng*., vol. 14, pp. 1357-1365, Sept. 1988.

[16] G. Booch, *Object-Oriented Analysis and Design*, 2nd Ed. Benjamin/Cummings, 1994.

[17] B.W. Boehm, B. Clark, and E. Hiriwitz, "Cost Models for Future Life Cycle Processes: COCOMO 2. 0," *Ann. Software Eng*., vol. 1, no. 1, pp. 1-24, 1995.

[18] V.B. Misic and D.N. Tesic, "Estimation of Effort and Complexity: an Object-Oriented Case Study," *J. Systems and Software*, vol. 41, pp. 133-143, 1999.

[19] S. Moser, B. Henderson-Sellers, and V.B. Misic, "Cost EstimationBased on Business Models," *J. Systems and Software*, vol. 49, pp. 33-42, 1999.

[20] P. Nesi and T. Querci, "Effort Estimation and Prediction of Object-Oriented Systems," *J. Systems and Software*, vol. 42, pp. 89-102, 1998.

[21] R.D. Banker, R.J. Kauffman, C. Wright, and D. Zweig, "Automating Output Size and Reuse Metrics in a Repository-Based Computer-Aided Software Engineeering (CASE) Environment," *IEEE Trans. Software Eng*., vol. 20, no. 3, pp. 169-187, Mar. 1994.

[22] G.C. Low and D.R. Jeffrey, "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Trans. Software Eng*., vol. 16, pp. 64-71, Jan. 1990.

[23] S. Moser, B. Henderson-Sellers, and V.B. Misic, "Measuring Object-Oriented Business Models," *Proc. TOOL Pacific'97 Conf*., 1997.

[24] L. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. The´venod-Fosse, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of The Art and Future Directions," *J. Empirical Software Eng*., vol. 4, pp. 387-404, Sept. 1999.

[25] F. Brito and E. Abreu, "The MOOD Metrics Set," *Proc. ECOOP'95 Workshop Metrics*, 1995.

[26] F. Brito, E. Abreu, M. Goulao, and R. Estevers, "Toward the Design Quality Evaluation of OO Software Systems," *Proc. Fifth Int'l Conf. Software Quali*ty, 1995.

[27] P. Coad and J. Nicola, *Object-Oriented Programming*. Prentice Hall, 1993.

[28] D. Wampler, "Aspect-Oriented Design in Java/AspectJ and Ruby," - *Companion, 2007. ICSE 2007 Companion. 29th International Conference on Software Engineering*, pp.184-185, 20-26 May 2007.

[29] G. Costagliola, F. Ferrucci, G. Tortora and G. Vitiello, "Class Point: An Approach for the Size Estimation of Object-oriented Systems," *IEEE Transactions on Software Engineering*, vol.31, no.1, pp. 52-74, Jan. 2005.

[30] A.J. Albrecht, "Measuring Application Development Productivity," *Proc. Joint SHARE/GUIDE/IBM Application Development Symp.*, pp. 83-92, 1979.

[31] S.D. Conte, H.E. Dunsmore, and V.Y. Shen, *Software Eng. Metrics and Models*. Menlo Park, Benjamin/Cummings, 1986.

[32] R. Harrison, S.J. Counsell, and R.V. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Trans. Software Eng*., vol. 24, no. 6, pp. 491-496, June 1998.

[33] J.B. Dreger, *Function Point Analysis*. Prentice Hall, 1989.

[34] C.F. Kemerer and B.S. Porter, "Improving the Reliability of Function Point Measurement: An Empirical Study," *IEEE Trans. Software Eng*., vol. 18, no. 11, pp. 1011-1024, Nov. 1992.

[35] J. Keyes, "New Metrics Needed for New Generation: Lines of Code, Function Points Won't Do at the Dawn of the Graphical, Object Era," *IEEE Software*, vol. 12, no. 6, pp. 42-52, 1992.

[36] P. Coad and E. Yourdon, *Object-Oriented Analysis, second ed*. Yourdon Press, 1991.

[37] M. Itakura and A. Takayanagi, "A Model for Estimating Program Size and Its Evaluation," *Proc. Sixth Int'l Conf. Software Eng*., pp. 104-109, 1982.

[38] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[39] I. Sommerville, *Software Engineering*, Addison Wesley, 1996.

[40] J. Lehman, "How Software Projects Are Really Managed," *Datamation*, vol. 3, pp. 119–129, 1979.

[41] Function Point Counting Practices Manual, Release 4.1.1, Int'l Function Point Users Group, 2001.

[42] F. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1995.

[43] B. Boehm, *Software Engineering Economics*. Prentice-Hall, 1981.

[44] H. Rehesaar and E. Beames, "Project Plans and Times Budgets in Information System Projects," *International Conference on Software Engineering: Education & Practice*, pp. 120–124, 1998.

[45] B. Boehm, "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, pp. 73–82, 1996.

[46] D. Phillips, "Project Management: Filling in the Gaps," *IEEE Software*, vol. 13, no. 4, pp. 17–18, 1996.

[47] R. B. Pittman, "Product & Project Planning: Key to Getting It Right the First Time," *IEEE WESCON/96*, pp. 91–95, 1996.

[48] Carnegie Mellon Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.

[49] M. C. Paulk, *Key Practices of the Capability Maturity Model*. Addison-Wesley, 1993.

[50] C. R. Snyder, "The Software Development Plan: A Key to Achieve SEI Capability Maturity Model Compliance," *ACM*, vol. 3, no. 7, pp. 106–112, 1992.

[51] R. C. Tausworthe, "The Work Breakdown Structure in Software Project Management," *Systems and Software*, vol. 81, pp. 181–186, 1980.

[52] D. Simmons, N. Ellis, H. Fujihara, and W. Kuo, *Software Measurement—A Visualization Tool Kit for Process Control and Process Improvement*. Prentice-Hall, 1998.

[53] R. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 1997.

[54] W. S. Humphrey, *Managing the Software Process*. Reading, Addison-Wesley, 1989.

[55] A. Tatnall and P. Shackleton, "IT Project Management: Developing On-going Skills in the Management of Software Development Projects," *Proceedings of Software Engineering: Education and Practice*, pp. 400–405, 1996.

[56] C. Chang, C. Chao, and T. Nguyen, "Software Project Management Net: A New Methodology on Software Management," *Proceedings of International Computer Software and Applications Conference*, pp. 534–539, 1998.

[57] E. Bennatan, *On Time Within Budget: Software Project Management Practices and Techniques*. John Wiley & Sons, 2000.

[58] D. Phillips, *The Software Project Management Handbook, Principles That Work at Work*. IEEE Computer Society, 2000.

[59] W. Keuffel, "People Based Processes: A RADical Concept," *Software Development*, vol. 4, pp. 27–30, 1995.

[60] C. Jones, "Patterns of Large Software Systems: Failure and Success," *IEEE Computer*, vol. 28, no. 3, pp. 86–87, 1995.

[61] C. Jones, "Management Tools and Software Failures and Success," *CrossTalk*, vol. 7, no. 10.

[62] W. Royce, *Software Project Management—A Unified Framework*. Reading: Addison-Wesley, 1998.

[63] F. McGrath, *16 Critical Software Practices for Performance-Based Management*. Software Program Management Network, 1999.

[64] A. Shenhar, "Strategic Project Management: The New Framework," *Portland IEEE International Conference on Management of Engineering and Technology*, pp. 382–386, 1999.

[65] M. D. Rosenau and M. D. Lewin, *Software Project Management, Step by Step*. Lifetime Learning Publications, 1984.

[66] F. J. Heemstra, "Software Cost Estimation," *Information on Software Technology*, vol. 34, no. 10, pp. 627–639, 1992.

[67] A. L. Lederer and J. Prasad, "Information System Cost Estimating: A Current Assessment," *Journal of Information*, vol. 8, no. 1, pp. 22–33, 1993.

[68] A. Cuelenaere, M. van Genuchten, and F. Heemstra, "Calibrating a Software Cost Estimation Model: Why and How," *Information and Software Technology*, vol. 29, pp. 558–567, 1994.

[69] C. Walston and C. Felix, "A Method of Programming Measurement and Estimation," *IBM System Journal*, vol. 16, no. 1, pp. 54–73, 1977.

[70] J. Bailey and V. Basili, "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*, pp. 107–116, 1981.

[71] R. Courtney and D. Gustafson, "Shotgun Correlations in Software Measures," *Software Engineering Journal*, vol. 8, no. 1, pp. 5–13, 1993.

[72] S. Conte, H. Dunsmore, and V. Shen, *Software Engineering Metrics and Models*. Benjamin Cummings, 1986.

[73] Y. Miyazaki and K. Mori, "COCOMO Evaluation and Tailoring," *Proceedings of the Eighth International Conference on Software Engineering*, pp. 292–299, 1985.

[74] C. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, 1987.

[75] C. Kemerer and M. Patrick, *Staffing Factor in Software Cost Estimation Models*. Windcrest/McGraw-Hill, 1993.

[76] D. Jeffrey, "Time-Sensitive Cost Models in the Commercial MIS Environment," *IEEE Transactions on Software Engineering*, vol. 13, no. 7, pp. 852–859, 1987.

[77] B. Kitchenham and N. Taylor, "Software Cost Models," *ICL Technology Journal*, vol. 4, no. 3, pp. 73–102, 1984.

[78] L. Briand, V. Basili, and W. Thomas, "A Pattern Recognition Approach for Software Engineering Data Analysis," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 931–942, 1992.

[79] K. Srinivasan and D. Fisher, "Machine Learning Approach to Estimating Development Effort," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 126–137, 1995.

[80] B. Kitchenham and K. Kansala, "Inter-Item Correlation Among Function Points," *Proceedings of the 15th International Conference on Software Engineering*, pp. 477–480, 1993.

[81] G. Wittig and G. Finnie, "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort," *Australian Journal of Information Systems*, vol. 1, no. 2, pp. 87–94, 1994.

[82] B. Samson, D. Ellison, and P. Dugard, "Software Cost Estimation Using an Albus Perception (CMAC)," *Journal of Systems Software*, vol. 12, pp. 209–218, 1997.

[83] A. Cuelenaere, M. V. Genuchten, and F. Heemstra, "Calibrating a Software Cost Estimation Model: Why and How," *Information and Software Technology*, vol. 29, pp. 558–567, 1994.

[84] F. Walkerden and R. Jeffery, "Software Cost Estimation: A Review of Models, Process, and Practice," *Advances in Computers*, vol. 44, pp. 59–125, 1997.

[85] T. Mukhopadhyav, S. Vicinanza, and M. Prietula, "Estimating the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation," *MIS Quarterly*, vol. 16, pp. 155–171, 1992.

[86] K. Atkison and M. Sheppered, "The Use of Function Points to Find Cost Analogies," *Proceedings in European Software Cost Modelling Conference*, 1994.

[87] K. Sengupta and T. Abdel-Hamid, "Impact of Schedule Estimation on Software Project Behavior," *IEEE Software*, vol. 3, no. 4, pp. 70–75, 1986.

[88] T. Abdel-Hamid and S. Madnick, *Software Project Dynamics: An Integrated Approach*. Prentice Hall, 1991.

[89] T. Abdel-Hamid, "Adapting, Correcting, and Perfecting Software Estimates: A Maintenance Metaphor," *Computer*, vol. 26, no. 3, pp. 20–29, 1993.

[90] B. Boehm, C. Abts, A. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*. Prentice-Hall, 2000.

[91] G. Finnie and G. Wittig, "A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models," *Journal of Systems Software*, vol. 39, pp. 281–289,1997.

[92] T. DeMarco and T. Lister, *Peopleware*. Dorset House, 1987.

[93] L. Putnam, “A General Empirical Solution to the Macro Software Sizing and Estimating Problem,” *IEEE Transactions on Software Engineering*, vol. 4, no. 4, pp. 345–361, 1978.

[94] A. Gelman, J. Carlin, H. Stern, and D. Rubin, *Bayesian Data Analysis*. Chapman & Hall, 1995.

[95] B. Kitchenham, “Empirical Studies of Assumptions That Underlie Software Cost-Estimation Models,” *Information and Software Technology*, vol. 34, no. 4, pp. 211–218, 1992.

[96] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, 1996.

[97] I. Jacobson, *Object-Oriented Software Engineering*. Addison Wesley Professional. 1992.

[98] OMG SysML Specification v. 1.0. September 2007.

[99] P. Kruchten, *The Rational Unified Process: An Introduction* (3rd Ed.). Publisher, 2004.

[100] K. Schwaber, *Agile Project Management with Scrum*, Microsoft Press, January 2004.

[101] M. Stephens and D. Rosenberg, *Extreme Programming Refactored: The Case Against XP*, Apress, 2003.

[102] W. Royce, "Managing the Development of Large Software Systems," *Proceedings of IEEE WESCON* 26 (August): 1-9, 1970.

[103] P. Avgeriou and Z. Uwe, "Architectural Patterns Revisited: A Pattern Language." *10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*, 2005.

[104] J. Coutaz, "PAC: an Implementation Model for Dialog Design". H-J. Bullinger, B. Shackel (ed.) *Proceedings of the Interact'87 Conference, September 1-4, 1987, Stuttgart, Germany*: North-Holland, pp. 431-436, 1987.

[105] D. Schoder and K. Fischbach, Core Concepts in Peer-to-Peer (P2P) Networking. In: Subramanian, R.; Goodman, B. (eds.): P2P Computing: The Evolution of a Disruptive Technology, Idea Group Inc, 2005.

[106] A. Leff and J.T. Rayfield, "Web-application Development Using the Model/View/Controller Design Pattern," *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, pp.118-127, 2001.

[107] S. J. Yun (2005). Productivity Prediction Model Based on Bayesian Analysis and Productivity Console. Dissertation, Texas A&M University, 2005.

[108] C. Baudoin and G. Hollowell *Realizing the Object-Oriented Lifecycle*. Prentice Hall, 1996.

[109] S. J. Yun and D.B. Simmons, "Continuous Productivity Assessment and Effort Prediction Based on Bayesian Analysis," *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International* , vol. 1, 28-30, pp. 44-49 Sept. 2004.

[110] D.B. Simmons, "Measuring and Tracking Distributed Software Development Projects," *2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 63-69, 28-30 May 2003.

[111] B. Boehm, et al. *Software Cost Estimation with COCOMO II*. Prentice-Hall, 2000.

[112] B. Clark, S. Devnani-Chulani and B. Boehm, "Calibrating the COCOMO II Post-Architecture Model," *Proceedings of the 1998 International Conference on Software Engineering*, pp.477-480, 19-25 Apr 1998.

[113] Y. Wang and Y. Yuan, "The Formal Economic Model of Software Engineering," *2006.Canadian Conference on Electrical and Computer Engineering,* pp. 2385-2388, May 2006.

[114] J. Li and G. Ruhe, "Decision Support Analysis for Software Effort Estimation by Analogy," *ICSE Workshops 2007. International Workshop on Predictor Models in Software Engineering*, pp.6-6, 20-26 May 2007.

[115] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Transactions on Software Engineering,* vol. 23, no.11, pp.736-743, Nov 1997.

[116] T. Menzies, C. Zhihao, J. Hihn and K. Lum, "Selecting Best Practices for Effort Estimation," *IEEE Transactions on Software Engineering*, vol.32, no.11, pp.883-895, Nov. 2006.

[117] J. Verner and G. Tate, "A Software Size Model," IEEE Trans. Software Eng., vol. 18, pp. 265-278, Apr. 1992.

APPENDIX A

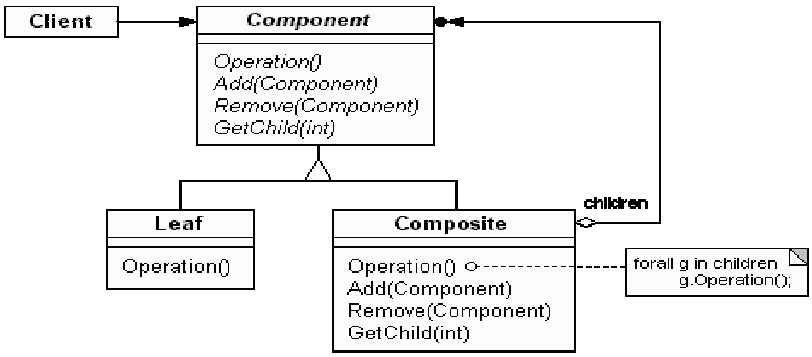


Figure 12: Structural diagram of the *Composite* pattern

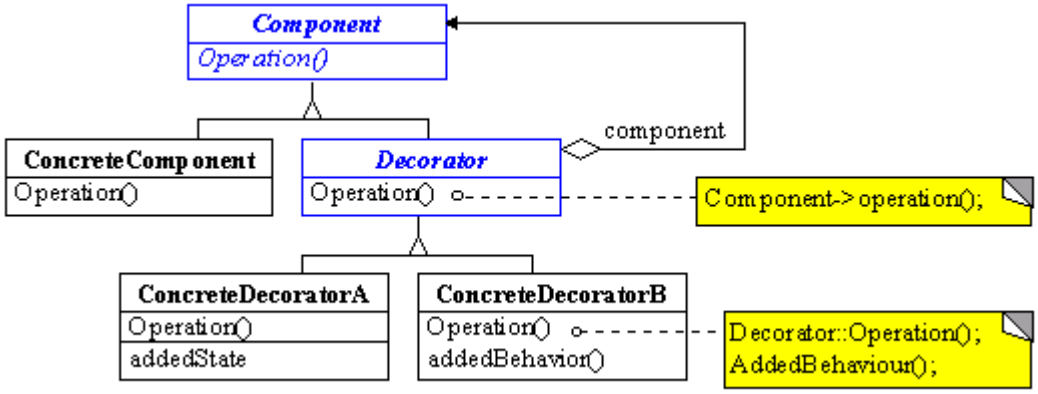


Figure 13: Structural diagram of the *Decorator* pattern

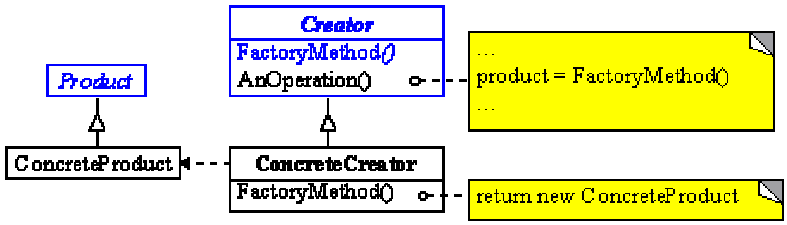


Figure 14: Structural diagram of the *Factory* pattern

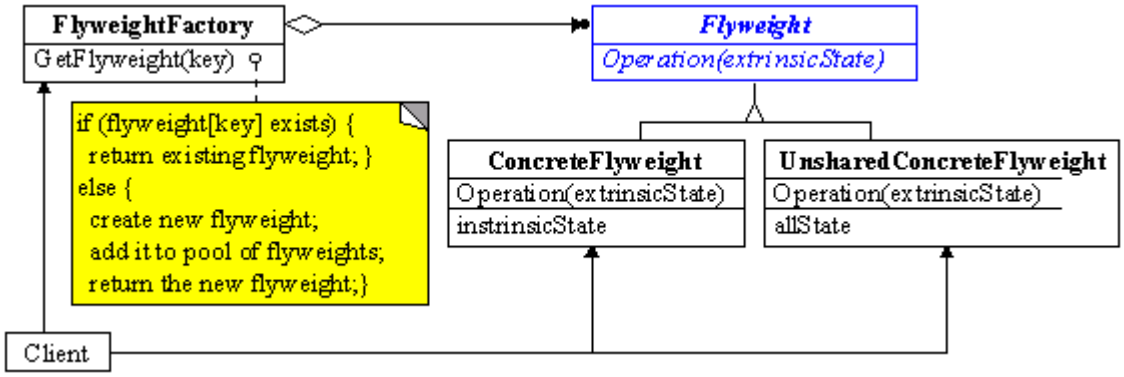


Figure 15: Structural diagram of the *Flyweight* pattern

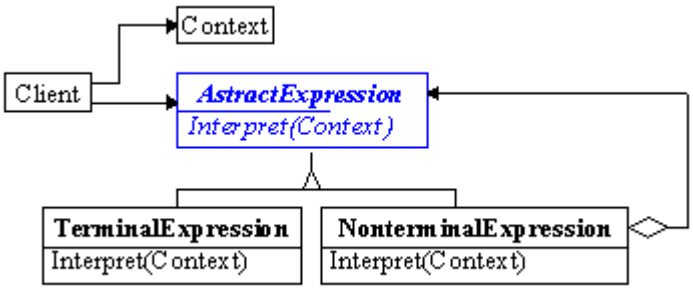


Figure 16: Structural diagram of the *Interpreter* pattern

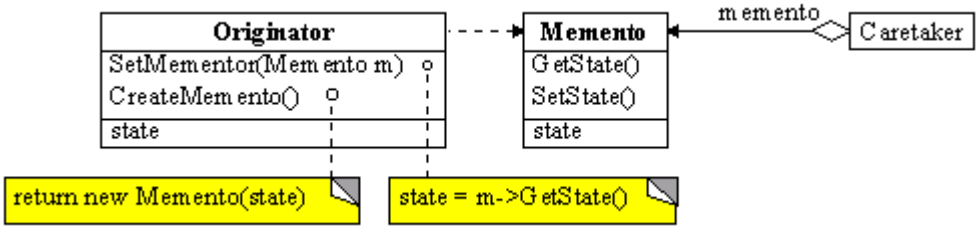


Figure 17: Structural diagram of the *Memento* pattern

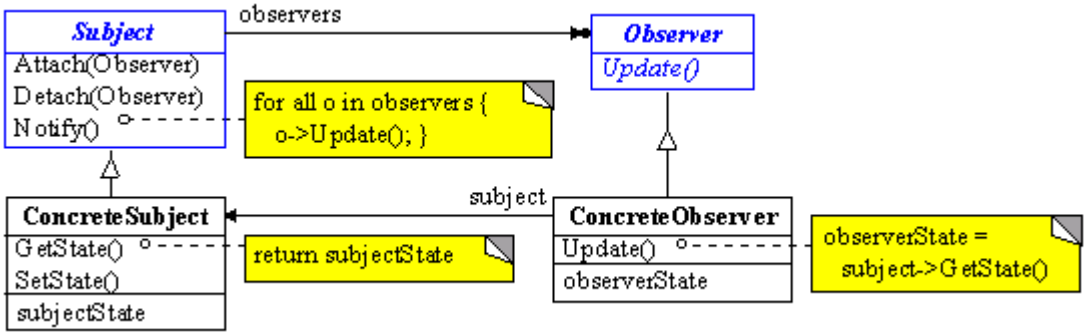


Figure 18: Structural diagram of the *Observer* pattern

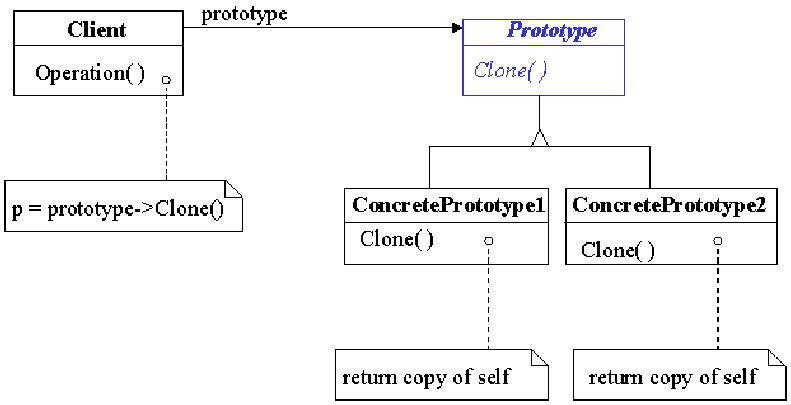


Figure 19: Structural diagram of the *Prototype* pattern

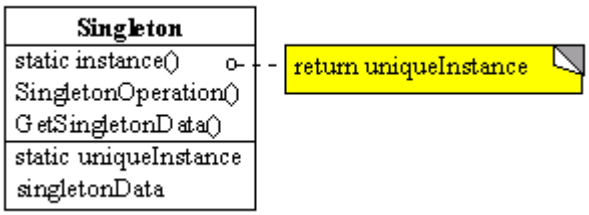


Figure 20: Structural diagram of the *Singleton* pattern

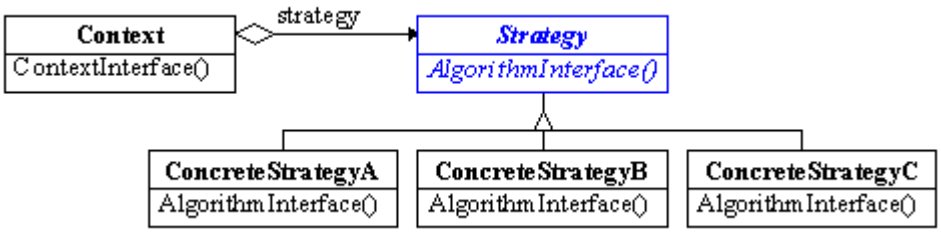


Figure 21: Structural diagram of the *Strategy* pattern

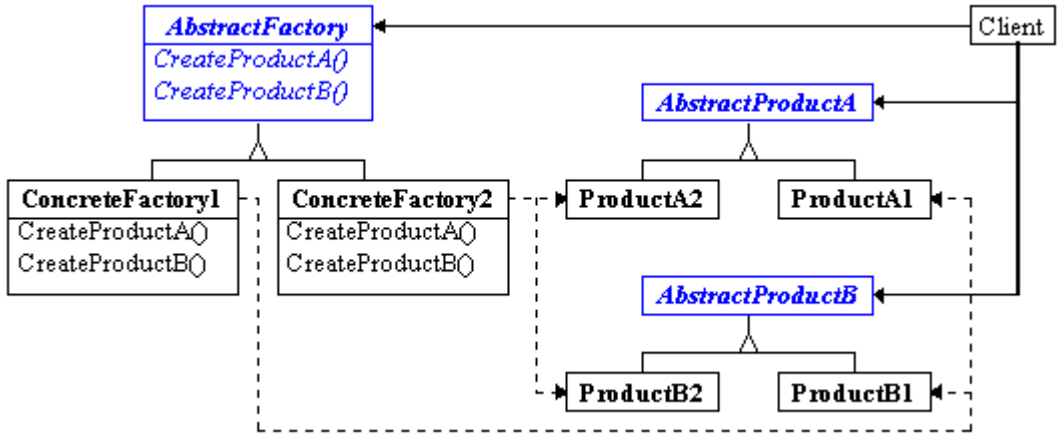


Figure 22: Structural diagram of the *Abstract Factory* pattern

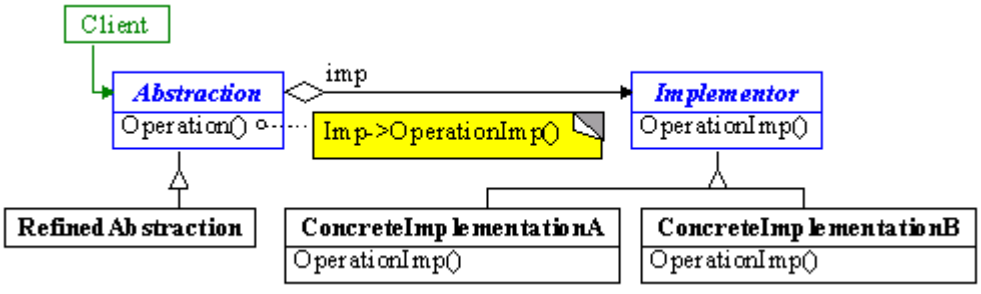


Figure 23: Structural diagram of the *Bridge* pattern

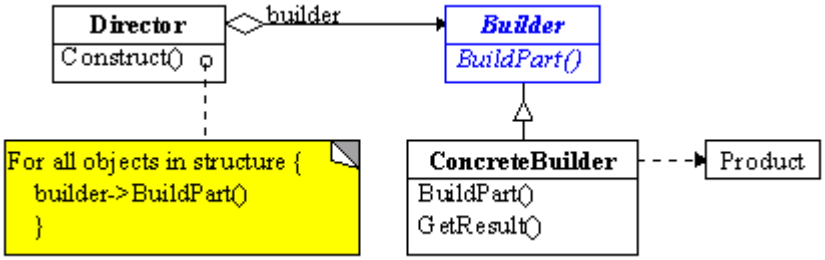


Figure 24: Structural diagram of the *Builder* pattern

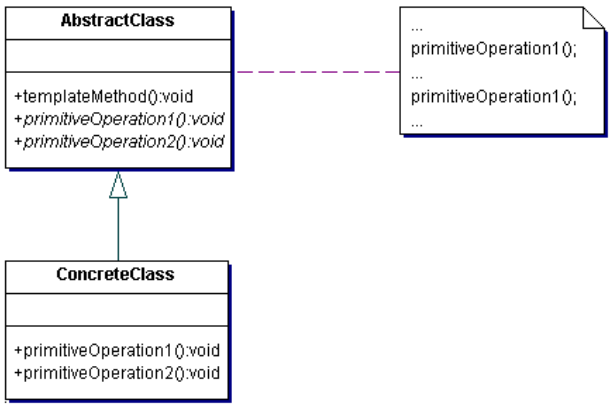


Figure 25: Structural diagram of the *Template Method* pattern

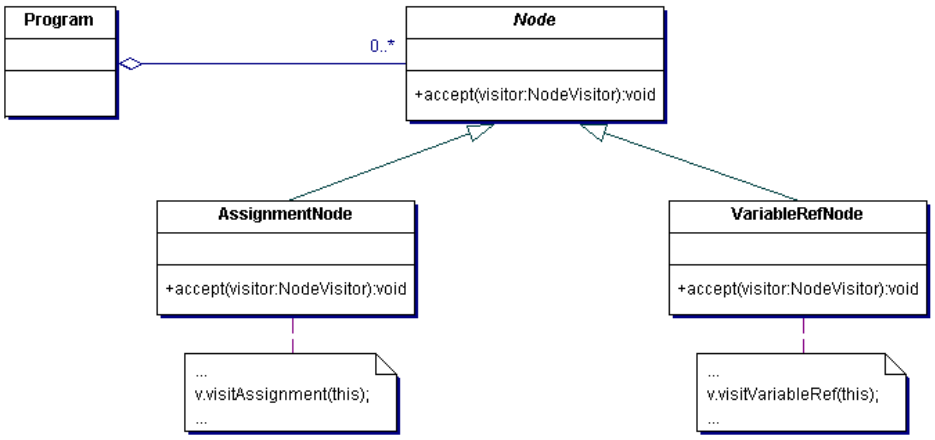


Figure 26: Structural diagram of the *Visitor* pattern

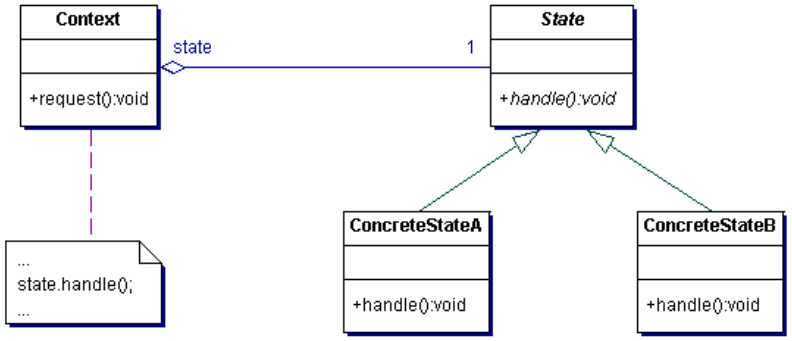


Figure 27: Structural diagram of the *State* pattern

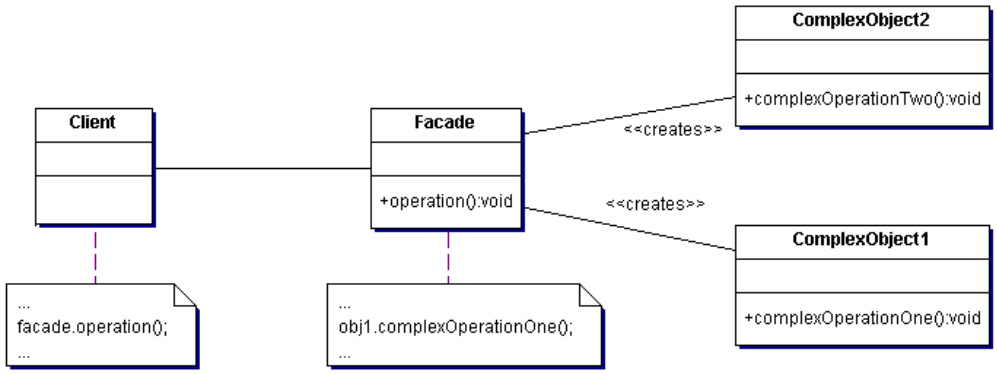


Figure 28: Structural diagram of the *Façade* pattern

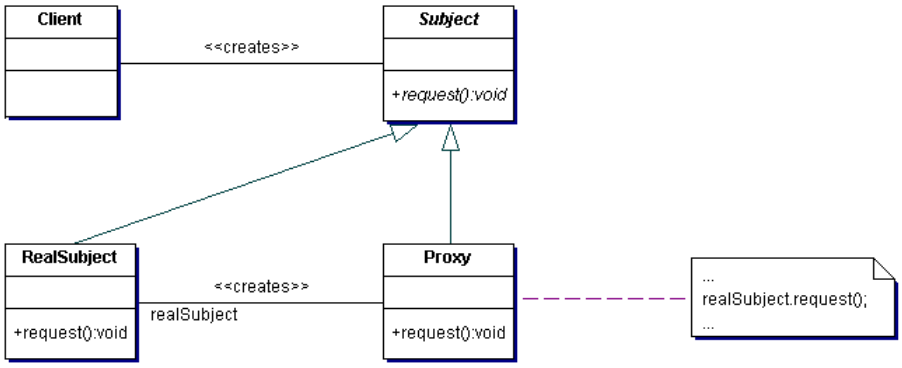


Figure 29: Structural diagram of the *Proxy* pattern

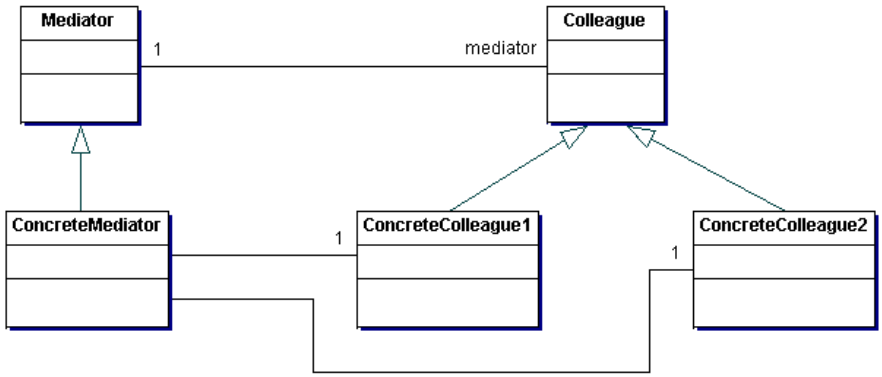


Figure 30: Structural diagram of the *Mediator* pattern

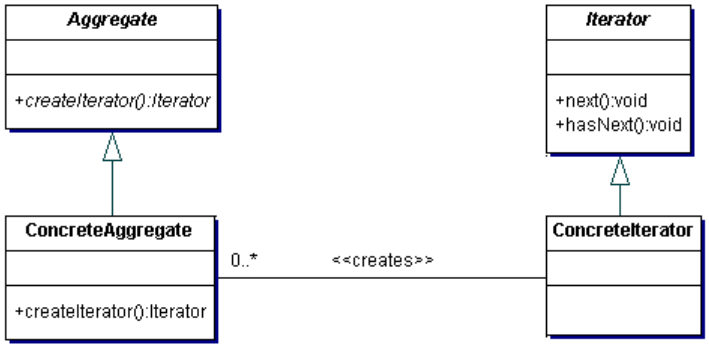


Figure 31: Structural diagram of the *Iterator* pattern

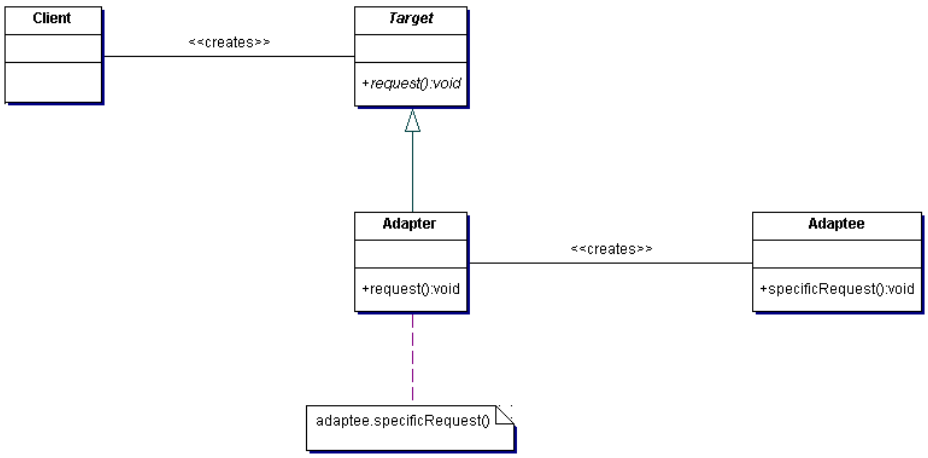


Figure 32: Structural diagram of the *Adapter* pattern

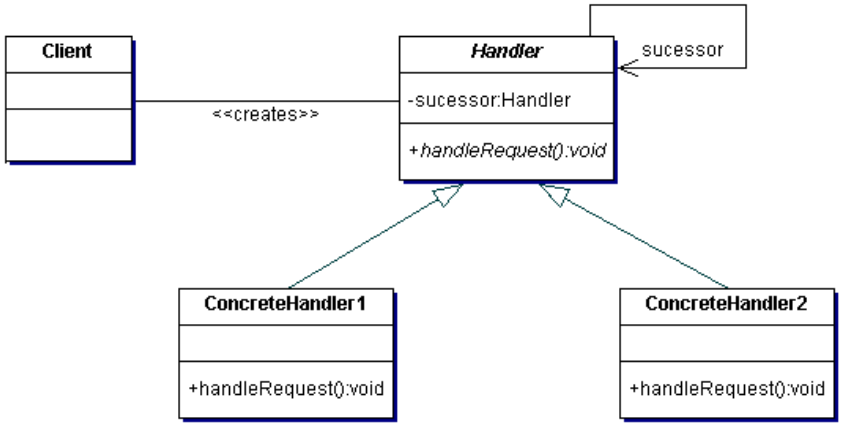


Figure 33: Structural diagram of the *Chain Of Responsibility* pattern

VITA

Olusegun Adekile was born in Ile Ife, Nigeria. He moved to Kuwait with his family while in middle school, where he completed an A-Level education at New English School, Jabriya, Kuwait in 1999. He then traveled to the United States of America to commence a Bachelor of Science in computer science at the University of Georgia, which he attained in 2002. After graduation, he was admitted into a doctoral program at Texas A&M University. This was interrupted periodically by stints in the software industry working for both IBM and Dell Inc. for a year. He received his Ph.D. in computer science in December 2008. He can be reached at 4205 S MIAMI BLVD, DURHAM NC 27703-9141.