7-2008

Timeline prediction framework for iterative software engineering projects with changes

Kay BERKLING
Polytechnic University of Puerto Rico

Georgios KIRAGIANNIS
Polytechnic University of Puerto Rico

Armin ZUNDEL
Polytechnic University of Puerto Rico

Subhajit DATTA
Singapore Management University, subhajitd@smu.edu.sg

Timeline Prediction Framework for Iterative Software Engineering Projects with Changes

Kay Berkling[1], Georgios Kiragiannis[1], Armin Zundel[1], and Subhajit Datta[2]

[1] Polytechnic University of Puerto Rico, Department of Computer Science and Engineering,
377 Ponce de León Ave., San Juan, Puerto Rico 00918
kay@berkling.com, gakirag@gmail.com, azundel@pupr.edu
[2] Department of Computer Science and School of Computational
Science, Florida State University,
Tallahassee, FL 32306, USA
sd05@fsu.edu

Abstract. Even today, software projects still suffer from delays and budget overspending. The causes for this problem are compounded when the project team is distributed across different locations and generally attributed to the decreasing ability to communicate well (due to cultural, linguistic, and physical distance). Many projects, especially those with off-shoring component, consist of small iterations with changes, deletions and additions, yet there is no formal model of the flow of iterations available. A number of commercially available project prediction tools for projects as a whole exist, but the model adaptation process by iteration, if it exists, is unclear. Furthermore, no project data is available publicly to train on and understand the iterative process. In this work, we discuss parameters and formulas that are well founded in the literature and demonstrate their use within a simulation tool. Project timeline prediction capability is demonstrated on various scenarios of change requests. On a real-world example, we show that iteration-based data collection is necessary to train both the parameters and formulas to accurately model the software engineering process to gain a full understanding of complexities in software engineering process.

1 Introduction and Background

Software projects often suffer from delays and budget overspending. With the addition of off-shoring in the software industry, the complexities of such projects have increased. While it is still very difficult to even understand the mechanics of regular projects, taking the next step in complexity to distributed teams, decreases the ability to trace the effects of change requests on the course of the project. Gaining understanding of and control over the timeline and consequently the costs of a project is often accomplished through experience of the project manager. However, without that experience, no comprehensive mathematical model of how the timeline is affected throughout iterations is available to replace that experience. Simulators of such a model would provide a deeper understanding of the parameters and how they drive a project.

K. Berkling et al. (Eds.): SEAFOOD 2008, LNBIP 16, pp. 15–32, 2009.

For the case of total effort estimation, there are a number of function point estimation tools on the market such as Charismatek, Softwaremetrics, TotalMetrics [29], EstimatorPal. But as far as the authors can tell, none of these open their parameters to the user or adapt these by iterations to the project itself. This can be problematic as depicted in Table 1. Despite the large number of studies on this subject, it can be seen how models of such projects can vary. While the trends (formula types) are consistent across studies, the parameters vary greatly, without providing enough guidelines in how they apply to a specific project.

In order to deal with this adaptation, one has to look at iterations to learn the parameters from the project. Drappa et al. [9] developed a simulator to train project managers and give them hands on experience. The project manager interacts with the simulator as s/he would with a team of software developers. The project manager is required to have a set of theoretical skills and uses the tool to gain "practical" experience. This simulator works with function points, that the project manager enters into the tool along with the number of workers used and a set of directions. The simulator will then advance a set amount of time and reflect the status of the project. At each step the project manager continues making decisions to navigate through to the end. Thus, this work takes into account iterations within a project and decision making of the project manager at each stage to change the course of the project. However, while Drappa's work deals with the interaction between project manager decisions and the workers, our work additionally deals with effects of change requests and opens both the parameters and formulas for adaptation. Both systems are based on similar rules of thumb [16]. Jones developed these rules of thumb that are widely quoted and used in the field of software engineering. However, it is still not proven that the same formulae hold for iterations within a project. For now, the simulator uses the rules of thumb that define the parameters needed for data collection but with an understanding that the formula may need to be adapted as data is collected.

Table 1. Table taken from Fairley [10] – demonstrating the large variety of models describing the relationship between development time and lines of code and time elapsed vs. man months

Effort Equation	Schedule Equation	Reference
$PM = 5.2\ (KDSI)^{0.91}$	$TDEV = 2.47\ (PM)^{0.35}$	Walston [26]
$PM = 4.9\ (KDSI)^{0.98}$	$TDEV = 3.04\ (PM)^{0.36}$	Nelson [19]
$PM = 1.5\ (KDSI)^{1.02}$	$TDEV = 4.38\ (PM)^{0.25}$	Freburger et al[12]
$PM = 2.4\ (KDSI)^{1.05}$	$TDEV = 2.50\ (PM)^{0.38}$	Boehm [6]
$PM = 3.0\ (KDSI)^{1.12}$	$TDEV = 2.50\ (PM)^{0.35}$	Boehm [6]
$PM = 3.6\ (KDSI)^{1.20}$	$TDEV = 2.50\ (PM)^{0.32}$	Boehm [6]
$PM = 1.0\ (KDSI)^{1.40}$		Jones [17]
$PM = 0.7\ (KDSI)^{1.50}$		Freburger et al[12]
$PM = 28\ (KDSI)^{1.83}$		Schneider [24]

In order to look at how change affects project timelines, it is necessary to understand the relationships between artifacts. Cleland-Huang et al. [14] worked on a framework to capture traceability in artifacts in order to propagate changes across the project correctly. The framework contains three parts: event server, requirements manager and the subscriber manager that combine to partially automate the process and support the workers in maintaining correct traceability. Our work builds on the subscription model for artifacts that she proposes in order to establish links between artifacts and propagate changes correctly. The traceability is important in order to correctly propagate the effects of change requests to all affected artifacts in the project.

Finally, the degree of change in indirectly related artifacts is important. To this end, Datta [7][8] suggests three metrics: Mutation Index, Component Set, and Dependency index. Mutation Index indicates the level of change a requirement has undergone across iterations; Component Set specifies all the components a particular requirement needs for its fulfillment; and Dependency Index reflects on the extent to which a particular requirement's implementation depends on the implementation of other requirements. These three metrics help evaluate the effects of requirement changes for a software system. Although our work groups function points according to use cases and not requirements, under reasonable assumptions, the Dependency Index is applicable in our scenario, and is referred to in this paper as . Mechanisms for extracting this metric value automatically from code is under development by Datta.

One of the difficulties in working on simulation of projects is the dearth of rich, publicly available training data. A number of databases are available in the public market. The main repository is available through the International Software Benchmarking Standard Group (ISBSG) [28]. This non-profit organization had put together a standard for benchmarking software development in three categories: software enhancements, software implementations, and software maintenance. The information enclosed in the repository is divided into a few types of data like: Rating, Sizing, Effort, Productivity, Schedule and others. However, this repository does not provide information on the changes of parameters as a function of time. The data is not given by iteration or phases.

This work argues towards the collection of discussed parameters by iteration and the importance of adapting the simulator to the specific project by allowing the user to adjust the parameters. Currently, available databases are not yet sufficient to train an iteration-based simulator, nor do they collect sufficient data to appropriately analyze the effect of addition, change and deletion on each iteration or the project as a whole. Yet, iterations and adaptations to very project-specific data are absolutely essential when outsourcing is involved in order to reliably estimate timelines. A more accurate timeline prediction for distributed projects will lead to fewer unpredictable events and will support management decisions by giving more specific and precise estimates. The rest of this paper will describe our approach to combining a number of formulas and parameters into a simulator that can then be used to simulate project timelines and collect data in order to adapt both functions and parameters built into the simulator. We demonstrate reasonable functionality of the current simulator based on well-known facts about projects and show that adaptation is absolutely necessary based on a real-world example, therefore making the call for data collection based on iteration.

Section 2 will discuss the building blocks of the approach used in this paper. Section 3 will discuss trends and parameters within software engineering projects that are used within the simulator. Section 4 will discuss the implementation of the simulator and validate the basic simulator functionality by looking at sequence of operations whose properties transcendent project-specific characteristics. Section 5 will conclude by looking at an example project, demonstrating the clear need and feasibility for both parameter and formula adaptation for any simulation tool on an iteration basis. Section 6 concludes by listing a number of enhancements necessary to expand the model under future work and propose the availability of a web-based tool for data collection and simulation and online adaptation.

2 Foundations

The theoretical foundations of this work include the methodology of software project management, Function Point estimation of project size based on Use Cases and Traceability usage in projects. These three topics are described in more detail before Section 3 will clarify their usage in this work.

2.1 Methodology

For the purpose of this work we use the terminology of the Rational Unified Process (RUP) because it presents the collection of best practices from industry and is readily reducible to other methods [15]. RUP defines the artifacts that the simulator produces to emulate a software project timeline. Artifacts are either final or intermediate work products that are produced and used during a project and generally include documentation and software. They are used to capture and convey project information and results. The simulator works with the major artifacts listed below:

- **Use Case**

Use cases capture the functional requirements of a project. They are usually based on a number of requirements to come together in order to formulate a goal that an actor/specific user of the system will achieve, such as “withdraw money”. A Use Case contains both functional as well as non-functional requirements. The Use Case further is the primary document used by the implementation team to produce the Class diagram, the implementation code and the test case.

- **Software Requirement Specification**

The Software Requirement Specification (SRS) is the document that contains all the functional and non-functional requirements of the system as a whole. The document refers to Use Case documentation for the functional details but retains the overall information. While functional requirements are mainly covered through the use cases, non-functional requirements are usually found in the SRS and can be categorized as usability-, reliability, performance, and substitutability-requirements, design constraints, platform environment and compatibility issues, or applicable standards. In addition, requirements that specify need of compliance with any legal and regulatory requirements may be included. Non-functional requirements that apply to an individual use case are captured within the properties of that use case.

- **Class Diagram**

The Class Diagram is a document which is based on the entirety of the project and therefore depends on all the Use Cases. A change to any Use Case can affect a change in the class diagram.

- **Code**

The Code is designed to implement a Use Case that describes its functionality. For the purpose of this paper the code may belong to several Use Cases as there may be some degree of overlap between Use Cases through common requirements. Therefore, change in one Use Case may affect different code pieces to varying degrees.

- **Test Case**

Test Cases are designed to test the code for a particular Use Case. A change in the Use Case may effect both Test Case and Code.

- **Test Code**

Test Code implements the test case.

2.2 Function Points

Function Points (FP) is a metric for measuring the functional size of a software system. The usage of function points is well known and a tested sizing technique in software engineering [21][18][25][11][13]. FPs have been used since 1979 when Allan Albrecht of IBM [3][4] introduced them. There are other Functional Assessment techniques, mainly Bang, BMA, CASE Size, Entity, IE, Mark II FPA, MGM, and Usability. According to McDonell, Table 2 summarizes that the most tested and generally used functional assessment technique is Function Point Analysis. Mark II FP expects 19 adjustment factors instead of 14 on the original FPA method, making the adjustment factor more difficult to asses in a step in the process where usually the user or PM has little information on the system. Boehm [6] developed and redesigned later an algorithmic cost model called (COCOMO). It provides formulas for the estimation of programmer-month and development schedule based on the estimated number of Delivered Source Instructions (DSI). COCOMO model is based on LOC, this metric is harder to obtain in early stages of the product life cycle making FPA the only tested and validated and more reasonable choice.

Table 2. Comparison of functional assessment and estimation methods (taken directly from McDonell [19])

Method	Automation	Comprehensive	Objectivity	Specification	Testing	Validity
Bang	No	Yes	No	Yes	Yes	No
BMA	Yes	Yes	Yes	Yes	Yes	No
CAES	Yes	Yes	Yes	Yes	Yes	No
Entity	Yes	Yes	Yes	Yes	No	No
FPA	No	Yes	No	No	Yes	Yes
IE	No	Yes	No	No	Yes	Yes
Mark II FPA	No	Yes	No	Yes	Yes	Yes
MGM	No	Yes	No	No	No	No
Usability	No	No	No	Yes	No	No

In this work we focus on the existing relationship between Use Cases, Function Points and duration of code implementation that has been studied by a variety of researchers in the past. While this is a controversial approach [1] [2], it has been shown to work in real-world industrial applications for certain types of projects [10] [5]. The following is a brief presentation of Function Points and the approach chosen for the simulation model in this work because it is empirically shown to work to a reasonable degree according among others also from the International Software Benchmarking Standards Group.

Function Points can be calculated in two parts. The first part relates to the entire project with a handful of parameters, such as: Data communications, Distributed data/processing, performance objectives, tight configuration, high transaction rate, on-line inquiry data entry, end user efficiency, on-line update, complex processing, code reusability, conversion/installation ease, operational ease, multiple site installation, facilitate change. The second number is calculated at the Use Case level by looking at the number of inputs outputs, files accessed, inquiries, and number of Interfaces. This model is based on Albrecht [4] and is more precise in estimation than the previous model of unadjusted function points.

The measurement for a Use Case results from a formula which combines the overall and the specific values into a final FP value. This final number relates to time spent on their implementation through a function that has been established [30] to have a non-linear relationship similar to what is approximated by Figure 1. The relation of function points versus effort can be estimated automatically after a few iterations, assuming that the workers are stable.

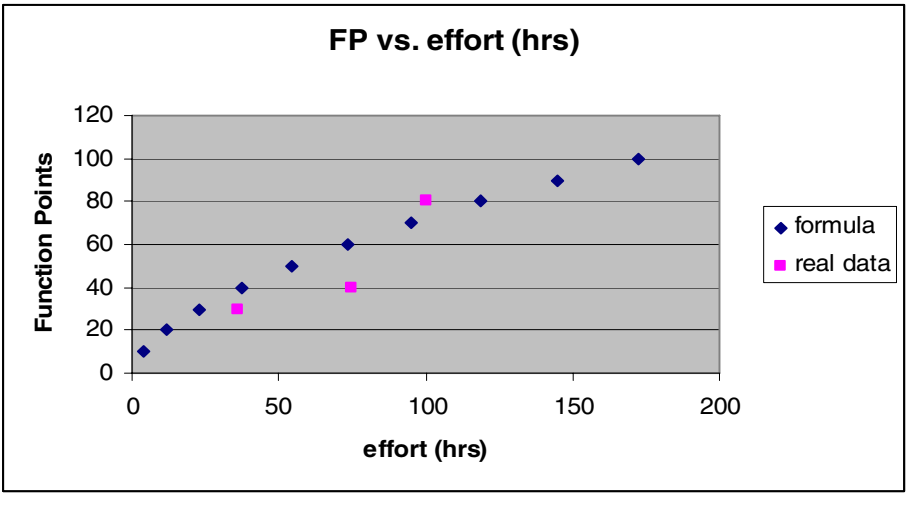

Fig. 1. Assumed relationship function between Function Points and Time spent on coding

2.3 Traceability

Traceability [27] is the process of tracking relationships between artifacts. It is used in software engineering for verification, cost reduction, accountability, and change

management. Tracking the effect of change requests, such as additions, changes or deletions of use cases on other artifacts are tracked in this manner. Its importance can be appreciated by this statement: "The US Department of Defense spends about 4 percent of its IT costs on traceability." [23][22]. A model to simulate project data, like artifacts, meeting minutes, meeting agendas, stakeholders, assumes certain required traceability links for artifacts involved in the project in order to propagate the effects of change correctly. Figure 2 below shows how change can be traced through various artifacts in a project.

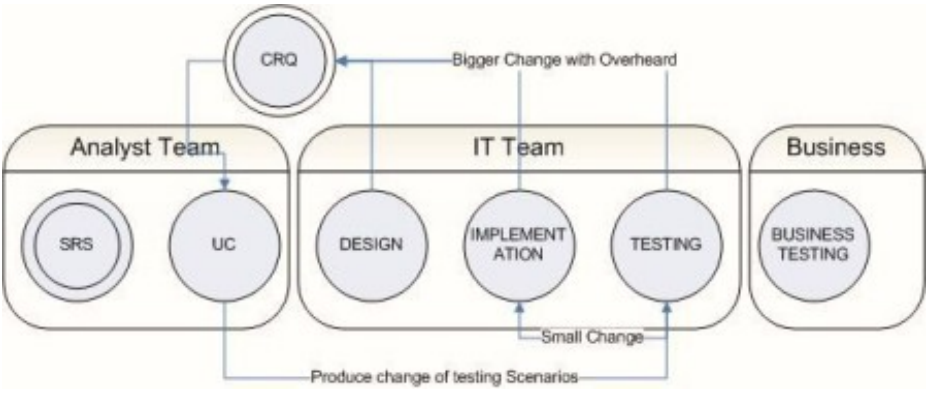

Fig. 2. Simplistic example of how change affects the software life cycle

This project simulator will process specific input like use cases and change requests through traceability models and assumptions into a static project spreadsheet that will capture specific changes in artifacts and all its links. In summary, traceability allows us to see how artifacts are interrelated within a project. This allows us to apply the rules to the project given the collected data.

3 Implementation

Each of the components described above covers aspects of project description that in combination are able to support the simulation model. In order to take the complex interrelationships into account that result in the model of iterations, this section describes a combination of formulas and parameters that make up the simulator.

3.1 Model

The Model presented in the previous section using the RUP terminology is now described in more detail with further assumptions and parameters and outlining the interrelationships between the artifacts.

- **Software Requirement Specification**

The Software Requirement Specification (SRS) is the document that describes the system as a whole and refers to the Use Cases for details of the functional specifications in

a modularized fashion. Change requests to Use Cases may affect the SRS. The time to write an SRS is related to the number of Use Cases and non-functional requirements of the system.

- **Use Cases**

For the purpose of this work, change requests act on Use Cases directly. More than one change request is required if more than one Use Case documentation is affected by the change. This does not hold true for code and class diagram. There is a degree of interdependence between class diagrams across Use Cases. A change request to a Use Case at the documentation level does affect code of other Use Cases to some degree. We model this interdependence with α as indicated by Figure 3. For the purpose of this work, we can assume that there is some degree of overlap between Use Cases regarding the Classes/Objects and the corresponding code sections that are generated. For example, imagine a system with two use cases. The first one describes how books are entered with title only, the second one how to search for them by title. Now, the first use case, for entering new books, receives a change request to add the author field. After those changes are made, the second use case receives the change request to be able to search by author as well. This change is done much faster than the first change since the class diagram has already been updated and the only change that is needed is at the user interface level. This difference in effort required due to the overlap is denoted by α in Figure 3 below. The overlap or interdependency of requirements that make up each of the Use Cases results in various degrees of interdependence between the Use Cases and is one of the parameters of the simulator that can currently be v aried. However, it represents a value that can be extracted from the software and is currently studied by one of the authors, S. Datta.

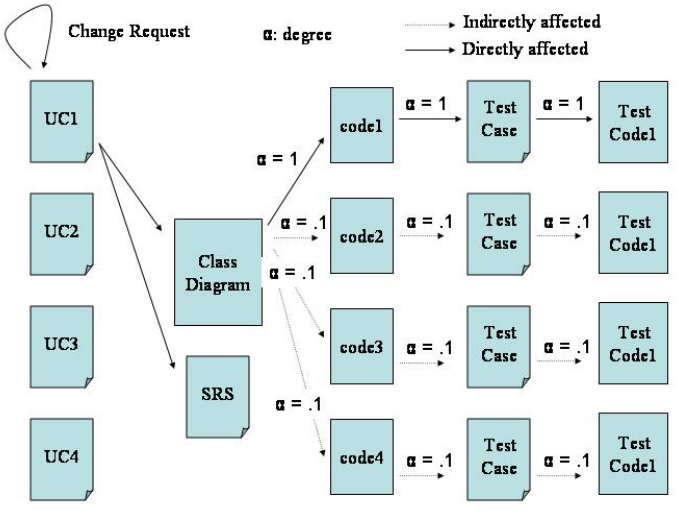

Fig. 3. Simplified view of interrelationships between artifacts

- **Class Diagram**

It is through the use cases that changes in the Class Diagram are effected and propagated through to the Code.

- **Code**

A change in the Use Case is measured in function points and effects a change in the code with the amount of effort related to the FP. Code can be reused between Use Cases whish is related through as described above. Therefore, change in one Use Case may affect different code pieces to varying degrees. Changes directly acting on code, such as refactoring of code, are not currently taken into account in this simulator.

- **Test Case**

A change in the Use Case effects a change in the test case directly.

- **Test Code**

Test Code implements the test case and is affected directly by a change in the Test Case.

There are other artifacts that belong to the Rational process which should be taken into account in a later version of this simulator. These are, among others the metrics report, the configuration management plan, the project plan, test management plan, the risk management plan, the risk list, the user manual and the installation manual. We currently leave their more detailed implementation for the future work section. Section 3.2 describes how it is possible to lump the entire lines of written documentation into an overall effort size that relates directly to function points as well.

In addition to taking into account the interdependence between artifacts that add to the level of complexity of changes, we model the penalty factor called "Level of change". It relates to the time difference between modifications of an artifact under the assumption that it becomes increasingly difficult to change older artifacts. For example, if a use case is inserted in iteration 3 and modified in iteration 7 then the level of change is 7-3=4. According to the level of change, x, the penalty is calculated by (1-(1/x^.5)) in the current simulator. This function is based on heuristics of managers, a verification of function and parameter is possibly only through iteration-based data collection.

3.2 Documentation Time

A number of formulas and parameters derived from various sources are combined to formulate the duration of tasks within the project plan. In this section, the formulas are listed, the parameters identified and the default values stated. The equation for the total number of pages produced in a project is related to function points as defined by Caper Jones [16] and given by Equation 1, where AFP stands for the adjusted function points and TNP stands for Total Document Pages in Project. The parameter p is a value defined as 1.15 Jones and is the default value used by the simulator as specified in Table 2.

$$\text{TNP} = AFP^{p} \tag{1}$$

The following documents are currently part of the simulator: Software Requirements Specifications, Use Case, and Test case documents. All other documents are lumped into a single set, containing metrics report, the configuration management plan, the test management plan, the risk management plan, the risk list, the user manual and the installation manual. Equation 2 shows how these components make up the total number of pages TNP from Equation 1, where uc, srs, tc, and o denote the percentage of

added pages to the total number TNP. This relationship has to be collected from data. The assumptions made by the simulator are stated in Table 2 but can be adapted after several iterations of the project to reflect the specific project more accurately.

$$TNP = uc \cdot TNP + srs \cdot TNP + tc \cdot TNP + o \cdot TNP \quad (2)$$

The total number of pages is converted into time by using yet another equation that relates writing time to page numbers [31] as defined by Equation 3, where WPP is words per page and WPM stands for Words per Minute.

$$\text{Documentation Minutes} = TNP * WPP / WPM \quad (3)$$

Though these formulas are research based, it seems unlikely that pages written for different documents can be written with equal speed. Therefore, this data should also be collected. The true relationship would have to be given through the data. Table 3 depicts the default values that are used in the current system that can be adapted after a few iterations. Similarly, Equation 2 could be rewritten differently not in terms of Function Points but rather in terms of number of Use Cases as well as function points. One can assume that the size of a Use Case is a relatively constant number UC_base since Use Cases have a limited size. The SRS also grows linearly with respect to the number of Use Cases added (SRS_base + n · SRS_add). Parts of the Use Case (activity diagram) and the Test Case (test scenarios dependent on activity diagram) depend heavily on the function points in terms of time to write those pages, but not necessarily in terms of number of pages. Therefore, none of these components weigh heavily in the polynomial. Most of the documenting pages therefore must be spent on the other documents that were lumped into "other" (such as project plan, risk management plan, test plan, etc.) or the formula seems wrong. Equation 4 depicts the form the resulting formula would take, which would need to be verified with real data.

$$\text{Time} = n \cdot FPA + FPA^q \quad (4)$$

Table 3. List of variables needed by simulator and their initial values

Documentation	Variable	Value
Use Case + Test Case + SRS	(uc + srs + tc)	n = .76
Exponent	p	1.15 [16]
Words per page	WPP	250 [31]
Words per Minute	WPM	19 [32]

3.3 Coding Time

As described in Section 2.1, coding time has a determinable relationship to function points usually depicted as a polynomial curve as defined by Equation 5, where the number of man months increases at a faster rate than the number of function points but is linear for smaller function point levels. It is also well-known that the slope

depends largely on the team and the type of project. Therefore, the user is asked to supply this variable, *q* in Equation 5, with each iteration. It is necessary to record this variable for each iteration during data collection in order to see the detailed effects of changes in a particular project. The non-linearity effect is not visible for small Use Cases and change requests.

$$SLOC = AFP^q; q = 0.6 \quad (5)$$

Effort is then calculated based on rate of coding (LOCperday) and hours worked per day (hrsperday) as described by Equation 6.

$$Hours = (SLOC / LOCperDay) \cdot hrsperday \quad (6)$$

These two formulas cover coding, but not really design. Class and database diagram are inherently related to function points as is the user interface. It is not unreasonable to assume a polynomial function relates Function Points to design effort in a similar way as it does to coding effort. This function can be approximated with a linear function for medium sized (3-6 months) projects, perhaps with a different constant that will have to be collected as well from the project. Table 4 summarizes the variables and their original default values that are consequently adapted after each iteration.

Table 4. List of variables relating design

Artifact	Variable	Formula
Source Code	Hrsperday / LOCperDay	= 8/100
Source Code	Q	= Entered by user; default .6
Class/Database Diagram	q'	= AFP^q' ; q' = q
GUI Interface	q"	= AFP^q" ; q" = q
Source Code	SLOC = AFP^q	q = 0.6
Test Code	SLOTC	= c ·SLOC, c=1

3.4 Assumptions

The model described above specifies key documents of the project management process. Similar models have to be developed for other documents. In addition, communication and meeting time becomes a major component as a function of both project size and distance between team members, becoming potentially non-linear. These relationships and their changes need to be captured for each iteration. The current simulator assumes one worker, a first step before expanding the model to several workers and distributed environments. The simulator also follows the assumption that the formulas in the literature are correct. However, as data is collected, these formulas as well as their parameters are open for adaptation. Section 5 will demonstrate this necessity on a sample project.

4 Simulation

The simulator proceeds in several steps that serve to collect project-specific data. In this manner the project variables can be set at the beginning and during the project.

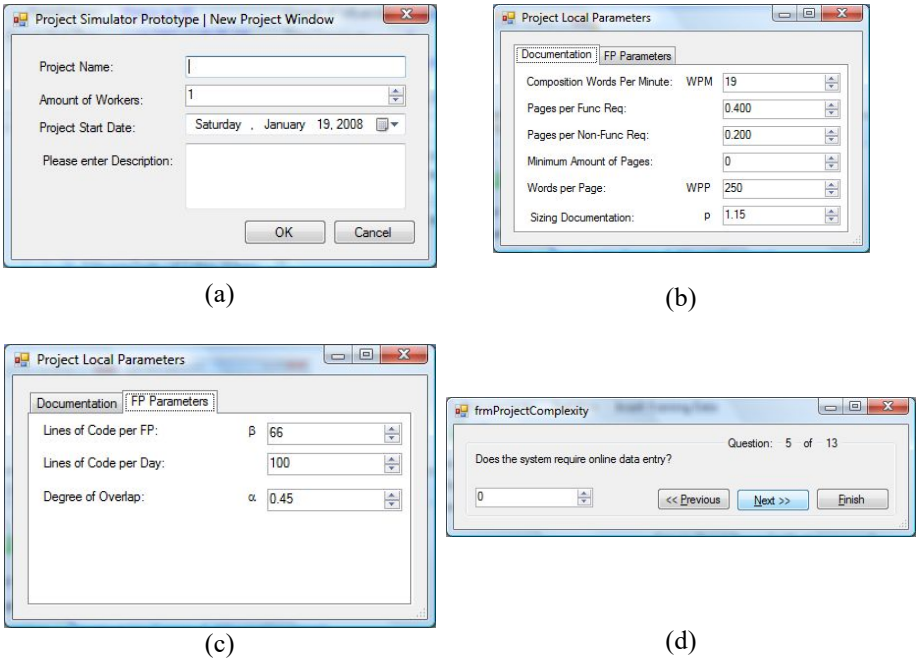

Fig. 4. Sequence of displays to start a project. (a) project specific information, (b,c) variables from Tables 3 and 4, (d) Use Case function points detailed entry form.

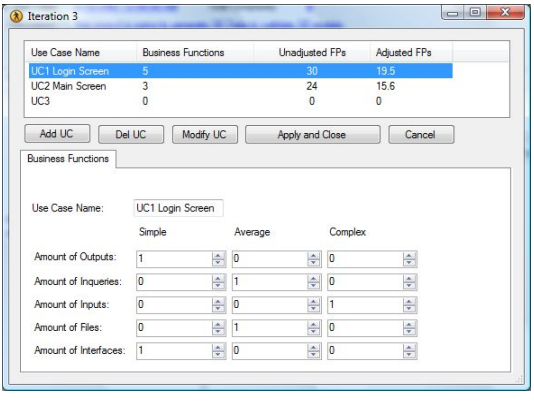

Fig. 5. Data entry for Adding, Subtracting and Changing a Use Case in terms of Function Points

The entry of function points for each use case and change request into the simulator is depicted in Figure 6 as described in Section 2. Each component (input, output, inquiries, files, interfaces) is qualified as simple, medium or complex. This categorization is clearly defined by Paton and Abran [21].

The resulting screenshot for the first iteration is shown in Figure 6. It shows the artifacts created within the project, using traceability rules: A Use Case as entered in the screen in Figure 5 is linked to several documents that depend on it: SRS, Code, Test Case and Test Code as well as the design documents.

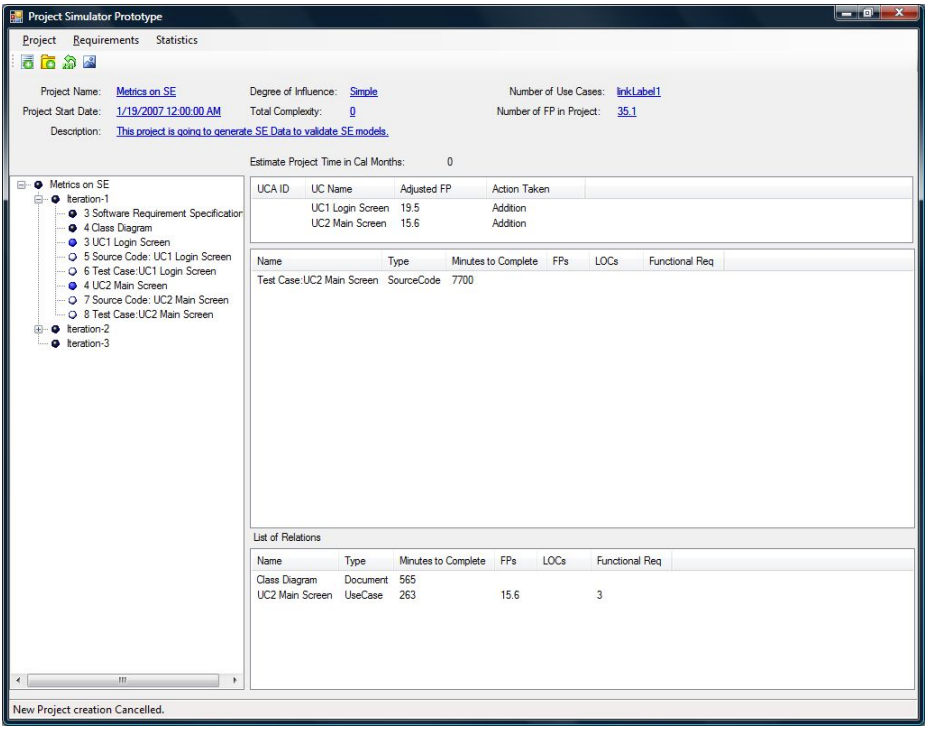

Fig. 6. Screenshot depicting the first iteration of Use Cases

The basic operations in Software Engineering regarding change requests are the adding, deleting and changing of use cases as well as the order and size these operations are presented in. There are well known effects on project timelines that result from particular scenarios. We know that change requests submitted late in the project are more expensive than early change requests, smaller use cases are easier to change than larger ones and less modular code and documentation is difficult to update according to change requests. With the current set of formulas and parameters the simulator is sufficiently complex to demonstrate the effects as expected. These scenarios hold true for a large range of parameters. That is because they transcendent project specific information. Below, are example simulation runs for specific parameters for each of these well-known scenarios.

"Change requests are more economical in the beginning of the project than in the end"

In this example four Use Cases with 10 FP each, are added consecutively in separate iterations. After four iterations the project is completed. The simulator should reflect that a change request to the first Use Case submitted in iteration 2 will affect the entire project less than the same change request submitted late in iteration 4. Below is the depiction of the project details of each scenario. A late change request has a bigger impact on the project duration. This demonstrates both the level of change property as well as the impact provided by changes on more artifacts due to the parameter α.

Scenario A: 4 Iterations with a new USE CASE in each iteration of 10FP with a change on iteration 2 of 8 FP	Scenario B: 4 Iterations with a new USE CASE in each iterations of 10FP with a change on iteration 4 of 8FP
Total Artifacts: 23	Total Artifacts: 27
Iterations 1: 1.38 days	Iterations 1: 1.38 days
Iterations 2: 3.63 days	Iteration 2: 1.38 days
Iterations 3: 1.63 days	Iteration 3: 1.5 days
Iterations 4: 1.75 days	Iteration 4: 5.5 days
Total: 8.38 days	Total: 9.75days

"A larger number of small use cases are more efficient than a smaller number of large use cases"

In this experiment we want to show that all being equal a Project of 80FP at the first Iteration and a change of 10FP in second iteration will take less time if the project is broken into more functional units. In order to show the effect, two scenarios are created. The first project contains 2 use cases of 40 FP each totaling 80 FP. Then Use Case 1 will be modified by adding 10 more FP. In the second project 4 Use Cases of 20 FP each totaling 80 FP will be followed by a change request to Use Case 1 by 10 FP.

Scenario A: 2UC with 80FP Total Count	Scenario B: 4UC with 80FP Total Count
Amount of Artifacts: 15	Amount of Artifacts: 25
Iteration1: 21.88 days	Iteration1: 15.63 days
Iteration2: 7 days	Iteration2: 7 days
Total: 28.88 days	Total: 22.13

In this example, a delay of approximate 6 days is due because the Use Cases were larger in Scenario 2. The entire project takes less time in the second example because changes to smaller and well modularized code a) have less dependency on other code and b) are less difficult to change due to their size. Point (a) is denoted by α as depicted in Figure 3 and is set to 5% for this demonstration. Point (b) is implemented with the non-linear function described in Equation 5. As a result, fulfilling a change request of 10 function points is less work when applied to a 20 FP Use Case than a 40 FP Use Case.

"Show effect of non-modularity of Use Cases"
In this example, the simulator compares two scenarios in which two use cases overlap to varying degrees as modeled by α depicted in Figure 3, a parameter that reflects the degree of overlap of components in the database model. In Scenario A, α is set to 5% overlap, modeling a good separation of the use cases; in Scenario B, α is set to 45% overlap, demonstrating a high degree of overlap between use cases. In both projects, a change request is submitted in the second iteration. The simulator can show that a larger degree of dependence between use cases results in a longer duration as predicted by common sense and the model. Both projects have two use cases with 80 FP in total and were affected by the same change request in the second iteration.

Scenario A: 5percent overlap within two usecases of 20FP each and a change of 10FP	Scenario B: 45percent overlap within two usecases of 20FP each and a change of 10FP
Amount of Artifacts: 15	Amount of Artifacts: 15
Iteration 1: 7.38	Iteartion 1: 7.38
Iteration 2: 4.63	Iteration 2: 6.63
Total: 12 days	Total 14 days

Scenario A is 2 days shorter due to the lesser degree of overlap between use cases. With increased overlap between use cases change requests add more effort to the total project because the change request has repercussions throughout a larger area of the project. It is appreciable that if you make a system more independent across use cases then you will diminish the amount of total time to create the changes across the life of the software development cycle. This effect is compounded as the project size increases as we know from the previous example.

5 Conclusion

They key to this simulator is not (only) to show the effects of change requests on code, but to specify the rules governing those changes and distill the parameters and functions that are essential to both model the data and define the data to be collected for each iteration. We have made the argument that only iteration-based data can support an accurate data-driven model for comparative studies of models based specifically on iterations. In this paper, the authors have suggested a number of parameters and functions that need further study for iteration-based projects in order to model a software process accurately. Only through full understanding can we grasp the additional benefits and difficulties that are involved in off-shoring parts of software projects.

Looking at a sample real-world project developed with off-shoring, it can be seen that many of the assumed parameters and functions may or may not apply for iterations and small- or medium-sized projects as can be seen in Figure 7. In this particular project at hand, provided by the fourth author S. Datta, the actual relationship is quite linear compared to the estimated relationships given by Equations 1 and 5. Figure 7 depicts the best fit linear function compared to the polynomial from the literature. Additionally, Figure 7 shows that coding requires most of the effort, followed by documentation, test code, code design and User Interface design. Each of

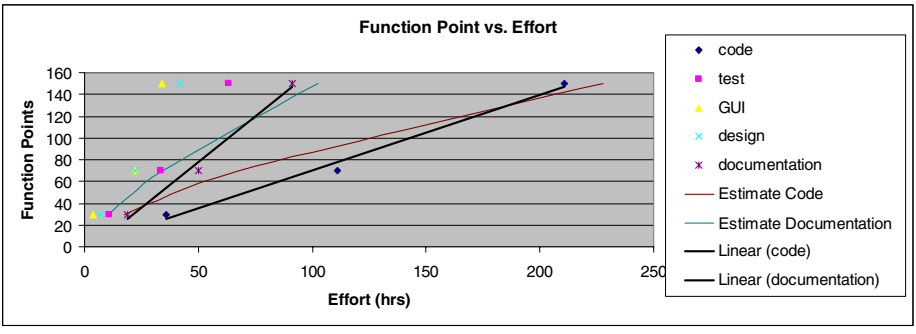

Fig. 7. Actual relationship between Function Points and effort for various artifacts compared to estimated relationship

the progressions seems linear up to the third iteration. This example clearly shows the need for iteration-based data collection to estimate both the function as well as the parameters by taking data of preceding iterations for the adaptation process to increase the prediction ability of the simulator for the next iteration.

Different artifacts are related to function points in a similar manner, for example, test case and use case documentation efforts exhibit a linear relationship for the same function point value as depicted in Figure 8.

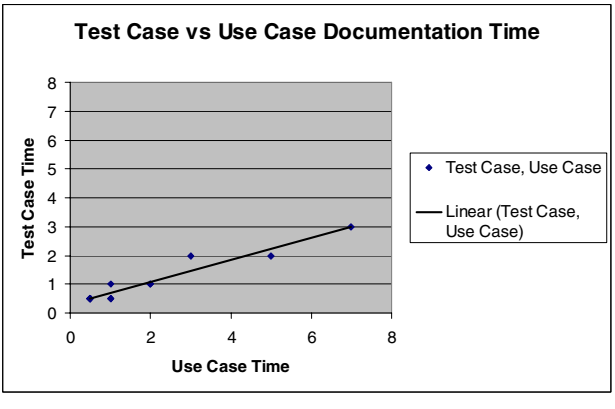

Fig. 8. Experimental data shows that not all artifacts are written at equal speeds

The following are recommendations for collecting the required data.

1. Parameters need to be collected with each iteration
2. Parameters include the following by iteration:
   - list of changes per Use Case (add, delete, modify)
   - LOC/FP
   - LOC/day/person
   - Time spent on each Use Case and related code per change

- Time spent on each Test Case and related test code per change
- Time spent on each other artifact (SRS, Design, GUI, etc.)
- Pages added or changed for each artifact as function Use Case operation (add, delete, modify)

3. Information that relates the factor *[i,j]* between artifacts *i* and *j* by describing how changes within one Use Case affect other Use Cases and their related code and data-tables.
4. Information about the amount of communication related to each iteration in terms of time spent with emails, meetings and other forms of communication.

6 Future Work

Future work clearly includes analysis of the collected data including the new variables. The addition of division of work effort through additional personnel will be needed along with the communication components. Meeting and communication equations are essential additions to this model that will support understanding of the off-shoring component. Therefore, it is very important to collect off-shoring project data with the iteration-based model. Using the current ISBSG database, it seems possible to show that off-shoring does not add an element of complexity to the project [33]. This however seems to run contrary to industrial experience reports, leading us to the idea that some parameters are still missing. Perhaps, iteration-based data will illuminate this issue further. We also want to introduce mean and variance into the predicted schedule. It can be shown that off-shoring may not change the mean prediction time of the project but the variance increases. We would like to show the origin of this increased variability and show how the variability can be controlled.

Finally, current efforts are underway to move the desktop simulator to a web-based application in order to serve as a tool and a data-collection instrument at the same time. It should have the ability to adapt parameters as well as functions automatically with the incoming data.

References

[1] Abran, A., Robillard, P.N.: Function Points: A Study of Their Measurement Processes and Scale Transformations. Journal of Systems and Software 25, 171–184 (1994)
[2] Abran, A., Robillard, P.N.: Identification of the structural weakness of Function Point metrics. In: 3rd Annual Oregon Workshop on Metrics, pp. 1–18 (1991)
[3] Albrecht, A.J.: Measuring application development productivity. In: IBM Corp. (ed.) IBM Application Develop Symp. (1979)
[4] Albrecht, A.J., Gaffney, J.E.: Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Transactions on Software Engineering SE-9(9), 639 (1983)
[5] Angelis, L., Stamelos, I., Morisio, M.: Building a Software Cost Estimation Model Based on Categorical Data. In: Pro. of the Seventh International Software Metrics Symposium METRICS, pp. 4–15 (2001)
[6] Boehm, B., Englewood Cliffs, N. (eds.): Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)

[7] Datta, S.: A Mechanism For Tracking The Effects of Requirement Changes In Enterprise Software Systems, Master's thesis, Florida State University (2006)
[8] Datta, S., van Engelen, R.: Effects of Changing Requirements: A Tracking Mechanism for the Analysis Workflow, pp. 1739–1744 (2006)
[9] Drappa, A., Ludewig, J.: Quantitative modeling for the interactive simulation of software projects. Journal of Systems and Software 46(2-3), 113–122 (1999)
[10] Fairley, D.: Making Accurate Estimates. IEEE Software, 61–63 (2002)
[11] Fetcke, T.: A Generalized Structure for Function Point Analysis. In: International Workshop on Software Measurement, pp. 143–153 (1999)
[12] Freburger, K., Basili, V.: The Software Engineering Laboratoy: Relationship Equations, Report TR764, Technical report, University of Maryland (1979)
[13] Ho, V.T., Abran, A., Oligny, S.: Using COSMIC -FFP to Quantify Functional Reuse in Software Development. In: Proc. of the ESCOM-SCOPE, pp. 299–306 (2000)
[14] Huang, J.C., Chang, C.K., Christensen, M.: Event-Based Traceability for Managing Evolutionary Change. IEEE Transactions on Software Engineering Journal 29, 796–810 (2003)
[15] IBM, Rational Unified Process Best Practices for Software Development Teams, Technical report, IBM (1998)
[16] Jones, C.: Software Estimating Rules of Thumb, 116–119 (2007)
[17] Jones, T.: Program Quality and Programmer Productivity, Technical report, IBM, IBM TR 02.764 (1977)
[18] Lawrie, R., Radford, P.: Using Function Points in Early Life Cycle estimation. In: Proc. of the 4th European Conference on Software Measurement and ICT Control, pp. 197–210 (2001)
[19] MacDonell, S.G.: Comparitive review of functional complexity assessment methods for effort estimation. In: Software Engineering Journal, pp. 107–116 (1994)
[20] Nelson, R.: Software Data Collection and Analysis at RADC, Technical report, Rome Air Development Center (1978)
[21] Paton, K., Abran, A.: A Formal Notation for the Rules of Function Point Analysis. Research Report 247, University of Quebec, pp. 1–49 (1995)
[22] Ramesh, B.: Process Knowledge Management with Traceability. IEEE Software, 50–52 (2002)
[23] Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability. IEEE Transanction on Software Engineering Journal 27(1), 58–93 (2001)
[24] Schneider, V.: Prediction of Software Effort and Project Duration: Four New Formulas. In: ACM SIGPLAN Notices (1978)
[25] Symons, C.: Come Back Function Analysis (Modernised) - All Is Forgiven!. In: Proc. of the 4th European Conference on Software Measurement and ICT Control, pp. 413–426 (2001)
[26] Walston, C., Felix, C.: A Method of Programming Measurement and Estimation(1), Technical report, IBM System (1977)
[27] Watkins, R., Neal, M.: Why and How of Requirements Tracing. IEEE Software 11, 104–106 (1994)
[28] http://www.isbsg.org
[29] http://www.totalmetrics.com
[30] http://www.engin.umd.umich.edu/CIS/course.des/cis525/js/f00/harvey/FP_Calc.html
[31] http://www.writersservices.com/wps/p_word_count.htm
[32] http://en.wikipedia.org/wiki/Words_per_minute
[33] SMEF 2005 proceedings, http://realcarbonneau.com/Publications/Carbonneau2005_SoftDevProd_SMEF.pdf