Estimating Software Based on Use Case Points

Edward R Carroll
Agilis Solutions, A Business Unit Of Hepieric, Inc
3601 SW Murray Boulevard
Beaverton, Oregon 97005
1-503-517-2867
edcarroll@agilissolutions.com

ABSTRACT

It is well documented that software product cost estimates are notoriously inaccurate across the software industry. Creating accurate cost estimates for software product development projects early in the product development lifecycle has always been a challenge for the industry. This article describes how a large multi-team software engineering organization (over 450 engineers) estimates project cost accurately and early in the software development lifecycle using Use Case Points, and the process of evaluating metrics to ensure the accuracy of the model.

The engineering teams of Agilis Solutions in partnership with FPT Software, provide our customers with accurate estimates for software product projects early in the product lifecycle. The bases for these estimates are initial definitions of Use Cases, given point factors and modified for technical and environmental factors according to the Use Case Point method defined within the Rational Unified Process. After applying the process across hundreds of sizable (60 man-months average) software projects, we have demonstrated metrics that prove an estimating accuracy of less than 9% deviation from actual to estimated cost on 95% of our projects. Our process and this success factor is documented over a period of five years, and across more than 200 projects.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *Cost estimation;* K.6.1 [**Management of Computing and Information Systems**]: Project and People Management*;* K.6.3 [**Management of Computing and Information Systems**]: Software Management – *Software process;* G.3 [**Probability and Statistics**]: Probabilistic algorithms

General Terms: Algorithms, Management, Measurement, Economics.

Keywords: Use Case, Use Case Point, Function Point Analysis, RUP, Rational Unified Process, estimating, parametric estimating, metric, Capability Maturity Model, CMM.

1. INTRODUCTION

Agilis Solutions is a software engineering consulting company based in Beaverton, Oregon, providing outsourcing software product development and engineering projects to client software companies working to move their products forward, but needing additional engineering capability to meet the demands of their marketing requirements.

Agilis Solutions is partnered with FPT Software Corporation, based in Hanoi, Vietnam, in a blended delivery model that combines Agilis Solutions seasoned Project Managers, Technical Architects and Analysts with the highly capable software engineering teams of FPT Software. FPT Software is one of only 120 organizations in the world certified at level-5, the highest level of the Capability Maturity Model (CMM) from the Software Engineering Institute (SEI) of Carnegie Mellon University.

It is well documented that software product cost estimates are notoriously inaccurate across the software industry. Creating accurate cost estimates for software product development projects early in the product lifecycle has always been a challenge for the industry. Most software engineering managers, and sometimes the teams, estimate project cost by using a gut feeling for the basis of their estimate. At best they base their estimates on their personal experience (which may be significant) in developing like work, although those experiences are likely not documented methodically in a database where the data can be extracted and applied systematically to the next project (with proper modification for the differences between the efforts). Nor, typically, do these engineering teams routinely measure the accuracy of their estimates. And measurements of the engineering processes that created the product are often limited to a one hour discussion at the end of the project (called a lessons learned meeting), if anything at all is ever discussed.

This article describes how the Agilis Solutions and FPT Software engineering teams create accurate estimates for software product projects early in the product lifecycle, well before the requirements are fully defined and the system fully designed. The bases for these estimates are initial definitions of Use Cases, given point factors and modified for technical and environmental factors. After applying the process across hundreds of sizable (60 man-months average) software projects, we have demonstrated metrics that prove an estimating accuracy of less than 9% deviation between actual and estimated cost on 95% of our projects. Our process and this success factor is documented across 200 projects over a period of five years.

1.1 History

Several estimating models have been developed over the years. Those preceding Use Case Point (UCP) and forming the basis for the UCP model include Function Point Analysis and the Constructive Cost Model.

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

Function Point Analysis (FPA) was a valuable technique developed by A. J. Albrecht, in 1977. FPA assigns a point to each function in an application. Various modifiers then act upon the function points in order to adjust for the product environmental. Modifiers typically included applying weighted percentages or multipliers that would simply increase or decrease the point values. Environment factors included modifiers for complexity of technical issues, developer skill level, and risk. One problem organizations attempting to use this method would run into was consistent definition of a function and consistent definition of environmental factors across multiple projects and multiple development languages. In order to produce reliably accurate estimates, FPA relies heavily on historical data to derive weighting values and modifiers.

COCOMO, which stands for Constructive Cost Model was created by Barry Boehm, in 1981. COCOMO used statistical returns to calculate project cost and duration within a given probability. The model sought to provide a tool for predictably estimating cost, and continues to evolve today under the sponsorship of the University of Southern California. The model was/is interesting and produced worthy merits in applying statistical analysis to the problem of cost estimating. However, a major defining point in statistics is sample set size. The underlying assumption for COCOMO (like FPA) is that a statistically significant historical database exists to drive the statistical factoring. This will become a common theme through many attempts to create estimating models. Software engineering teams are typically very good at collecting list of bugs, but notoriously bad at gathering meaningful historical statistically significant metrics useful in predicting future projects.

In the mid-1990s, Jim Rumbaugh, Grady Booch and Ivar Jacobson of Rational Software Corporation developed the Unified Modeling Language (UML) as notation and methodology for developing object oriented software. UML was incorporated into the Rational Unified Process (RUP) by Rational Software. Within UML is the concept of defining the requirements for software products with Use Cases. Around the same time, Gustav Karner, also of Rational Software Corporation, created a software project estimating technique based on Use Case Points, much the way that FPA assigns points to functions, and including statistical weighted modifiers. Karner's technique is incorporated into RUP.

Use Cases, as defined by UML, describe the things actors want the system to do and have proven to be an easy method to capture the scope of a project early in the project lifecycle. For our use, we like the ability to create estimates early in the project lifecycle as a way to be responsive to the needs of our customers. Additionally, we find Use Cases to be a more consistent artifact then functions upon which to base an early project estimate. However, like COCOMO and FPA, the accuracy of estimates created using the RUP UCP estimating technique is largely dependent upon a large amount of relevant historical metrics.

The Agilis Solutions/FPT Software team started collecting relevant project estimate metrics approximately five years ago; when we first started down the road toward certification under CMM. Three years ago, we started using the Rational Software suite and adopted the RUP methodology. Based on extensive feedback from our historically compounding data provided by our project teams, our estimating model has been validated.

The objective of this paper is to describe how our feedback process has been effective in creating estimates that are accurate to less then 9% deviation from original estimate to completed actual, across 95% of hundreds of projects. Our average project size is 60 man-months, with team sizes of 6-12 members. Our data is based on results from over two hundred (200) projects, over a five year period, using a resource pool that has grown to over 450 software engineers cross-assigned into projects in a software engineering project consulting business environment with continuously changing customers and priorities.

2. ESTIMATING USE CASE POINTS

This section will describe the Use Case Point estimating model. The model is described here for clarity in illuminating our continuous process improvement progress. The model presented is essentially the same as published by Geri Schneider and Jason Winters in their book, *Applying Use Cases* [1].

2.1 Actors

The process starts by considering the Actors. For each actor, determine whether the actor is a simple, average or complex actor. A simple actor represents another system with a defined Application Programming Interface (API). An average actor is either another system that interacts through a protocol such as TCP/IP, or it is a person interacting through a text-based interface. A complex actor is a person interacting through a graphical user interface (GUI).

Count the number of simple, average and complex actors and list the quantity of each in the table. Multiply the quantity of each type of actor times the weighting factor and sum the total.

Table 1. Weighting Actors for Complexity

Actor Type	Description	Qnty	Weight Factor	Sub total
Simple	Defined API	3	1	3
Average	Interactive or Protocol driven interface	2	2	4
Complex	Graphical User Interface	1	3	3
Total Actor Points				10

It should be apparent that the weighted values used to modify the quantity of actors (in this section, as in subsequent sections) are only valid if the impact they have on the result is backed up by feedback from historical data. In the case of Actors, use the feedback from historical data to adjust the definition of simple, average or complex.

2.2 Weighting Use Cases

For each use case, determine whether it is simple, average or complex based on the number of transactions in a use case, including secondary scenarios. For this purpose, a transaction is defined as an atomic set of activities which is either performed entirely or not at all. A simple use case has 3 or fewer transactions, an average use case has 4 to 7 transactions, and a complex use case has more than 7 transactions.

If analysis classes have been defined for the system, and it has also been identified as to which ones are used to implement a particular use case, use this information in place of transactions to determine the use case complexity. Note: Used use cases or extended existing use cases do not need to be considered.

Count the number of simple, average and complex use cases and list the quantity of each in the table. Multiply the quantities of each type of use case times the weighting factor and sum the total.

Table 2. Weighting Use Cases for Complexity

Use Case Type	Description	Qnty	Weight Factor	Sub total
Simple	3 or fewer transactions	3	5	15
Average	4 to 7 transactions	2	10	20
Complex	Greater than 7 transactions	1	15	15
Total Use Cases				50

Add the total for Actors to the total for use cases to determine the Unadjusted Use Case Points (UUCP).

- Weighted Actors + Weighted Use Cases = UUCP

  10 + 50 = 60

The UUCP will be further modified to reflect the complexity of the project and experience level of the development team.

2.3 Weighting Technical Factors

Weighting technical factors is an exercise to calculate a Use Case Point modifier which will modify the UUCP by the weight of the technical factors. Start by calculating the technical complexity of the project. This is called the technical complexity factor (TCF). To calculate the TCF go through the following table and rate each factor from 0 to 5. A rating of 0 means the factor is irrelevant for this project, 5 means it is essential. For each factor multiply its rating by its weight from the table.

Table 3. Weighting Technical Factors

Technical Factor	Factor Description	Weight Factor	Project Rating	Sub total
T1	Must have a distributed solution	2	5	10
T2	Must respond to specific performance objectives	1	3	3
T3	Must meet end-user efficiency desires	1	5	5
T4	Complex internal processing	1	5	5
T5	Code must be reusable	1	3	3
T6	Must be easy to install	.5	3	1.5
T7	Must be easy to use	.5	3	1.5
T8	Must be portable	2	0	0
T9	Must be easy to change	1	5	5
T10	Must allow concurrent users	1	0	0
T11	Includes special security features	1	5	5
T12	Must provide direct access for 3rd parties	1	0	0
T13	Requires special user training facilities	1	3	3
Total TFactor				42

Add the resultant values together to get the total T factor.

- (Weighting Factor) * ∑(Tlevel) = TFactor

The TFactor does not directly modify the UUCP. To calculate Technical Complexity Factor (TCF), multiply TFactor by 0.01 and then add 0.6.

- (0.01 * Tfactor) + 0.6 = TCF

  (0.01 * 42) + 0.6 = 1.02 TCF

Calculate the size of the software (use case) project by multiplying UUCP times TCF.

- UUCP * TCF = SzUC

  60 * 1.02 = 61.2

Note on Reusable Components: Reusable software components should not be included in this estimate. Identify the UUCP associated with the reusable components and Adjust the size of SzUC accordingly.

2.4 Weighting Experience Factors

The level of experience for each team member can have a great affect on the accuracy of an estimate. Consider the experience level for each team member, called the Experience factor (EF).

Table 4. Weighting Experience Factors

Experience Factor	Factor Description	Weight Factor	Project Rating	Sub total
E1	Familiar with FPT software process	1	4	4
E2	Application experience	0.5	2	1
E3	Paradigm experience (OO)	1	4	4
E4	Lead analyst capability	0.5	4	2
E5	Motivation	0	4	0
E6	Stable Requirements	2	2	4
E7	Part-time workers	-1	0	0
E8	Difficulty of programming language	-1	3	-3
Total EFactor				12

To calculate EF, go through the table above and rate each factor from 0 to 5. For factors E1-E4, 0 means no experience in the subject, 3 means average, and 5 means expert. For E5, 0 means no motivation on the project, 3 means average, and 5 means high motivation. For E6, 0 means unchanging requirements, 3 means average amount of change expected, and 5 means extremely unstable requirements. For E7, 0 means no part-time technical staff, 3 means on average half of the team is part-time, and 5 means all of the team is part-time. For E8, 0 means an easy to use programming language is planned, 3 means the language is of average difficulty, and 5 means a very difficult language is planned for the project.

For each factor, multiply its rating by its weight from the table above. Add together all of these factors to get the total E factor.

- ∑(Elevel) * (Weighting Factor) * = Efactor

Calculate the Experience Factor (EF) by multiplying Efactor times –0.03 and adding 1.4.

- (-0.03 *Efactor) + 1.4 = EF

  (-0.03 * 12) + 1.4 = 1.04

To calculate Use Case Points (UCP), multiply SzUC by EF

- SzUC * EF = UCP
- 61.2 * 1.04 = 63.648
    - or UUCP * TCF * EF = UCP

      60 * 1.02 * 1.04 = 63.648

3. ESTIMATING EFFORT

3.1 Calculating Man-hours from UCP

Translating use case points into man-hours per UCP is a matter of calculating a standard usage or effort rate (ER) and multiplying that value by the number of UCPs. In our case, our historical data sets the project man-hour effort rate at 28 man-hours per UCP. And, our experience agrees with Schneider and Winters[1] in that not all projects are equal, so we differentiate simple from complex projects by using 20 man-hours per UCP on simple projects.

Count the number of factor ratings of E1-E6 that are below 3 and the number of factor ratings of E7-E8 that are above 3. If the total is 2 or less, then use 20 man-hours per UCP. If the total is 3 or 4, use 28 man-hours per UCP. If the total is 5 or more then consider restructuring the project team so that the numbers fall at least below 5. A value of 5 indicates that this project is at significant risk of failure with this team.

Calculate man-hours by multiplying UCP by the effort rate:

- ER * UCP = total man-hours

  20 * 63.648 = 1,272.96 man-hours

  or 28 * 63.648 = 1,782.144

3.2 Adjusting Man-hours for Risk

Up to this point, the UCP estimating model has considered factors that affect every project in general. However, it is likely that each project will have specific influencing factors unique to that project. The Risk Coefficient is a project specific modifier applied as a ratio to the estimate to accommodate risk factors that are not inherently incorporated elsewhere in the model. Such risk coefficients might include identified risk for special training, out of normal development for reusability, or special system level integration requirements. Note: the risk coefficient might also present an opportunity for management to up the estimate to assuage their gut feeling. Try to resist this temptation, because it will eschew the value of the historical data feedback process.

Risk Coefficients are a specific addition to the original model by Agilis Solutions/FPT Software and not defined in other publications. Agilis Solutions/FPT Software use risk coefficients to adjust an estimate when a project includes specific risk that cannot be compensated for within the model itself. Process controls have been put in place to prevent over zealous use.

Identify the assumptions and apply a coefficient as a percentage. Multiply total man-hours by the percentage change:

- (1.0 + .xx[%]) * total man-hours = adjusted man-hours
- i.e., Increase estimate by 5% for a new reuse effort:

(1.0 + .05[%]) * 1,782.144 = 1,871.25 (adjusted man-hours)

3.3 Estimating for Reports

Agilis Solutions/FPT Software deviates from the UCP estimating model when calculating our estimate for reports. Calculate effort for reporting output directly into man-hours, rather than use UCP. This is because development of reports typically does not involve the same level of programming complexity as the rest of the application. Additionally, report creation often has a repetitive aspect to it that is easily leveraged and therefore would eschew UCP estimates when mixed in with estimates for the application.

Begin by creating a comprehensive list (if possible) of the reports to be produced and classify each report for complexity (simple, medium, complex).

Table 5. Reporting Output

No.	Report Name	Complexity			Avail
		Simple	Medium	Complex	
1	Sales by $	X			X
2	COGS		X		X
3	Rev. by Dept			X	

Add up the number of reports for each complexity type and multiply the number of each report type times the average number of man-hours it takes to create one report of each type, using historical data to determine effort levels.

Table 6. Weighting Reports for Complexity

Report Type	Quantity	Average Man-hours	Sub total
Simple	20	12	240
Average	15	20	300
Complex	10	40	400
Total Report Man-hour Estimate			940

Total the number of reporting man-hours and add this total to the Adjusted man-hours calculation for the project.

- Adjusted man-hours + Reporting man-hours = total man-hours

  1,871.25 + 940 = 2,811.25 total man-hours

As it is throughout the UCP estimating process, the key to a good estimate for report development lies in the accuracy of the average man-hour calculation used per report type. And like UCP data, this average must be derived from empirical historical data, not gut feelings. Keep in mind, both when creating and gathering the metric and when evaluating the historical data, that report development time varies by platform and tools used.

4. THE FEEDBACK LOOP

Feedback into the estimating process centers around two processes: 1) extensive collection of empirical post-mortem data from each project, and 2) a detailed causal analysis of the data by the Software Engineering Process Group (SEPG) every six months. A project post-mortem is a rigorous process that works from multiple goals ranging from obtaining a documented record of understanding on what happened on the project; to personnel management and penalty/reward; to gathering data for analyses that will drive improvements to management and engineering processes, including the estimating model. The post-mortem and causal analyses are about the analysis of the data and applying lessons learned from that analysis, not an exercise in collecting data, which is actually gathered throughout the project life-cycle. For the purpose of validating the estimating model, anomalies that arise in the post-mortem of a single project or are noticed during the project but outside of the regular six month SEPG analyses process can also be promoted for special review. The overall process improvement cycle is identified below.

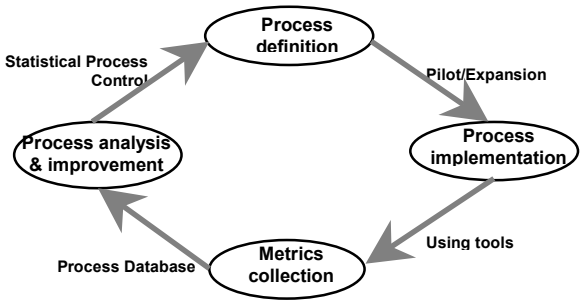

Figure 1. Process Improvement Cycle

It is important to understand that in the early years, we modified the model as events and trends indicated the need to change. However, through that process, we discovered that it was difficult to maintain consistency in our estimating process, and have therefore reverted to essentially the original model. We have not frozen the model, but we have decided that it is better to suffer changes to our development processes instead of changing the model. The model has not substantially changed in two years.

4.1 The Analysis Process

Analyzing for process improvement is a bit of an art applied to statistical process control. We collect significant quantities of metric data and conduct statistical analyses of various kinds. However the decision on whether or not to change a development process requires deep cause-effect justification which often reflect trends in human nature.

4.1.1 Performance Analyses

Measure performance data looking for a significant deviation between planned and actual in the calculation of UCP.

Table 7. Performance

Metric	Unit	Planned	Actual	Deviation(%)
Plan start date	DD-MMM-YY	29-Jul-04	29-Jul-04	0
Plan end date	DD-MMM-YY	29-Oct-04	N/A	15.22
Duration	Days	92	106	15.22
Max team size	Persons	12	14	16.67
Effort usage:	Person-day	420	245.98	-41.43
Development	%	60	56.84	-5.27
Management	%	7	5.36	-23.41
Quality	%	33	37.8	14.54
Product size	UCP	120.06	93.7	-21.96
Productivity	UCP/pd	0.29	0.38	33.26
Efficiency	USD/pd	0	0	N/A

Notice the anomalies: team size increased over plan from 12 to 14, yet actual effort is 41.43% below plan. Was this a case where adding more people got the work done faster? Then notice that the number of UCPs decreased 21.96% from 120.06 to 93.7, what happened there?

In any single project, the reason for deviation could be based on factors such as the maturity of the estimator, maturity of the team, miss calculation of actor, or use case or technical complexity; or the anomaly could simply be caused by a component of the project being canceled after the estimate process is long over.

4.1.2 Deliverable Analyses

Analyses of deliverable timeliness give commonly known indicators of where and when problems may exist in any project.

Table 8. Deliverables

Deliverable	Committed date	Actual date	Status	Deviation (%)
Claims Work-prototype	26-Aug-04	26-Aug-04	Accepted	0
Claims Work-source code and/or executable	15-Sep-04	4-Oct-04	Accepted	20.65
Claims Work-updated	28-Sep-04	N/A	Cancelled	N/A

In collecting deliverable data, we track to the half day. It may seem like nit-picking, but the % change deviation equalizes the differences; and the point of the metric is accuracy.

Deliverable data is then aggregated and compared with performance norms (in percentage of deviation) sliced by factors from timeliness through translation cost.

Table 9. Quality Objectives

Name	Unit	Norm	Targeted Value	Actual Value	Deviation (%)
Timeliness	%	85	100	80	-20
Requirement Completeness	%	95	100	100	0
Leakage	WDef/UCP	0.3	0.3	0.17	-43.08
Customer Satisfaction	Point	90	90	N/A	N/A
Correction Cost	%	9	7	9.81	40.11
Process compliance	NC	3	3	2	-33.33
Response Time	Hour	24	N/A	0	N/A
Translation cost	%	N/A	N/A	0	N/A

The key is to look for trends that should then be drilled down to causal affects. In The example above, the 20% improvement in time raises some questions; the 43.08% leakage will cause someone to ask if the QA team actually caught all of the defects; and the 40.11% correction cost will definitely raise a question.

4.1.3 Schedule Analyses

Analyzing schedule is a look at the macro view of the deliverable schedule; again, looking for anomalies that may require further research or explanation, such as why initialization went long in the project example below.

Table 10. Project Schedule

Stage	Is on time	Planned duration	Actual duration	Duration deviation(%)
Initiation	Yes	13	13	0
STAGE-Aug04	Yes	34	34	0
STAGE-SEP04	Yes	34	34	0
Termination	N/A	8	N/A	N/A

Products represent aggregated deliverables. Schedule anomalies point out potential issues or opportunities in the release processes.

Table 11. Product Schedule

Product name	Planned release date	Actual release date	Schedule deviation(%)
Claims Work-SRS doc	21-Aug-04	21-Aug-04	0
Claims Work-prototype	26-Aug-04	26-Aug-04	0
Claims Work-design	1-Sep-04	1-Sep-04	0
Claims Work-Test case & data	3-Sep-04	3-Sep-04	0
Claims Work-source code	15-Sep-04	4-Oct-04	55.88

Notice the 55.88% schedule delay in the code release, and then relate it to the previous overrun in correction cost.

4.1.4 Effort Analyses

To analyze effort by process, read down the list looking for anomalies in the percent of effort [Planned (%)] spent between one phase to the next. Do they look reasonable?

Table 12. Effort by Process

Process	Planned (pd)	Planned (%)	Actual (pd)	Actual (%)	Deviation (%)
Requirement	42	10	15.12	6.15	-63.99
Design	63	15	47.69	19.39	-24.31
Coding	138.6	33	95.56	38.85	-31.05
Deployment	12.6	3	2.25	0.91	-82.14
Customer Support	21	5	2.88	1.17	-86.31
Test	84	20	55.56	22.59	-33.85
Configuration Management	4.2	1	1.62	0.66	-61.31
Project planning	4.2	1	3	1.22	-28.57
Project monitoring	25.2	6	10.19	4.14	-59.57
Quality Control	12.6	3	9.1	3.7	-27.78
Training	8.4	2	0.88	0.36	-89.58
Total	420	100	245.98	100	-41.43

Compare the percent of total planned effort used in requirements to the percent of effort used in design and so on down the list. Then look for anomalies between planned percentage and actual percentage. For example, in the previous tables we learned that initialization took longer and requirements less time than planned. In the above table we see that requirements took much less effort than expected. Perhaps some key requirements were defined in the initial phase but it took some extra calendar time (not effort) to obtain final approval. A common enough occurrence, but if this proves to be the case across many projects, the trend might indicate a change in process or team training is warranted.

Table 13. Effort by Type

Type	Effort(pd)	Effort(%)
Study	9.75	3.96
Create	150.25	61.08
Review	15.1	6.14
Test	46.75	19.01
Correct	24.12	9.81
Total	245.98	100

Slicing the effort data by type of effort gives an indicator of either the maturity of the development team, or the difficulty of a new technology. Too much time testing and correcting might spot poor coding technique or sloppy reviews. Anomaly trends in effort by type can be handled by strengthening the review process, training personnel on better coding practices or techniques in technical problem solving. A change in the estimating model might be reflected in a new addition to the effort table or changes to weight

values to more accurately reflect the average level of maturity for the team at large.

4.1.5 Defect Analyses

Analyses of defects tend to point toward technical factors. The first place to look might be to analyze when the defects were injected into the system.

Table 14. Defect by QC activity and origin

Detection on	Injection on requirement	Injection on design	Injection on coding	Weighted defects	% of total
Requirement review	1	0	0	1	0.2
Design review	0	16	3	19	3.78
Code review	0	0	9	9	1.79
Unit test	0	0	37	37	7.36
Integration test	0	0	0	0	0
System test	0	26	332	360	71.57
Acceptance test	0	0	15	16	3.18
Total	10	68	407	503	100

In this analysis, compare when the error was injected against when it was found in order to point out anomalies in the bug detection processes. An important part of the QA process is to predict ahead of the phase the number of defects to be found during reviews, testing and QA activities. The intent is to be able to decipher between poor processes and poorly trained personnel, complex problems and difficult customers. Notice in the table above that the vast majority of code errors were found as late as system test. If this is a trend across projects, then improvements in code reviews and unit testing processes may be warranted. Further research may indicate a change in the weighted technical factors for the type of technology involved. A pervasive trend might force a change to the technical factor weights, as a way to force more time into the estimates for a longer testing phase.

Table 15. Defect by Severity

Severity	Number of defects	% of total	Number of defects by review	% of total by review	Number of defects by test	% of total by test
Fatal	0	0	0	0	0	0
Serious	17	6.69	1	4.17	15	7.14
Medium	98	40.17	12	50	84	40
Cosmetic	124	51.05	11	45.83	111	52.86
Total	239	100	24	100	210	100

Similar to defect by origin, defect analyses by severity typically points to anomalies affecting technical factors. Since this data is generalized, anomalies in this analysis will require more research to dig into the nature of the defects themselves before making final conclusions. However, trends might point to problems in coding processes, complex design issues, or review problems.

Table 16. Defect by Type

Classification type	Fatal	Serious	Medium	Cosmetic	Weighted Defect	% of total
Functionality (Other)	0	1	15	5	55	10.93
User Interface	0	0	4	80	92	18.29
Performance	0	1	0	0	5	0.99
Design issue	0	0	1	8	11	2.19
Coding standard	0	0	0	0	0	0
Document	0	0	2	8	14	2.78
Database integrity	0	0	1	0	3	0.6
Portability	0	0	0	0	0	0
Req mis-understanding	0	2	0	0	10	1.99
Feature missing	0	1	6	1	24	4.77
Coding logic	0	12	52	16	232	46.12
Business logic	0	0	17	4	55	10.93

Classifying the type of defect further drills down on which process to improve. Notice that the classification types are specifically referenced, in order to separate engineering processes from an isolated project problem.

Table 17. Defect Prevention Goals

#	Item	Unit	Planned	Actual	Deviation
1	Coding logic defect rate:	WDef/UCP	1.9	2.47	0.57
2	Defect rate:	WDef/UCP	5.5	5.37	-0.13
3	User Interface defect rate:	WDef/UCP	0.65	0.98	0.33

Referring to the [planned] defect prediction rate above, how many engineering teams are there that predict their defect rate ahead of testing, let alone using the data to indicate where improvements might be found in their processes. The goals identified change from one project to the next, depending on the project situation. This project required a large number of end-users be heavily involved in the project.

4.1.6 Causal Analyses

Causal analyses are those made by the team leadership, giving the team a chance to explain anomalies in their own words. Team leaders are trained in how to look objectively at the work of their team and to focus on the process, not the person. Since everyone is rewarded when processes work well, and everyone is involved in continuous process improvement, team leads are motivated to be objective in their own analyses and not be defensive or take things too personally when issues come up during a project implementation.

Table 18. Causal Analysis

Name	Unit	Planned Value	Actual Value	Deviation from norm(%)	Cause
Defect rate	Wdef /UCP	110	58.57	-46.76	Use of the development wizard generated better code then expected
Size achievement	%	120	78.04	-34.96	1. Claims Work - updated was cancelled 2. Inventory 1 & 2 were not as complicated as estimated.

Notice the explanation of the low defect rate due to the use of a development wizard. Elsewhere (not shown) the team complained that the same wizard limited their ability to unit test.

Table 19. Team Evaluation

Name	Role	Process	Skill & Technology	Point
Joe	Developer	Coding	Coding Web form using VB.NET	5
	Developer	Coding	Coding crystal report	4
	Developer	Test	Unit test	3
Tom	Developer	Coding	Coding crystal report	5
	Developer	Coding	SQL server, store procedure	4
Sue	Tester	Test	Prepare test case and test plan	4
	Tester	Test	Testing web application	4
Sally	Developer	Coding	Coding web form using VB.NET	4
Lee	Project Director	N/A	N/A	0

Team data can explain anomalies in earlier material and documents the assumptions made in the experience section of the estimating tool.

4.2 Quantitative Management Analyses

When trying to discuss metric gathering and analysis, it is more understandable to view the specific metric data than simplified graphs. For this reason, the examples given thus far have been from a single project. However, single projects do not reflect trends that might prevail across teams. Every six months, the SEPG aggregates detailed project data from all projects completed during that term into quantitative management graphs. Quantitative management graphs (such as that below) easily highlight anomalies: outside of predetermined limitations (9% for upper and lower control limits), one standard deviation (5.2% below), and showing clear deviation trends. All projects are included in these analyses. High and low points crossing out of limitation indicate anomalies that require further explanation. Trends are noticed in the slope of a line or in multiple results consistently falling outside of limits.

5. CONCLUSION

Perhaps the reader will notice that there are no easy systematic answers offered. The decision of whether to change a work process, provide employee training, modify the estimating tool, or some other action (no action is an action), is often subject to the judgment of the evaluator. Our estimating tool is still fairly close to the model published in *Applied Use Cases [1]*. This is intentional, in order to preserve the integrity of the model. A constantly changing model is difficult to use consistently, and consistency is our life-blood. Our ability to accurately predict the cost of a project early in the project lifecycle is a major differentiator for our business. Therefore, we suffer continuous process improvement to our processes and training, and evolve our estimating model very slowly. A good motto for our team might be: *continuous process improvement – as we learn, we improve – as we improve, we evolve.*

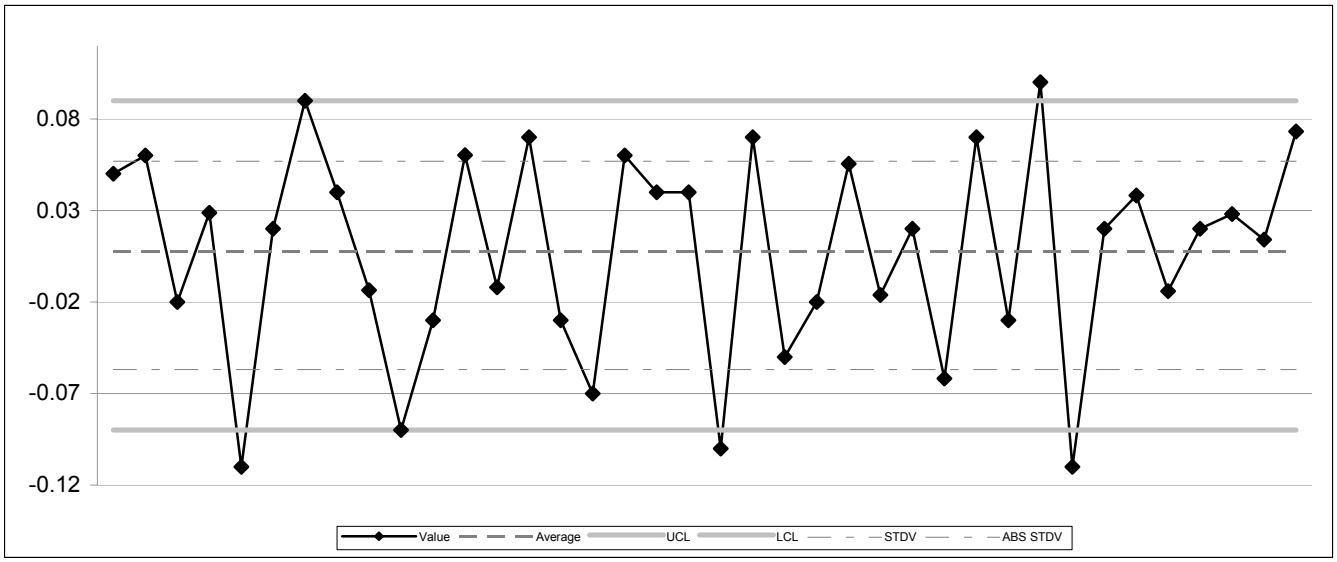

Figure 2. Quantitative Management Graph

Simply collecting metrics does not guarantee success. The metrics collected must be specifically relevant to improving the engineering processes; which cannot be effectively done until those engineering processes are standardized and well managed. Collecting data, devoting time to analysis, and seriously looking for ways to improve processes are not trivial activities; however one clear benefit is a very accurate parametric estimating tool.

The decision on whether to improve a process, educate the team, or change the estimating tool is handled by the Steering Committee for Quality Assurance. The committee seeks to balance changes in process against education. In the past two years, there have not been any changes to the estimating model, demonstrating the stability of the model (and our continuously improving processes).

6. REFERENCES

Thanks to FPT Software Corporation for a partnership that goes well beyond simply working together, but encourages continuous process improvement in every aspect of our relationship.

7. REFERENCES

[1] Schneider, G.,Winters, J.P., *Applied Use Cases, Second Edition, A Practical Guide,* Addison-Wesley, Reading, MA, 2001.

[2] Cockburn, A., *Writing Effective Use Cases,* Addison-Wesley, Boston, MA, 2001.

[3] Rumbaugh, J, Jacobson, I., and Booch, G, *The Unified Modeling Language Reference Manual,* Addison-Wesley, Boston, MA, 1999.

[4] Brooks, F., *The Mythical Man-Month, Anniversary Edition* Addison-Wesley, Boston, MA, 1995.

[5] Paulk, M., Curtis, W., Chrissis, M., Weber, C., *Capability Maturity Model for Software, Version 1.1,* Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.