Measurement for improving accuracy of estimates: the case study of a small software organisation

Sylvie Trudel

Abstract

This paper describes the case study of a small Canadian software development organisation that has established and sustained a measurement program for its software activities, which includes functional size measurement using the COSMIC-FFP method. This company has been in operation for over 20 years, and has 11 employees, all directly involved in software projects. Their "not to exceed" estimate business model, guarantees that fixing all defects found by their customer are free of charge. Quality is absolutely not an issue since they deliver a new software release to their major customer every other week with less than one defect per release on average, which is usually fixed within three hours. For that reason, they do not require a defect management system. Thus, the motivation for a measurement program came from other issues such as the inaccuracy of initial estimates, commitment to quality and productivity by applying best practices and continuous improvement, and the desire to improve productivity due to the loss of potential contracts to offshore organisations.

Many challenges were encountered and resolved while implementing the measurement program such as dealing with company growth, tuning of estimation models to improve accuracy between initial estimates and project actual performance data, and applying the required rigour to sustain measurement activities.

This paper also describes a simple measurement plan. Measurement results presented were used to improve the accuracy of estimation models, in which the step-by-step approach is described. With more accurate estimates, several sound business decisions were made regarding future projects.

1. Introduction

For many involved in software projects, accurate estimation is still perceived as an art, despite the fact that several estimation methodologies have been developed and published over the last years. Of course, the success of an estimation methodology implies collecting and analyzing accurate project measures that are later used in an estimation model. Some of these measures, namely effort and size, are known to be highly correlated as development effort is dependant on the software size. But these two factors alone do not guarantee accuracy of an estimation model and an organisation must stop and reflect in order to understand the other factors influencing the productivity of a software team. Adding factors to an estimation model may look like it increases its accuracy, but it could make it less accurate due to the error propagation inherent with each factor used [1].

Part of this thought process was made in a small Canadian software development company in order to come up with a more accurate estimation model. The need for accuracy came from their business model. The factors that were examined came from their product technology and their software process. This paper describes what was done and measured to improve the accuracy of their estimation model.

2. Company description

2.1. Company overview

The small Canadian company was started 22 years ago by its president who is now acting as project manager. All of the 11 employees are developers, two of which are also analysts responsible for requirements development. They have no overhead but accounting (one day per week) and housekeeping are subcontracted. The company has up to six active customers, one of which is a large financial organisation managing loans for assets acquisition that makes up for an average of 80% of the company's annual gross revenues. For that customer alone, the company develops and maintains a series of systems to support their sales and operations. One of these systems, a sophisticated ERP called SUM, is interfaced with 10 peripheral systems and has been developed and used for over 10 years. Maintaining and developing new features in SUM required seven person-years in 2006.

They have a backlog of projects roughly defined and planned to keep the whole team busy for the next 6 to 8 months.

2.2. Business model

The company's business model is to provide each customer a "not to exceed" project estimate. If it takes less effort to develop a project than what was estimated, the customer is invoiced at a lower price. When it takes more effort to develop than what was estimated, the customer is invoiced the estimated price. Therefore, there is a strong motivation for accurate estimating because, if the estimation is too high, the customer may decide to outsource the project elsewhere, a situation which has occurred in the past. As well, if the estimation is too low, the company does not make any profit and may even have to absorb a loss.

In addition, any defect found is to be fixed at the company's expense. Therefore, there is a strong commitment to quality and it became a company goal to deliver defect free products, which can also be seen as a competitive advantage.

Every hour spent performing activities during "Initiate project" and "Analyze and estimate" (see section 3.2 for process overview) is always billed to the project. As a result, the effort estimates must include all other types of activities: project management, software development and testing, documentation, packaging, and validation. Different hourly rates are used per activity depending on the skill sets required, i.e. individuals performing project management or analysis activities have significantly higher wages.

3. Process description

3.1. Process improvement initiative

The company missed deadlines on several features given their short bi-weekly release cycle. They experienced cost overruns on half of their projects. Quality was not an issue since they usually have less than one defect per release, found once the release is in production and fixed within a three hour period. Nevertheless, some of their potential projects were lost to major outsourcing organisations in India from 2001-2002 and they needed to be more competitive. They had learned about the existence of the CMMI [2] and were concerned about applying its best practices to increase their efficiency and productivity.

In November 2004, the company formally began to continuously improve its software process by hiring the author as an external consultant. They had a stable but undocumented process. They were not facing any big issues related to software development but small irritants were observed, namely in software development estimation and lack of formality in customer communication. The path adopted was to learn about the CMMI, assess their practices against CMMI practices at a rate of one process area per month, and start improving their process on a continuous basis.

The demand for features from their largest customer was growing, so was the team size. They wanted to continue to be results-oriented and needed to involve more team members into project coordination.

The process was and still is project-oriented. They define a project as being a set of one or more related features laid out to develop or modify a part of existing or new modules. The average project effort is approximately 150 hours; some bigger projects attain more than 1,300 hours.

They document user needs and requirements in simple text files which include interface mock-ups. They apply the Scrum methodology [3] for managing their projects and related detailed requirements. They use spreadsheets to gather planning data, design decisions, and test cases. Peer reviews are applied to selected deliverables. Effort is measured using a home-grown timesheet system called eMalaya.

3.2. Process overview

The company's process has nine phases (Figure 1), each of them being detailed in a set of activities directly producing or updating an output.

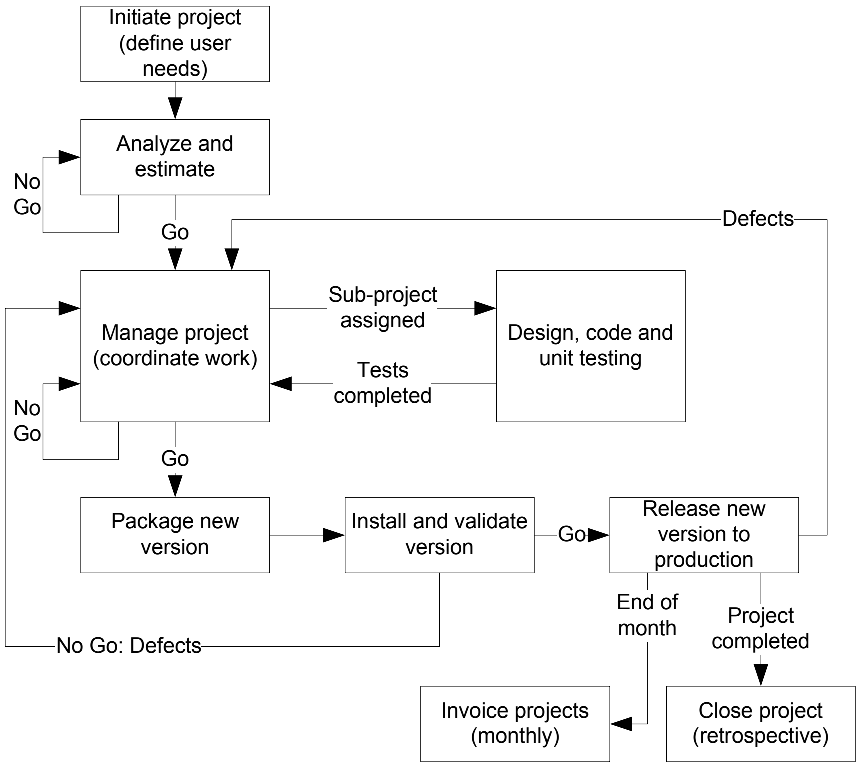

Initiate project (define user needs)
Analyze and estimate
No Go
Go
Defects
Manage project (coordinate work)
Sub-project assigned
Tests completed
Design, code and unit testing
No Go
Go
Package new version
Install and validate version
Go
Release new version to production
No Go: Defects
End of month
Project completed
Invoice projects (monthly)
Close project (retrospective)

Figure 1. Process overview.

3.3. Measurement program

The company already had a measurement program that included effort and schedule measures, and its main purpose was for billing the customers at the end of every month and at the end of the project and tracking R&D. However, no measurement plan existed that related the measures to the business goals because some of the required measures and indicators were not available. In the fall of 2006, it was decided to document the measurement plan as an exercise to understand the information needs of the manager and team members.

In small companies, when a new practice is introduced in the process, such as developing and maintaining a measurement plan, its advantages must be clear for the manager; otherwise he or she may decide quickly to abandon this new practice, especially if it seems cumbersome. So a very simple approach for documenting it had to be taken. It was decided to use the classic "Goal-Question-Metric" (GQM) technique and to put the measurement plan into a spreadsheet with three worksheets in it: one for the goals (Table 1), one for the questions and indicators (Table 2), and one for basic measures (Table 3).

Table 1. Company goals.

ID	Goals	Reason
G1	Deliver projects within effort estimates	Reach corporate goal of 30% gross margin.
G2	Deliver defect free versions into production	Ensure product quality and customer satisfaction, minimize rework.

One of the challenges the manager and analysts encountered at the beginning of the measurement program was the continuous rigour and discipline required to feed the measurement program. At first, the main file used to monitor all projects, the project portfolio file, did not contain any of the measures or indicators and this data was simply kept in its source repository. It was decided early on to include these measures and indicators in the project portfolio file so the manager would be able to view a whole year of projects at a glance. Only then did the motivation to sustain the measurement program arise, because it looked simple, useful, and they knew exactly why these measures were taken and what decisions or actions should be taken based on indicator values.

4. Product description

4.1. Product overview

SUM is deployed in 14 locations throughout Canada and is utilized by approximately 250 users on a daily basis. Three servers are used simultaneously to provide the required performance: Montreal (Quebec), Toronto (Ontario), and Calgary (Alberta). It is built on client-server architecture for Windows.

Ten years ago, SUM was fully developed in Visual Fox Pro (VFP), including its database (see Figure 2 for Previous SUM Architecture). In 2003, the database was changed to MS-SQL Server, which was more secure and reliable than VFP, and they began a six-year reengineering plan in .Net C# (for both Windows and Web user interfaces), which includes refactoring the business logic and the data into separate layers to accommodate various interface types (Windows, Web, Mobile) without duplicating important portions of the source code that also requires more intensive testing (see Figure 3 for New SUM Architecture).

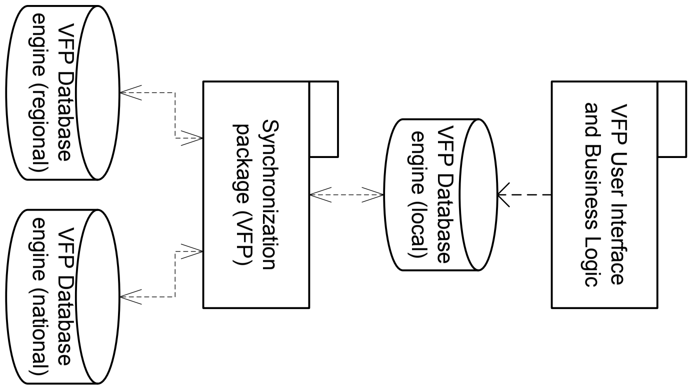
Today, SUM is made of 310 windows and screens distributed in 11 internal modules and 10 external modules; it has over 1 million lines of code (see Table 4) and 474 database tables containing a total of 4,333 fields (see Table 5).

Table 2. Example of questions and indicators (related to goal #1).

ID	**Q1**	**Q2**	**Q3**	**Q4**
Indicators (questions)	For each project, what is the difference between actual effort and planned effort?	What project proportion has an overrun > 5%?	What are the differences between the planned effort and the initial Scrum detailed estimate?	How many defects do we have per year and per release?
Formula	(Actual effort - (planned effort + CRs))*100 /(planned effort + CRs)	(Number of projects of overrun>+5%)*100/ total number of projects	Planned effort – Scrum initial effort	Number of defects per release and total
Goal	G1	G1	G1	G2
U of M	%	%	Hour	Unit
Source of data	Project Portfolio file	Project Portfolio file	Project Portfolio file	Version file
Responsible	Manager	Manager	Scrum Master or PM	PM
Stored where	"Effort overrun" column	On top of the "Effort overrun" column	Separate "Scrum effort" column	"Defects" column
Stored when	End of every project	Quarterly	As soon as Scrum detailed estimated is done	After every release
Stakeholders	Manager+PM+Analyst	Manager+PM+Analyst	Manager+PM+Analyst	All
Analysis Procedure	Use conditional formatting to highlight any overrun.	When > 15%, highlight in red.	When Scrum estimate exceeds planned effort of more than 5%, highlight in red.	When > 1, highlight in red.
Possible Actions	- Verify that the process was applied, especially on CRs. - Verify any encountered issue.	When > 15%, adjust estimation model	- Re-estimate either plan or Scrum. - If appropriate, advise customer of an estimate change prior to beginning project.	When > 1, do a retrospective.

Table 3. Some of the basic measures (related to goal #1).

ID	**M1**	**M2**	**M3**	**M4**
Measures	Actual effort	Planned effort	Total effort for all CRs	Scrum initial effort
Scope	Per project	Per project	Per project	Per project
U of M	Hours	Hours	Hours	Hours
Precision	1 hr	1 hr	1 hr	1 hr
Measured by	Employees	PM	PM	Employees
Data source	Anatime	Project plan	CR files	Scrum Works
Data collection procedure	Timesheet must be entered every day	Project < 50 hrs = manual only Project > 50 hrs = FSM	As soon as a CR is approved, enter it in the CR Follow-up table in the project plan.	As soon as Scrum initial effort is completed, the PM copies the effort value in the project portfolio file.
Quality assurance	Verify timesheets prior to invoicing (monthly): right project, right task, consistent number of hours.	Peer review estimates to ensure that nothing was forgotten. Validation with customer.	Peer review estimates to ensure that nothing was forgotten. Validation with customer.	Review by the PM.



VFP User Interface and Business Logic
VFP Database engine (local)
Synchronization package (VFP)
VFP Database engine (regional)
VFP Database engine (national)

Figure 2: Previous SUM Architecture.

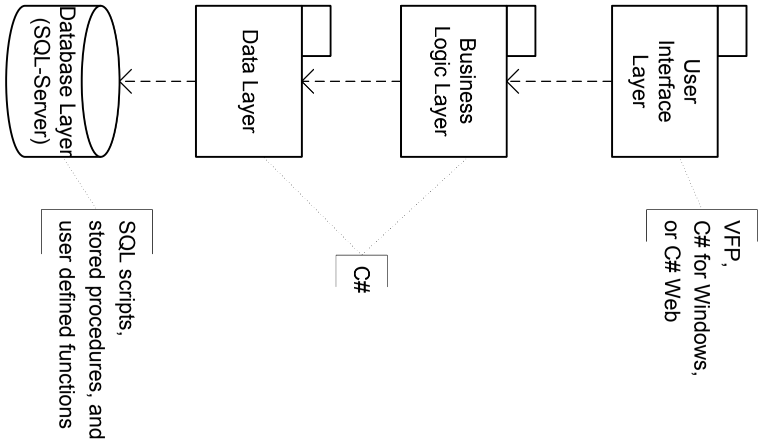


Figure 3: New SUM Architecture.

Table 4: Source code physical measures.

Language	# lines, incl. comments	# commented lines	Comments ratio
VFP	541,288	211,071	39%
.Net C#	484,082	121,391	25%
SQL[Note 1]	100,098	18,800	19%
Total:	**1,125,468**	**351,262**	**31%**

Note 1: The database layer contains 398 Stored Procedures for 73,515 lines of code and 999 User Defined Functions for 20,327 lines of code.

Table 5: Database structure physical measures.

	Dynamic[Note 2]	Static	Total
Number of tables	172	302	**474**
Number of attributes (fields)	2,751	1,582	**4,333**

Note 2: The content of Dynamic tables is synchronized every 15 minutes between the Montreal, Toronto, and Calgary servers, for redundancy and security reasons.

4.2. Product release cycle

The product is released into production every other week. Supplemental releases may be required for bug fixing or the addition of an urgent feature. A new version releases several features from several projects. A project can also be developed iteratively and delivered over several product releases, in this case hidden or partially deployed project can be included in a release.

For every release, the build master compiles a new version from the source code repository, installs it on the test environment and tests are performed for half a day (Thursday morning). If defects are found at this point, verbal communication occurs with the developer in charge who fixes the defect immediately, stores the modified code into the repository, and advises the build master who rebuilds a new version. These defects are not measured.

When tests results show no defects, a readiness for validation notice is sent to their customer. Two team members supervise validation testing at the customer's site during one to four hours (Thursday afternoon). At this point, any defect found is fixed Friday morning and the product is retested in the afternoon. These defects are not measured either. Once validated, the new version is released into production by the customer's IT staff, using the installation procedure during the weekend. When defects are found after it has been sent to production, they are noted and measured in the "Version file". These defects are fixed immediately and a new version is sent before noon that day.

On average, fixing defects takes one hour, building the application requires 10 minutes, and deployment requires 30 minutes. This efficiency is possible using a home-grown deployment tool.

4.3. Product quality

Delivering defect free products is one of the company's main goals. In 2006, 35 releases of SUM were deployed, of which 17 had zero defects. In the other 18 releases, 28 defects were found and each of them was fixed within half a day. It represents an average of 0.8 defect per release. The main reasons for these astonishing results are: a robust software architecture, in-depth knowledge of the business domain, the framework used, and strong commitment to quality from management.

Considered as a rare situation, this company does not use any defect management tool because there are simply not enough defects to manage.

5. Project estimation

5.1. Initial estimation process

Before 2005, estimation was done on a task-effort basis. During analysis, a list of tasks was detailed to define the software work needed to accomplish the project in the four main software layers: user interfaces and reports, business logic, data and database. Every task was estimated by the analyst and a developer would validate the list of tasks and associated design and programming effort. A percentage was then added for testing and project management.

As a result, half of the projects ended up exceeding estimates.

5.2. Functional size measurement

Functional size measurement (FSM) using COSMIC-FFP [4] was introduced in the fall of 2005 as an ingredient to develop a new estimation model. FSM was then performed on 3 to 4 projects. In December 2005, as the "Guideline for sizing business application software using COSMIC-FFP" [5] was published, validation was applied to actual functional size and modifications were made resulting from a better understanding of the COSMIC-FFP method.

The project plan template was modified to add a worksheet for FSM, which contains the list of functionalities with their associated data groups. The analyst was responsible to measure the size of current and new projects. When using the project plan template, the analyst identifies each functionality, its associated data groups and relevant data movements. Functional size is automatically calculated based on identified data movements.

5.3. Early estimation models based on FSM

A productivity model was established based on functional size and actual effort. It was mostly used to validate estimates made on a task-effort basis. Significant differences in productivity models were observed among projects, ranging from 1.5 to 6 hours per size unit. Variation between estimates and project performance was sometimes more than 50%. This inaccuracy makes a big difference for a 150,000$ project vs. a 600,000$ project, where the potential customer would have accepted the first, but refused the second.

5.4. First observations on inaccurate results

It was no surprise to find out that the most important factor for differences between effort estimates and projects actual effort was due to change requests (CR) that were not systematically estimated and measured. A so called "small request" made by the customer over the phone can become a 45% effort increase. A decision was taken to formally document any CR for approval by the customer and to monitor all CRs of a project in the project portfolio file. Also, effort for implementing CRs is entered in the timesheet system with specific description in order to isolate that effort easily.

When effort from implementing CRs in a project was extracted, variations between actual effort and estimated effort were less than 27%. However, it was still significant and research was undertaken to improve accuracy.

6. Improving estimation models

Intuitively, the analysts and project manager believed in the FSM approach for effort estimation and decided to refine the concept of measuring functional size without having to switch to the developer's viewpoint. The following sections describe the detailed steps that were taken to come up with more accurate estimation models.

6.1. Step 1: assess reasons for inaccuracy from product and process

Investigation helped explain this variation and it appeared that the main difference came from the technology used: VFP projects productivity averages 2.5 hours per size unit, and C# projects productivity averages 4.5 hours per size unit, once they thought the learning curve of team members was over. One of the difficulties was when several technologies (i.e., VFP for GUI and C# for business logic) needed to be integrated because it is more complicated and requires more effort, up to 6 hours per size unit. Also, a lot of effort had to be spent on creating stored procedures for new database tables. Many projects developing new features were actually using existing tables along with their stored procedures. In these cases, the required effort per size unit was lower than the productivity model of projects for which most tables needed to be created.

The team productivity model behaved as if new development occurs only for the first functional process (i.e., creating GUI, business logic, data persistence, and database layers) and as if it switches to maintenance mode thereafter (i.e., stored procedures are already created and used in an evolutive or adaptative software maintenance).

Training was provided by an expert on estimation based on FSM in fall of 2005. This expert described the following estimation ratios for software maintenance:

- Add a new data movement = 100% of effort
- Delete a data movement = 10% of effort, mainly required for testing
- Modify portions of an existing data movement = 50% of effort.

These ratios were applied on the estimation model itself (e.g., if the estimation model is 5 hours per size unit, deleting a data movement would require 0.5 hour and adding one or several attributes to a data group would require 2.5 hours for each affected data movement.) The functional size remains the same; an adjusted estimation model is simply applied on each data movement. The analysts tried to estimate using this technique but felt uncomfortable with it for the following reasons:

1. This method seemed appropriate only when the software architecture is in a single layer because it implies that all data movements related to that new data group are also new and have to be fully developed.
2. When developing in a multi-layer architecture, a new data group requires more effort to create when developing the first functional process, and less effort when reusing that data group data layer code and business layer code in any subsequent functional process.
3. When modifying existing data groups and data movements, such as adding attributes, there is a significant difference of effort due to the number of attributes affected, and thus the 50% ratio for maintenance needed to be redefined.

Using the developer's viewpoint was considered for FSM but the idea was quickly abandoned for fear of increasing measurement effort by having to measure all data movements for each of the software layers. Instead, it was decided to continue using the user's viewpoint but try to establish the impact of reuse on each data movement.

6.2. Step 2: evaluate impact of reuse from software architecture layers

In depth knowledge of the development process tied to the software architecture led the analysts to evaluate the impact of in-process reuse of basic software components. The analysts measured the ratio of effort required to develop several new functional processes requiring all new data groups. For example, such functional process could be a screen to assign an employee to a department. On average, half of the effort is spent developing the business and data layers in C#, 20% is spent on developing the database layers (SQL stored procedures and user defined functions), and the rest is for the user interface for which effort may vary depending on the technology used. The technology issue will be addressed later.

When a second functional process is developed, such as displaying the list of employees with their current department, all required components from the database layer and many of the components from the business logic and data layer already exist, so the developer simply has to reuse them. In such case, the display (exit) of an employee/department record is considered as a new data movement, but reading the employee, employee-department, and department tables is done with reusable components, for which minor adjustments in the business logic must be done, along with required effort in the user interface for selection or filtering criteria.

A function that requires a minor change is defined as adding one to three attributes in an existing data group, affecting all relevant data movements (e.g., adding a customer web site link in a user interface, which requires to add the web site field in the customer table, add the business logic to validate format of the web site link, and add the field in the user interface to be displayed.) All components exist but affected data movements require small modifications throughout all layers. The same principle applies for a major change defined as adding more than three attributes to an existing data group, affecting all relevant data movements (e.g., adding a shipping address and shipping phone number of customers on the "display invoices" function.) Example of obtained results is shown in Table 6.

Table 6. Effort ratio per software layer.

	Effort ratio			
Software layer	**New**	**Reuse**	**Minor change**	**Major change**
User interface	30%	15%	10%	30%
Business logic and data (C#)	50%	10%	10%	30%
Database layer (SQL)	20%	0%	10%	10%
Total:	**100%**	**25%**	**30%**	**70%**

6.3. Step 3: apply reusability factors to data movements

Taking into account the reusability allows "weighting" of each data movement (i.e., calculating a fraction of its size by applying the appropriate effort ratio: new=100%, reused=25%, minor change=30%, and major change=70 %). This activity is performed as part of FSM and is automated in the estimation spreadsheet, as shown in Figure 4.

			Total :	13	14	13	12	52	-9,75	42,25
Module	**Functional Process**	**Data Group**	**Reuse type**	**FFP Read**	**FFP Exit**	**FFP Entry**	**FFP Writ e**	**FFP Total**	**Reuse Impa ct**	**Weighted size**
Create email/fax	Display main window	Trigger	New	0	0	1	0	1	0	1
Create email/fax	Display main window	Document Header	Reuse	1	1	0	0	2	-1,5	0,5
Create email/fax	Display main window	curDocHe ader	New	0	0	1	1	2	0	2
Create email/fax	Display main window	Error message	New	0	1	0	0	1	0	1
Create email/fax	Maintain address book	Contacts	Reuse	1	1	1	1	4	-3	1
Create email/fax	Maintain address book	curContac t	New	0	0	1	1	2	0	2
Create email/fax	Maintain address book	Error message	New	0	1	0	0	1	0	1
Create email/fax	Maintain address book	UserLogo n	Reuse	1	1	1	1	4	-3	1
Create email/fax	Maintain address book	curUserLo gon	New	0	0	1	1	2	0	2
(File continues…)										

Figure 4. Example of FSM and weighted size.

This reusability factor is applied during FSM and requires only one to two seconds per data group per functional process, which is negligible knowing that an average size project requires approximately 1.5 hour to measure.

6.4. Step 4: establish estimation models per technology

To establish an estimation model based on the weighted size, several projects were measured, weighted, and then compared with actual effort. At mid-2006, the initial estimation models based on weighted size units (WSU) per technology were established (see Table 7).

Table 7. Initial estimation model based on weighted size per technology.

Technology	**VFP**	**C# for Windows**	**C# Web**
Estimation model (hours/WSU)	3.22	3.86	5.15

These estimation models were not readjusted until recently when it was decided to follow-up on their accuracy. Three C# projects and two VFP projects were estimated and measured using the technique described above (see Section 6.6).

6.5. Step 5: adjust effort estimation with risk factors

The analysts and project manager also discovered three risk factors that were influencing their productivity on certain projects: technology (known or unknown), complexity (low, medium, high) related to knowledge of business domain and business process of the customer, and number of other stakeholders involved (none, third party from the client, one or many vendors).

Contingency is added as a percentage of total estimated effort when risk is perceived. No contingency is required when the technology is known, the complexity is low, and no third party or vendor is involved. The sample of projects used in this case study did not require any contingency, so as the majority of developed projects.

6.6. Step 6: validate effort estimation with actual data

The last step was to enter actual data in the project portfolio spreadsheet: actual functional size, estimated and actual effort, change requests effort, and weighted size. Then, automated computations were done for productivity (actual hours per functional size unit) and estimation models (actual effort per WSU), the latter being monitored to readjust the overall estimation model on a periodic basis (see Table 8).

Table 8. Actual performance data for a sample of projects.

Project #	Technology	Funct. Size (FFP)	Weighted size units (WSU)	Original effort estimate (hours)	Actual effort w/o CRs (hours)	Overrun %	Productivity model (Hr/FFP)	New estimation model (Hr/WSU)
1	C# Win	218	159.0	598	567.4	-5%	2.6	3.6
2	C# Win	74	53.3	131	109.7	-16%	1.5	2.1
3	C# Win	124	89.5	223	236.9	6%	1.9	2.6
						Average for C# Win:	**2.0**	**2.8**
						Variance for C# Win:	**0.3**	**0.6**
4	VFP	47	42.0	102	78.7	-23%	1.7	1.9
5	VFP	66	55.5	155	138.3	-11%	2.1	2.5
						Average for VFP:	**1.9**	**2.2**
						Variance for VFP:	**0.1**	**0.2**

7. Preliminary results of the "weighted size” approach

The number of data points was not sufficient to conclude if the weighted size approach is successful or not to gain accuracy between estimates and actual effort. However, these preliminary results were observed from the measurement exercise:

- The average productivity for C# Windows projects went from 4.5 to 2.0 hours per COSMIC-FFP functional size units, which is very good compared to similar projects found in the ISBSG repository [6]. This can be explained by three factors: i) the learning curve for the C# technology may not have been over last year when the initial estimation model was created; ii) six months ago, the manager has dismissed an employee perceived to be a “net negative producing programmer” [7] and, after that employee left, the manager perceived an increase of the overall team productivity; iii) the software process is applied consistently.
- The productivity difference between C# for Windows and VFP projects seems to have decreased significantly, which may open new business opportunities.
- There seems to be a tendency to overestimate, which is desired to a certain extent, due to the business model.

8. Conclusion and future work

It is no surprise to observe that inaccurate projects are those for which the team did not have the discipline of formalizing change requests. Even if it is unrealistic to desire an estimation model that is 100% reliable, early refinement results for C# projects with less than 16% variance are encouraging. Several other projects were being developed and measured at the time this article was being written, and the organisation will continue to monitor their actual performance data in order to readjust the estimation models on a periodic basis. However, if the weighted size approach does not result in an increased accuracy, it may be abandoned to continue using the model of hours per COMIC-FFP functional size unit.

9. Acknowledgements

The author wishes to thank Michel Martel, president, and Stéphan Laporte, analyst of Analystik inc., (michel.martel@analystik.ca; stephan.laporte@analystik.ca) for their precious collaboration and information sharing. Special thanks to Carmela Caterina, president of the Montreal SPIN, for her comments that improved the written English of this text.

10. References

[1] Santillo, L., “Error Propagation in Software Measurement and Estimation”, in IWSM/Metrikon 2006 conference proceedings, Potsdam, Berlin, Germany, 2-3 November 2006.

[2] Chrissis, M.B., Konrad, M. and Shrum, S., “CMMI: Guidelines for Process Integration and Product Improvement”, Addison-Wesley, the SEI Series in Software Engineering, Boston, 2003.

[3] Schwaber, K., “Agile Project Management with Scrum”, Microsoft Press, Redmond, WA, 2004.

[4] Abran, A. e.a., “COSMIC FFP Measurement Manual 2.2”, January 2003, http://www.lrgl.uqam.ca/cosmic-ffp .

[5] Lesterhuis, A. and Symons, C., “Guideline for sizing business application software using COSMIC-FFP”, the Common Software Measurement International Consortium, version 1.0, December 2005, http://www.lrgl.uqam.ca/cosmic-ffp .

[6] International Software Benchmarking Standards Group, http://www.isbsg.org .

[7] Schulmeyer, G.G., “The Net Negative Producing Programmer”, http://www.pyxisinc.com/NNPP_Article.pdf , consulted on February 18th 2007.