SmartFP: Function Points as a Decision Tree

Carlo Pescio

pescio@eptacom.net

Version 1.1

08/17/2008

Abstract
Function Points estimation has been traditionally considered a *counting* problem. This traditional view, however, has several disadvantages, including significant estimation effort, lack of necessary data during early phases, and so on.
Alternatively, Function Points can be seen as a *decision* problem, guided by a decision tree. This seemingly minor change in perspective can significantly speed up estimation, while still obtaining the same result under the same information. The decision tree approach is also more advantageous during early phases, when information has higher uncertainty or is not entirely available.
The decision tree can be easily supported by a tool, averaging two mouse clicks to obtain the complexity estimate of any data or transaction function.

Keywords:
D.2.9.b Cost estimation
D.2.9.p Time estimation
D.2.8 Metrics
D.2.1.h Tools

SmartFP: Function Points as a Decision Tree

Olly is a junior buyer at LargeCompany, Inc. He's been asked to find new suppliers for business stationery, including the matching envelopes. It's his first real assignment, and he really wants to prove himself, no matter how hard he has to work. After careful research, several phone calls, and some paperwork, he has finally selected a handful of suppliers. He's now working closely with their sales department to define the exact scope of the supply, and get their best price.

BigSeller, Inc. has a strong reputation for high-quality stationery. Olly is a little intimidated by their RFQ (Request for Quotation) form. Among other things, they require him to specify the exact amount needed, for each format. It's a lot of work, because his company is using Letter paper in the USA, A4 pages in Europe, but also several smaller cards for promotional material. There is also a variety of formats for envelopes, some used on a regular basis, some just occasionally.

He has no immediate access to the necessary data, so he'll have to talk with all the involved departments, worldwide, about their present and future needs. Still, he wants to impress his boss: this careful data gathering will surely make him look like a pro.

When he’s got all the necessary data, he calls back BigSeller to get a quote. He already knows the retail price of each item: they just have to define the discount he can get.

"Hello Mary", he says. "I've finally got all those data you asked for. We'll need about 250,000 letter-sized pages annually and..."

"Oh, ok", Mary says. "Say no more: with 250,000 pages you qualify for our higher discount class: 23%. Of course, that extends to any other item too".

Sounds silly? Shouldn't salespeople at BigSeller explain their policies better, saving Olly and his colleagues a lot of unnecessary work? Well, turns out we can say the same for Function Points as well!

Function Points as a Counting Problem

Function Points (FP) [1] are a widely adopted model to estimate or measure software size in a language-independent way. FP are usually calculated through a relatively long process:

- First, we calculate the complexity of the five major components: External Inputs, External Outputs, External Inquiries, Internal Logical Files, and External Interface Files. Each of these requires two parameters, which have to be “counted”.
- The Unadjusted Function Points (UFP) can now be calculated, by associating a value to each complexity factory and adding the values together.
- An adjustment factor (VAF) is calculated to account for the unique properties of each project (for instance, stringent performance requirements).
- Function Points are obtained by multiplying UFP and VAF.

Counting the parameters for each of the five major components requires quite some effort, and in some cases, the necessary data is not readily available. If FP have to be estimated early in the process lifecycle (e.g. during a Request for Quotation, where many details haven't been worked out yet), estimators may have to *guess* several parameters. However, as stated, the FP counting procedures require precise numbers: uncertainty is not allowed.
In fact, Function Points have always been considered a "counting problem". In Albrecht's early works, the counting process was still relatively informal, but already well in place. For instance, in [2] we read statements like: "*Count each unique user data or user control input type that enters the external boundary of the application being measured, and adds or changes data in a logical internal file type*".
In the beginning, complexity classification was largely informal; in the same paper, for instance, an External Input is classified as simple if "*Few data element types are included in the external input type, and few logical internal file types are referenced by the external input type*". Highly informal definitions, however, can lead to various inconsistencies and strong variance between estimators [3].
Over time, precise counting and classification rules have been established and official counting guidelines have been published by organizations like IFPUG [4], thereby reinforcing the idea that function points must be "counted".

Function Points as a Decision Tree
Most of the effort in FP estimation goes into calculating the Unadjusted Function Points, that is, in counting the 5 parameter pairs we need to calculate the complexity of each major component. What has not been emphasized so far, however, is that we can obtain the *same* result by asking ourselves better questions.
Consider, for instance, the rules for "counting" the External Input component. An External Input (EI) is an elementary process, whereby data or control information enters inside the application boundary. To calculate the EI, you count the number of Data Element Types (DETs) and the number of File Types Referenced (FTRs). DETs are defined as user-recognizable, nonrepeated fields or attributes crossing the application boundary. FTRs are just the number of Internal Logical Files (ILF) or External Logical Files (ELF) used inside the EI transaction.
Once you obtain the precise DETs and FTRs count, you can calculate the EI complexity, using a matrix like in Fig. 1

		Data Element Types		
		1-4	5-15	> 15
File Types Referenced	< 2	Low	Low	Average
	2	Low	Average	High
	> 2	Average	High	High

Fig. 1
External Input complexity matrix.

As you can see, a lot of information is lost during the conversion from DETs + FTRs to a single complexity score. Therefore, you don't need the *exact* count for DETs and FTRs to get the *exact* complexity score.
If, for instance, you know that FTRs is 1 (because your transaction is just taking some data from an input screen and storing it in a single database table), you don't need to count DETs precisely: you only have to know if it is bigger or smaller than 15. Although this may seem like a small difference, looking at function points through this lens allows a simpler, faster, *more realistic* and *more reliable* process to determine the complexity of all five major components.
A decision tree can be defined for all the 5 major components of Function Points. Each tree is actually made of two major sub-trees, one for each starting parameter. For instance, in fig. 1, we could start with DETs or FTRs. The decision tree for External Input is given in fig. 2. For space reasons, only the "start with DETs" major subtree is shown; the other subtree is symmetrical.

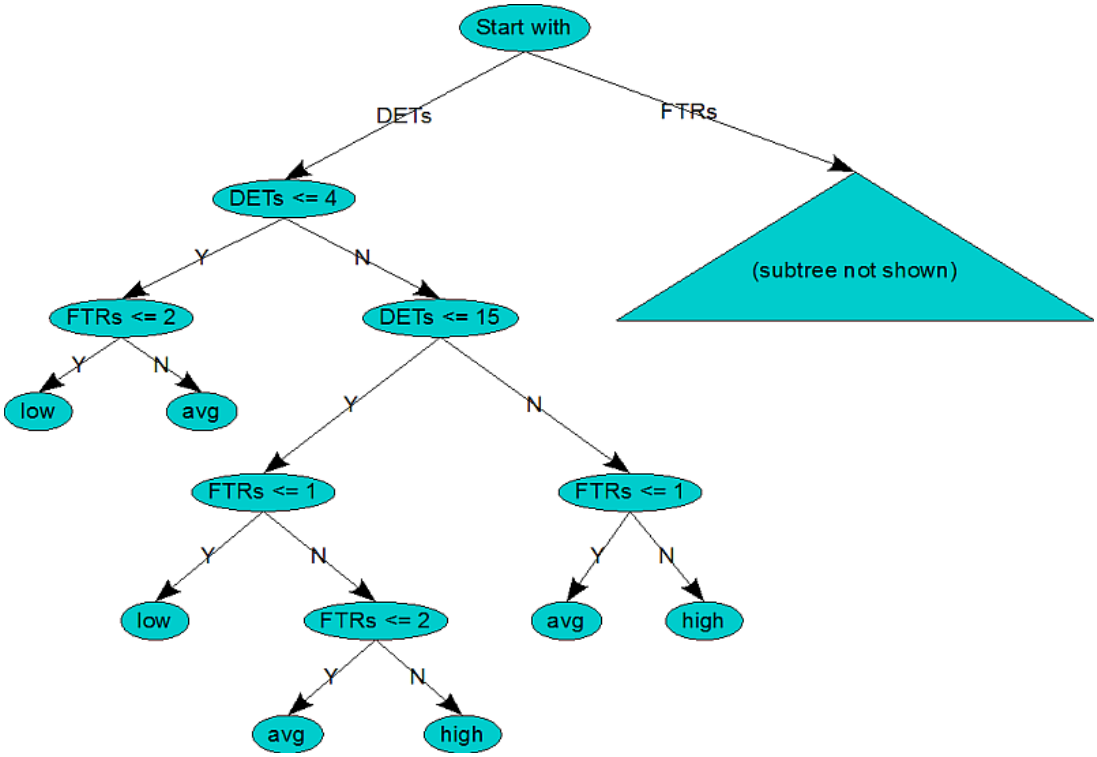

Fig. 2
Decision tree for External Input estimation.

Different questions, different process

If we look at FP as a decision tree, the very first question we have to ask for every data or transaction function is: *at this given time, which of the two parameters can be calculated (or estimated) with minimum effort, or higher reliability?*

Consider, for instance, External Inputs. In a given case, FTRs might be simpler to calculate (or estimate) than DETs, or viceversa.

If you are developing a new product, and are required to estimate FP based on a paper prototype or mock-up, you can easily estimate DETs, as input fields usually map nicely to DETs. FTRs can be more problematic to calculate, as database design is often not available in the early phases. However, consider a "complex" form with (e.g.) 18 fields. You don't really have to count the FTRs; you only have to solve a simpler problem (or make an educated guess). Will your transaction, involving those 18 fields, reference only one table, or more than one? From Fig. 1, it all boils down to that.

Now, in the real world, you'll usually need more than a simple SQL insert statement into a single table to handle an 18-fields form. It doesn't really matter if you need 2, 5, or 15 FTRs: the complexity is high, and the exact FTRs count is simply not needed.

Sometimes, of course, you might want to go the other way. If you have to add some new forms to an existing product, the database design is readily available. The exact number of DETs, however, might not be *precisely* known: the customer may (and will) ask for changes. Still, consider an input form with a hierarchical selection of data (for instance, state/province), plus a bunch of other fields. Since you maintain those data into two separate tables, state and province count as two ILF or EIF (depending on application details). You also need at least another table for the rest of the data, therefore FTRs is higher than 2. From Fig. 1, you only have to determine if DETs are bigger or smaller than 5, which is a simpler and also more *reliable* question to answer.

Don't neglect Uncertainty

Consider a different scenario in our opening story. Olly calls Mary and says: "We need about 230,000 letter-sized pages, and ..." After a quick check, Mary replies: "Ok, according our company policy you qualify for a 10% discount on all items in this order". Olly presents a memo to his boss, only to discover a (few weeks later) that Frank, a senior buyer, has got a 23% discount from the same supplier. Somewhat amazed, he asks Frank how he did it. Frank smiles: "Oh, I just took your memo, called BigSeller again, and told them I would buy 250,000 pages instead of 230,000". Olly is dismayed: all that hard work was just for nothing.

The good news is: it wasn't for nothing. The bad news is: he worked too hard, not smart enough. That's a big problem with the conventional approach to FP counting. You work hard to get an exact count of DETs, FTRs, and so on. You put those numbers into a tool, and you get a complexity factor. What the tool is not telling you, however, is that a minor change in your requirements might move you to the next level.

Consider, for instance, the previous example, where instead of 18 fields you initially have 14. You go ahead with the counting approach, spend some effort to estimate FTRs (say 2), and calculate a complexity factor (in this case, "average"). Obviously, 14 and 2 are just *tentative* numbers: requirements have never been know for their long-term stability. Yet, under the counting approach, you get a *high-certainty* answer ("average"). Unfortunately, if in the end you need 16 DTEs, or 3 FTRs, that complexity factor will be wrong (the right one from Fig. 1 being "high").

Under the decision tree approach, instead, thresholds are visible from the very beginning: indeed, thresholds become your unit of reasoning. That allows you to count faster, but also to count safely. Your initial goal is not "calculate the DETs", but "determine if DETs are less than 5, less than 16, or 16 or more". When you get an *early* 14, it is very natural to see that you're in close proximity of a threshold (a visible 16). Are you going to provide an optimistic estimate? Do you want to think more about the DETs count stability? Is it likely that more fields will be added? Removed? These questions should not be avoided if our aim is a realistic estimate.

Some uncertainty during estimation is unavoidable: the earlier the estimate, the higher the uncertainty. Faced with the task of estimating FP for a new development, however, under the "standard" counting process we have no choice but to neglect uncertainty. We have to anticipate database design, user interaction, interfaces with external systems (and so on) to come up with precise numbers. This is a fragile process, yet the fragility is hidden by the "FP as counting" approach.

Benefits

Calculating FP using a decision tree has several benefits:

- We ask better questions: we spend our time reasoning about ranges and thresholds, not about exact (but fragile) values.
- We obtain the *same* result we would obtain under the "FP as counting" approach, yet in a fraction of time. From informal measurements, I can say it's common to take $1/10^{th}$ of the time, or less, especially with appropriate tool support.
- We obtain more robust results: whenever we are near the threshold, we can investigate more about customer's needs.
- We can immediately see the impact of requirements change: if it's going to change our path down the decision tree, we need to review the estimate.
- We don't have to cheat: when the knowledge we need is not readily available, we can substitute an educated guess based on the thresholds.
- Being faster to calculate, we can also use probabilistic techniques like PERT 3 estimates [5] or my own BetterEstimate (see www.eptacom.net/betterestimate) to deal with large projects with high uncertainty.
- Being faster to calculate, we can easily keep the count up to date. Software development is a knowledge acquisition activity [6]. Still, my experience in the industrial field tells me that few managers update their estimates as they gain more knowledge. It's just too much work! However, under the decision tree approach, in most cases a new estimate is just a few clicks away.

Related works

Function Points are roughly 25 years old. Obviously, there have been several attempts to speed things up.

Early works by Capers Jones [7] on the SPQR/20 method, for instance, tried to speed things up by simplifying the factors. All functions are weighted as "average", and the adjustment factor is also reduced to account only for logical complexity and data complexity, therefore removing the 14 general system characteristics. Jones claimed that SPQR/20 would result in a similar estimate (within 1%) of traditional FP. However, other researchers [8] found that although highly correlated with development effort, SPQR/20 counts deviate more consistently from FP counts. The decision tree approach leads exactly to the same count (under the same knowledge).

Seaver [9] also adopted a simplified approach to create the Fast Function Points method. As for the SPQR/20 approach, the final result is not the same as the FP count for the same system.
Several other methods have been inspired by the “counting metaphor” of FP. For instance, in [10] the authors propose the Object Point technique for sizing object oriented software. Interestingly, the decision tree approach can be easily adopted in those cases as well, therefore speeding up counting for an entire family of techniques.

A Task-Centered tool

I’ve been teaching software estimation in industrial settings for several years. I teach several techniques, from probabilistic expert estimation to pair comparison, and also “traditional” model-based estimation using COCOMO and Function Points. Indeed, the concept of “function points as a decision tree” emerged largely as I was trying to make FP more appealing for the average company. However, I quickly found that without tool support, people were often unwilling to adopt the idea.
More recently, I developed a free tool (SmartFP, see www.eptacom.net/smartfp) to assist in calculating the Unadjusted Function Points using a decision tree. The tool is extremely simple to use, but lacks the ability to store and retrieve counts. It should be considered the equivalent of a handheld calculator, not like a full-fledged spreadsheet. However, as it gets more widely adopted, I plan to extend the tool beyond basic calculation.

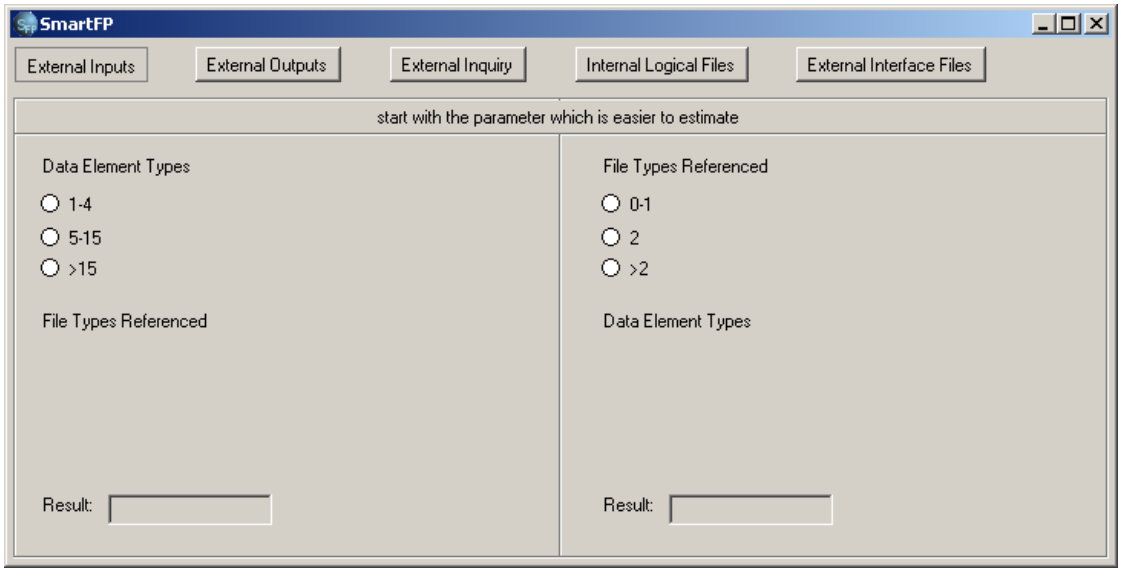

Fig. 3
SmartFP main screen

The interface (see Fig. 3) is completely task-centered. The first step is to choose the data or transaction function you want to “count” (for instance, External Inputs). Then you decide whether to start with (e.g.) DETs or FTRs. This first step does not require any clicking: you just start working on the left side or on the right side. Once you select the appropriate range, only the relevant choices for the remaining data are presented (see Fig. 4). One more click, and you get your count (Fig. 5).

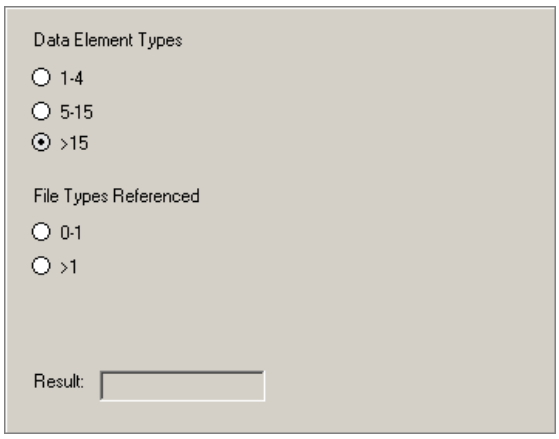

Fig. 4
Left panel after selecting DETs: only the relevant FTRs choices are presented.

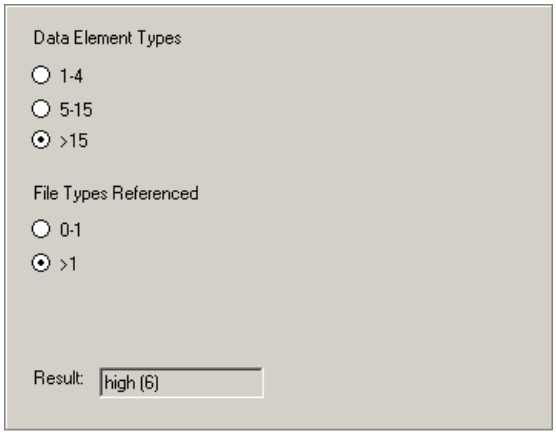

Fig. 5
Left panel after selecting FTRs: the complexity and the score are calculated with two mouse clicks.

In the end, only two clicks are required to get an answer or three clicks if you are changing function type (e.g. moving from External Inputs to External Outputs).

Note that by splitting the screen in two halves, one for each (potential) starting parameter, not only we remove one click (that would be otherwise necessary to choose the starting parameter), but we also make the decision more explicit. Ranges for the two parameters are different, and *we're not asking to count*: we're asking to choose the range that is easier to estimate. The simple, yet effective user interface has been inspired by the work of Jef Raskin, who discusses keystrokes and mouse gestures minimization while introducing the GOMS model [11].

A simple case study

The decision tree approach is expected to give the same results of the traditional counting approach, just faster. Therefore, to show SmartFP at work, I've selected a problem inspired by a case study published in a widely adopted book [1]. The decision tree is also expected to work better under partial information, and to allow better risk management. The chosen case study allows to see this kind of reasoning in action.

A company wants to build a simple Customer Relationship Management program. When a prospect is first contacted, several data must be recorded: his name, company, role, and so on. Although knowledge of the entire set of fields is irrelevant for our purposes, it's useful to know that in [1] 10 fields are listed. Fields include phone and fax number, but not (for instance) email address and mobile phone number.

The same set of fields can be modified at any time through an Update transaction, after retrieving the prospect. A prospect can also be removed: this transaction requires the company and contact name as input.

As part of the company internal procedures, a brochure is sent to the prospect after first contact. When the brochure is sent, the date must be recorded. Again, company and contact name are required as input, plus the exact date. After that, several phone calls can take place between the company and the prospect. For each call, we want to track the date and possibly some notes. Again, company and contact name are required, together with date of contact and notes. Since we have to record several calls, we need a separate database table.

When a prospect is retrieved for display (again, using company and contact name as input) all the 10 fields will be visible, plus the dates and notes above.

Any transaction may fail at some time; in that case, an external Error file is used to look up the error message. According to [1], the error file has 4 fields, one of which is the error message; no further details are given about the other 3, but one is most likely the error code to look up.

Overall, we have:

1 Internal Logical File to keep track of prospect and calls.

1 External Interface File for error lookup.

5 External Input transactions, to Create, Update, Delete the prospect, and to record when the brochure is sent and when the prospect is called.

1 External Inquiry to retrieve the prospect data.

Applying the SmartFP approach, the ILF can be quickly estimated. We have a total of 13 fields to maintain: 10 for the prospect, two dates and the notes. Hence, we're in the 1-19 range. We only have to pick the RETs number now. We have two tables, one for the prospect, one for the phone calls, hence we fall in the 1-5 range (the other choice being "> 5"). Complexity is Low. Note how thinking about ranges can immediately help reasoning: the 10 fields are hardly a definitive list. Most likely, email, mobile phone, and possibly a few others will be needed. However, we have some slack (6 fields). Even if some field ends up on a separate table, we have some slack there too (3 RETs), so the Low complexity is probably a safe bet.
The EIF is interesting, as we work with only partial knowledge. We have 4 fields; hence DETs fall well within the 1-19 range. We don't know much about the file structure, so estimating RETs is not necessarily trivial. However, the choices are restricted to 1-5 or >5. Given that we only have 4 fields, 1-5 is a really safe bet. Again, we get a Low complexity, with a large slack.
Creating a prospect requires to enter 10 fields, plus dealing with error lookup, which adds 2 fields (error code and error message), for a total of 12 (range 5-15). For that range, we have to choose between 0-1, 2, or >2 FTRs. We have 2, so we end up with an Average complexity. Note that the slack is quite small. If one more file gets involved (for instance, to validate City/State/ZIP code, which are required data), we move to >2 FTRs, which means High complexity. The slack is minimal.
For space reasons, I'll leave out a discussion of the remaining 4 External Input transactions: I encourage you to estimate them using SmartFP. You'll see how reasoning with ranges speeds things up, while at the same time helping with risk management.
The External Inquiry (retrieve) is still interesting. 15 fields are involved: 13 for the customer, plus 2 for error management. We're rather safely in the 6-19 range. The appropriate FTRs range is 2-3, giving Average complexity. There is some slack for fields (5); this is more than in the Create transaction (which was just 4).
As expected, the complexity score we obtain is the same as in [1]. The line of reasoning we adopt by following the decision tree approach, however, is remarkably different, and makes it much easier to make a safe bet under partial knowledge, or to highlight risky areas in the existing requirements.

Conclusions

I’ve seen several companies adopt Function Points and then abandon them after a short while. I’ve also witnessed several software engineers refusing to adopt function points, even though their application domain (MIS) was particularly well-suited to FP estimation. When asked why, they often cited the large estimation effort, the lack or uncertainty of data at estimation time, and a general feeling of a labor-intensive method, largely at odd with their agile, productivity-oriented working environment.

Still, there is value in FP. To reap this value, we can shift from the hard-working, counting approach to the smart-working, decision tree approach. Moreover, the decision tree approach works better under uncertain information, as is typical in early phases.
Interestingly, the decision tree approach can be applied equally well to the large number of estimation techniques inspired by function points. By reducing estimation effort, and improving on the results, this may ultimately lead to a more widespread adoption of formal estimation models.

Bibliography

[1] D. Garmus, D. Herron, *Function Point Analysis: Measurement Practices for Successful Software Projects*, Addison-Wesley, 2000.

[2] A. J. Albrecht, J. E. Gaffney, *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*, IEEE Transactions on Software Engineering, Volume 9, Issue 6, November 1983.

[3] G. C. Low, D. R. Jeffery, *Function Points in the Estimation and Evaluation of the Software Process*, IEEE Transactions on Software Engineering archive Volume 16, Issue 1, January 1990.

[4] IFPUG, *Function Point Counting Practices Manual*, Release 4.2.

[5] J.J. Moder, C.R. Phillips, E.W. Davis, *Project Management with CPM, PERT and Precedence Diagramming,* Blitz Publishing, 1995.

[6] P. G. Armour, *The Laws of Software Process: A New Model for the Production and Management of Software*, Auerbach Publications, 2004.

[7] T. C. Jones, *A short history of function points and feature points*, Software Productivity Reserch Inc., 1988.

[8] D.R. Jeffery, G.C. Low, M. Barnes, *A comparison of function point counting techniques*, IEEE Transactions on Software Engineering, Volume 19, Issue 5, May 1993.

[9] D. Seaver, *Fast Function Points*, 15th International Forum on COCOMO and Software Cost Estimation, October 24-27, 2000.

[10] G. Costagliola, F. Ferrucci, G. Tortora, G. Vitiello, *Class Point: An Approach for the Size Estimation of Object-Oriented Systems*, IEEE Transactions on Software Engineering, vol. 31, no. 1, January 2005.

[11] J. Raskin, *The Humane Interface: New Directions for Designing Interactive Systems*, Addison-Wesley, 2000.

Biography

Carlo Pescio has been in mutual friendship with software since 1978. As a consultant and mentor for several companies, he has designed software for medical devices, industrial process control, banking, finance, CAD/PLM, military, embedded/real-time and several other fields.

He's mostly interested in mixing software engineering research with out-of-the-box thinking and tackle tough software development challenges.

He received his Laurea degree, magna cum laude in computer science at University of Genoa. He is a member of IEEE Computer Society, the IEEE Technical Council on Software Engineering, and the ACM. Contact him at Via Bernardo Forte 2/3, 17100 Savona SV, Italy; pescio@eptacom.net.