Functional Sizing of Real-time & Embedded Systems
How long can you wait for CMMI Compliance?

Manage your software development in guaranteed compliance with the CMMI® NOW!

processMax includes all the necessary policies, procedures, guidelines, criteria, templates, and forms in role-based, step-by-step instructions, ready for use—everything you need for CMMI compliance. Integrated with robust document management and workflow, processMax is the intranet web-based solution for effective and efficient management of your software development projects. processMax also includes a complete measurement and reporting capability. Measurement collection and report generation are fully automated—giving managers real-time, fact-based insight into all aspects of project progress. processMax is the proven solution—as verified by more than 50 successful independent appraisals.

processMax
THE PROCESS SOLUTION

www.pragmasystems.com

pragma Systems Corporation, 1810 Samuel Morse Reston Drive, VA 20190
877.838.PMAX E-mail: info@pragmasystems.com
processMax is a registered trademark of pragma SYSTEMS Corporation.
CMMI is a registered mark of Carnegie Mellon University. Copyright © 2005 pragma SYSTEMS Corporation.
Data & Analysis Center for Software (DACS)

Tech Views:
By Ellen Walker,
DACS Analyst

The software community has been trying to accurately estimate the size of software products for a long time and has searched for a metric that is objective, consistent, accurate, a good predictor of effort, and is obtainable with minimal resources. This ideal metric would provide a common frame of reference to communicate and make decisions about software development across organizations, regardless of the domain or development environment. Physical size metrics such as source lines of code (SLOC) or number of requirements statements have been widely used but some groups within the software community believe that functional size metrics are better than physical size metrics because Functional Size Measurement (FSM) [1], in theory, has the following characteristics:

- Works for all types of software (scientific, business apps, web portals, embedded systems, etc.)
- Works for all types of projects (new development, enhancements, maintenance, etc.)
- Is language independent
- Is technology independent
- Is repeatable (consistent) – two analysts independently measuring the same software will arrive at essentially the same size assuming they are both adhering to the same FSM method.
- Produces statistically significant results
- Can be applied early in the development life cycle

But in reality, there is a pervasive perception in the software community that functional sizing is only good for business applications and not applicable to other types of systems. Those who are knowledgeable about functional sizing techniques claim that FSM works for any type of software and assert that those responsible for technical applications such as real time systems, embedded systems, control software and utility software, just need to get educated about functional sizing. Thus, we have “the great divide”, the gap between real world practitioners and the metrics experts who are perceived to be, essentially, “preaching to the choir”. Our goal in presenting this issue of STN is to narrow this gap by providing information to developers and/or acquirers involved with real time and embedded systems that will aid them in understanding the potential benefits that can be achieved from implementing FSM.

Figure 1, provided courtesy of Pam Morris[2], presents an historical view of the development and evolution of functional sizing methods from one construct to several, illustrating the fact that now there are choices of sizing methods.

Functional sizing first made its debut in the early 80s with Allan Albrecht’s Function Point Analysis (FPA) method. During the 80s and 90s the number of methods grew from one to six (Full FPs emerged). Some (3-D fps and Feature Points) faded away for lack of support or lack of robustness. In the late 90s there was a convergence of methods into two key methods that gained international support (IFPUG and COSMIC) and two methods with apparent national scope (NESMA and MK II). Appropriately, an International Standards (ISO) framework for FSM (ISO/IEC 14143 series) has been developed along with ISO standards for each of the major FSM methods.

continues on page 2
As the diagram in Figure 1 shows, developers today have choices to make regarding the FSM method that best meets their needs. Both of the major FSM methods (COSMIC and IFPUG) assert that their method works for sizing all types of software. Both methods yield a count that quantifies the functionality of the subject software in some way, regardless of the domain. But, so what? The question that each developer must ask is, “What is the value of having this particular functional size metric? What does it do for us?” We are really talking about how useful the functional size metric is for predicting and decision-making. The theory is that functional size should be a good predictor of effort, which, in turn, enables us to predict cost. Therefore, the size metric must be independent (decoupled) from, rather than a reflection of, effort. We want to be able to determine size easily and early and use it to estimate the effort and cost associated with development in our particular environment with assurance that the estimates are, in fact, reasonably accurate. That is a simple statement to make but the target remains elusive for many developers.

In the last issue of the STN (Vol 9-2 June 2006) the DACS presented a synopsis of a methodology specifically designed for use with ‘process-rich’ applications such as real-time and embedded systems. The issue also presented information about sizing with benchmark data from the project repository maintained by the International Function Point Users Group (IFPUG) and demonstrated how the repository data could be used to obtain reasonably accurate size estimates early in the project life cycle with minimal effort.

Now, in this issue, we present functional sizing as it relates particularly to non-business applications such as real-time and embedded systems, utility applications and control systems. We also provide some commentary on innovative uses of functional sizing that are independent of the chosen sizing method.

Sizing, by itself, is just a number. It does not increase productivity; it does not increase quality; it does not reduce cost [2]. However, it can provide consistent objective quantitative data, which we can then use to make better decisions that will help us increase productivity, increase quality and reduce costs.

The articles in this issue of the Software Tech News were selected to help you further understand functional sizing by:

1. Presenting a synopsis of a methodology specifically designed for use with ‘process-rich’ applications such as real-time and embedded systems
2. Comparing IFPUG and COSMIC sizing methods to each other to demonstrate why an organization might choose one method over the other depending on how the size metric is to be used
3. Illustrating the importance of statistical analysis to determine the circumstances in which the size metric can be considered a good predictor for effort estimates
4. Presenting size metric usage scenarios that reflect innovative and effective acquisition strategies based on the ‘cost per delivered functional unit’ paradigm.

In the first article, author, Charles Symons, describes the

*continues on page 3*
methodology for functional sizing called COSMIC-FFP (Full Function Point). COSMIC, the Common Software Measurement International Consortium, is the group responsible for the creation and maintenance of COSMIC-FFP, established in 1998. The consortium designed this sizing approach specifically for real time and embedded systems software that is essentially “process-rich”. This method has a strong user base in Europe and the data from many projects sized with COSMIC-FFP resides in the ISBSG repository. “Cisu” denotes a COSMIC functional size unit. Symons exemplifies how the sizing method works by applying it to the security alarm system in his home.

In the second article, Pam Morris makes a technical comparison of the IFPUG and COSMIC methods and illustrates the impact that the choice of method can have in the implementation of outsourcing agreements or other types of contracting based on cost per delivered functional unit. This article should help readers to understand why there is currently not a single “one size fits all” functional sizing method that is equally valuable (useable) for all types of software development.

Article three, by Alain Abran, illustrates the use of functional size measures (using COSMIC-FFP) in building estimation models for maintenance projects. The research is based on 19 maintenance projects (implementing small functional enhancements) on a single real-time embedded software application in the Canadian Defense industry. His article illustrates that there is no easy road to deriving value from using functional sizing. An organization must be prepared to do thorough analyses of their data in order to find the best models for their particular environment, whether they are involved in maintenance or developing new software.

Article four, by Terry Wright, titled “With or without a navigator? --- it’s your call!” describes the use of functional sizing as an acquisition strategy. The initiative, called SouthernSCOPE, undertaken by the State of Victoria in Australia, was unique in two ways: First, it required bidders to submit proposals based on a ‘cost per delivered functional unit’; and second, the acquirer engaged a ‘Scope Surveyor’ throughout the development life cycle.

Although the initiative proved very successful, sustaining the best practices of that initiative proved to be very difficult. Terry tells the story about a functional role that addresses sizing throughout the project and seems to be important for success. He hints, however, that the cultural aspects of project management are a formidable barrier to the adoption of this new role by acquirers and developers alike.

The last article, written by Juan Carlos Molina, describes how his organization has used functional sizing as the basis for charging customers for code generation based on a Model-driven Architecture approach (MDA). The MDA tool automatically sizes the application (using the IFPUG counting method) from the software model before code is generated. Developers or Acquirers can then calculate the cost of code generation before any resources are committed to it. Developers focus on the model, not the code. Subsequently, this approach can have a significant impact on planning both the effort and schedule needed to develop software and therefore the cost. It seems to be a WIN-WIN situation for all --- the developer, the acquirer and the service provider --- because everyone knows up front how much the code will cost, and because the method used for determining size is consistent and objective. The developer can focus on estimating the effort required to develop the model.

The information about FSM is vast. It is of global significance and it touches many software domains. This issue only scratches the surface of FSM knowledge, but it does demonstrate that FSM is no longer relegated to the domain of business applications only. These articles all suggest that FSM is not something you do occasionally. Nor is it very useful in a short-term scenario. Its value increases as it is applied to more and more projects, and as the organization builds its project repository. “Which method is best?” is not necessarily the right question. Instead, you need to ask, “Which method is right for us?” Focus on what you want to do with the metric and let that drive your decision-making. Think usage! Usage! Usage!

References


About the Author

Ellen Walker, a DACS Analyst, is currently developing a series of publications on software “best practices” as part of the DACS Gold Practice Initiative. She has spent the past 20 years as a software developer in various roles spanning the entire software life cycle including project management of multiple business process re-engineering efforts within the DoD community. She is also experienced with assessment initiatives such as the Capability Maturity Model for Software (CMM-SW) and the quality management practices of the New York State Quality Award program. Ellen has an MS in Management Science (State University of New York (SUNY) at Binghamton), and bachelor degrees in both Computer Science (SUNY – Utica/Rome) and Mathematics (LeMoyne College).

Author Contact Information

Email: ellen.walker@itt.com
ewalker@dacs.dtic.mil
Phone: 315-334-4936
The DACS Gold Practice Initiative:

- Promotes effective selection/use of software acquisition & development practices
- Defines essential activities/benefits of each practice
- Considers the environment in which each practice is used
- Addresses the timeliness of practice benefits
- Recognizes interrelationships between practices that influence success or failure
- Contains quantitative and qualitative information
- Includes discussion forums, case studies and survey opportunities
- A continually evolving resource for the DoD, Government, Industry and Academia
- Free to use/free to join

Current Gold Practices:

- Acquisition Process Improvement
- Architecture-First Approach
- Assess Reuse Risks and Costs
- Binary Quality Gates at the Inch-Pebble Level
- Commercial Specifications and Standards/Open Systems
- Ensure Interoperability
- Formal Inspections
- Formal Risk Management
- Goal-Question-Metric Approach
- Integrated Product and Process Development
- Model-Based Testing
- Plan for Technology Insertion
- Requirements Management
- RequirementsTrade-Off/Negotiations
- Statistical Process Control
- Track Earned Value

Learn More About the DACS Gold Practice Initiative:
http://www.goldpractices.com

Become a Registered Member!
http://www.goldpractices.com/registration/register_step1.php

Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

http://iac.dtic.mil/dacs
Sizing and Estimating for Real-time Software – the COSMIC–FFP Method

By Charles Symons, Common Software Measurement International Consortium

If a reliable method existed that allowed managers of real-time software development or enhancement projects to....

A. estimate a software size from its requirements, early in the life of a project
B. enter that size together with other parameters into formulae to estimate project effort and duration (where the formulae had been calibrated on recently-completed projects in the same domain)
C. track the software size as requirements evolve (e.g. for contract control purposes) and compute project performance parameters (e.g. productivity, speed of delivery and defect density) when the project is delivered

... then you would expect the method to be widely used.

The world of business application software has had such a process for more than 25 years, since the 1970s when Allen Albrect created 'Function Point Analysis' (FPA) [1].

This article describes a method called COSMIC–FFP (Common Software Measurement International Consortium – Full Function Points) that can be used for sizing 'real-time' software requirements, as well as those of business software, to provide the key ingredient for project estimating and project performance measurement. Under 'real-time' software we include process control, embedded, telecoms, infrastructure (i.e. operating system) software and such-like.

1. The Background to COSMIC and Its Aims

Figure 1 of the Tech Views article (see page 2 of this issue) presents the history and evolution of Functional Size Measurement (FSM). It illustrates how a number of variants of Albrect's FPA emerged and evolved over time and that, now, most of them have either ceased, or converged into two major methodologies, International Function Point Users Group (IFPUG)-sponsored FPA and COSMIC-sponsored FFP. IFPUG FPA and its variants are collectively known as '1st Generation' (1G) FSM methods because they were designed for and embraced the business software paradigms and terminology of the 80s and 90s. As new software paradigms and languages evolved and even as the business world relies more heavily on real-time software, the scope of applicability of these 1G FSM methods is, perhaps, more limited. They remain strong and useful in the pure business application world, but are losing relevance for hybrid software environments as new functional sizing methods, such as COSMIC–FFP, emerge.

Lacking a functional sizing method that seemed appropriate, many large real-time environments have relied on estimating Source Lines of Code (SLOC) as the size measure and have built up large databases containing SLOC size measures for their projects. They perform the effort estimating step, either by formulae or by searching the database for analogues of the new project. However, although SLOC can be counted accurately and automatically upon project completion, estimating size in units of SLOC at the requirements stage is, obviously, not ideal.

With these aforementioned issues in mind, a group of software metrics experts gathered in late 1998 to form 'COSMIC', the Common Software Measurement International Consortium. Most of the founding members (including this author) were members of an ISO Working Group (ISO/IEC JTC1 SC7 WG12) on Functional Size Measurement that had laboured for several years to try to agree on some basic principles for FSM [3], but progress was slow. It became clear to some WG12 members that if ever we were to seriously improve on 1G FSM methods, this would have to be undertaken as a private initiative, outside the formal ISO processes.

An example of the sort of difficulty in this subject that WG12 faced is that there is still no definition of 'functional'.

The ISO/IEC 14143 standard, published in 1998 [3], defines "Functional size" as:

“A size of the software derived by quantifying the Functional User Requirements", where the latter are defined as "a sub-set of the users’ requirements …. (they) represent the user practices and procedures that the software must perform to fulfill the users’ needs. They exclude Quality Requirements and any Technical Requirements".

This definition, with hindsight, has a strong flavour of the business application software world. For our purposes, think of "functional size" as a size measure concerned purely with requirements for information processing, thus excluding any requirements concerned with quality, technical or implementation factors. The latter must be taken into account when estimating project effort, but do not contribute to functional size.

When COSMIC was established in late 1998, it set itself a clear initial aim, namely ‘to develop, test, bring to market and seek acceptance of new software sizing methods to support estimating and performance measurement’. The methods should:

- Measure a ‘functional size’ of software
- Be applicable to business application and real-time software and hybrids of these
- Be applicable for software in any layer of a multi-layer architecture
- Be applicable at any time in a software project life-cycle
Sizing and Estimating for Real–time

Continued from page 5.

from early requirements gathering through to post-
implementation, and for maintenance and enhancement
activities

Today, COSMIC is still a voluntary, ‘open’ organization, but
now with representatives from 15 countries [4] on its International
Advisory Council. The functional sizing method that has been
developed is called COSMIC-FFP. Its stability is demonstrated by
the fact that an International Standard for COSMIC-FFP [5] was
published in 2003. The method is rapidly gaining acceptance in
the market place (see side panel for examples of two current users
of the method in the real-time domain).

2. Overview of the COSMIC-FFP method

Space does not allow a full description of the method in this
article, so here we will give only a summary of the key concepts
with some elaboration of particularly important concepts for
those working in the real-time software domain. For the full
definition of the method, including definitions of all the main
concepts, see the ‘Measurement Manual’. This is freely available
download from the COSMIC web site in several languages [6].

The COSMIC-FFP Measurement Process consists of three
main phases:

1) Setting the Measurement Strategy
2) Mapping the ‘Functional User Requirements’ (or ‘FUR’) of
the software to be measured to the COSMIC-FFP concepts
3) Measuring the resulting COSMIC-FFP model of the FUR

Phase 1: Setting the Measurement Strategy

This phase consists of first establishing the Purpose and Scope
of the measurement. The Purpose is important because it affects
the accuracy that will be required of the measurement, the tim-
ing of the measurement in the software life-cycle, the software
artefacts (statements of requirements, physical screens, etc.) that
must be used, and so on. The measurement Scope determines what
functionality is included in any one measurement. It might be, for
example, that if the Purpose is to support estimating for a distrib-
uted software system and if the various components are planned to
be implemented on different types of technical platforms, then the
FUR must be separated into a number of separate measurement
Scopes, one for each platform.

The Purpose and Scope together determine the ”Measurement
Viewpoint“ that will be adopted. This defines the form of abstrac-
tion used for the FUR and which will be used for the measurement1.
In this short article, it is best described by examples.

• In the domain of simple real-time (for example, embedded)
software, the FUR are normally described in terms of the

functionality that must be provided to the ‘direct users’
of the software. These ‘direct users’ are typically hardware
devices, e.g. sensors, buttons, display panels, valves etc.,
and also other pieces of software that interact directly
with the software to be measured. They are called ‘direct’
users because there is no software between those users and
the software to be measured. We call this abstraction the
‘Direct User’ measurement viewpoint2.

• In the domain of business application software, the FUR
are normally described in terms of the functionality that
must be provided to the human users of the software. All
hardware (screens, keyboards, printers, etc) and software
(operating system layers) between the human user and
the application software to be measured is invisible in this
abstraction and is ignored. We call this the ‘End User
measurement viewpoint.

The COSMIC-FFP method can be applied to measure a
functional size using either of these two standard measurement
viewpoints, and the resulting sizes will obviously differ, since dif-
ferent functionality is revealed in each abstraction.

If, for example, our purpose were to support development
effort estimation for some embedded avionics software, then we
would almost certainly want to measure using the Direct User
measurement viewpoint.

Alternatively, the total aircraft control system could be
measured from the End User measurement viewpoint, where the
end users are the aircrew. Such a measurement could be relevant
in the context of some aircraft simulator training software, but it
would be useless for the developer of any avionics software to be
embedded in the real aircraft.

Introducing the concept of different standard measurement
viewpoints has been a vital break-through for FSM. We now
understand that 1G FSM methods were designed to measure size
as seen by human End Users. It explains why 1G FSM methods
cannot be easily applied to measure real-time software. For most
practical purposes, the functionality of real-time software needs
to be measured from the viewpoint of all of its ‘direct’ users.

COSMIC is also introducing into the method3 a factor that,
if ignored, can result in highly misleading size measurements.
This is the need to consider possibly varying ‘levels of granularity’
(or of decomposition) of the FUR to be measured. For example,
in the early stage of a software project the development effort will
probably need to be estimated when the requirements have only
been agreed at a high-level of granularity, i.e. all the details have
not yet been worked out. Guidance is therefore needed on how
to scale a size measured at this high-level to the (bigger) size at
the lower level of granularity of the full requirements.

Phase 2: Mapping to the COSMIC-FFP Concept

continues on page 7
Sizing and Estimating for Real–time
Continued from page 6.

**tual Models**

The COSMIC-FFP method relies on a number of carefully defined and inter-related concepts, which are summarized in Table 1. The Measureur's task in this phase is, in effect, to extract a model of the software in terms of these COSMIC-FFP concepts from whatever software artefacts are available.

Figure 1 presents the generic software model for COSMIC-FFP, in which these concepts are implemented.

**Phase 3: The Measurement Phase**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers</td>
<td>Software is structured in 'layers' (COSMIC-FFP provides rules for distinguishing layers for cases where the software to be measured does not exist in an established layered architecture; the Scope of any one measurement must not encompass software in more than one layer).</td>
</tr>
<tr>
<td>Events</td>
<td>Software is triggered to do something by 'events' in the world of its 'users' (which will be hardware devices and/or other items of software, in the direct user measurement viewpoint).</td>
</tr>
<tr>
<td>Functional Process</td>
<td>Each unique event type triggers a unique ‘functional process’ in the software that is complete when it has performed whatever is required to respond to the triggering event.</td>
</tr>
<tr>
<td>Sub-processes</td>
<td>Any functional process comprises ‘sub-processes’ that either move or manipulate data (see Fig. 1).</td>
</tr>
<tr>
<td>Data Movement</td>
<td>A 'data movement' is the movement of a single 'data group' (of one or more 'data attributes') about a single 'object of interest' (i.e. 'of interest' in the world of the users). There are four types of data movements: Entry, Exit, Read, and Write.</td>
</tr>
<tr>
<td>'Entry' and 'Exit'</td>
<td>‘Entry’ and ‘Exit’ data movements move data across a 'boundary', from or to a user respectively.</td>
</tr>
<tr>
<td>'Read' and 'Write'</td>
<td>'Read' and ‘Write’ data movements move a single data group from or to ‘persistent storage’ respectively (where ‘persistent’ means that the data survives the functional process that stores it).</td>
</tr>
</tbody>
</table>

In this phase we simply count the number of data movements identified in each functional process of the software to be measured, as the measure of its functional size. Each Entry, Exit, Read and Write is counted as one ‘COSMIC functional size unit’ or ‘Cfsu’.

Any one functional process must have a minimum size of 2 Cfsu, since to be meaningful it must have a ‘triggering Entry’ plus a successful outcome, either one Exit or one Write. However, there is no upper limit to the size of any one functional process (unlike the upper limit to the size of an IFPUG ‘elementary process’ which may not exceed 7 function points). In some individual military avionics software, we have measured single functional processes with over 100 data movements (i.e. of size > 100 Cfsu).

The method can also be used to measure the size of software changes for maintenance or enhancement activities. The size of a changed functional process is defined as the sum of the number of added, modified and deleted data movements.

**What about data manipulation sub-processes?**

All software includes some data manipulation sub-processes; why not identify and count them? The answer is simply that no one agrees how to measure the functional size of an algorithm, so we cannot measure data manipulations on any scale that would be widely acceptable. We therefore make an approximation and assume that the functional size of an item of software is proportional to the number of its data movements, and assume each data movement includes certain associated data manipulation. For example, an Entry is defined to include all manipulations associated with input data validation that do not involve another data movement.

The COSMIC-FFP method is, therefore, considered applicable only for business applications and real-time software that is characterized as ‘movement-rich’. It cannot accommodate ‘algorithm-rich’ software involving heavy mathematical manipulation.

![Figure 1: The COSMIC-FFP Generic Software Model](image-url)
3. A Simple Example

Applying the COSMIC-FFP method to size real-time software in the Direct User measurement viewpoint is normally remarkably simple. Although today the measurement process is manual, there is reasonable hope for automation of the sizing if the FUR become available in a Computer Aided Software Engineering (CASE) tool [7].

Some simple, fully documented, ‘real’ case studies of real-time software are available from the COSMIC web site for free download [8]. For the purposes of this article, I will illustrate the measurement process with a simple example from my house, which has a basic intruder alarm system. Until I started this article, I had never considered how it might work and I have never developed any embedded software in my life. But having looked at it, I believe that a diagram showing the embedded software and its (direct) users must look something like the diagram presented in Figure 2 and that I can figure out how to measure the functionality.

The system has a conventional keypad and red/green LED’s for the human interface. The system is also connected to a device that senses whether the main front door is open or not, several internal movement detectors, and an internal and an external alarm. There is a battery to take over if the main power supply fails, so there must be a main voltage detector. Finally, our PIN code is stored in the system and can be changed, so there must be some persistent storage hardware, which, no doubt, also contains various system parameters.

There must be an internal clock mechanism, since certain functions must be completed within pre-set times. For example, having set the ‘exit code’ before leaving the house, the front door must be closed within a given time or alarms sound. I assume the internal clock is started in the software each time it is needed, so this functionality to keep track of elapsed times (by counting clock-cycles?) is a form of data manipulation, which we can ignore.

I have no idea whether the software periodically polls its input devices or whether the devices send signals to the software whenever they detect something. But this difference has no ef-

![Figure 2: The Simple Intruder Alarm System](image)

fect on how we map the software functionality to the COSMIC-FFP Generic Software Model (see Figure 1).

The following appear to me to be the unique types of events that the alarm can detect, each of which triggers a unique functional process. There are probably more event types, known only to the maintenance engineer.

- A new, or changed, PIN code is entered via the keypad (2 events)
- The ‘exit code’ is entered via the keypad thereby activating the system (before leaving and closing the front door within a pre-set time)
- The front door sensor detects that it is open whilst the alarm system is activated
- The system is activated, or de-activated, whilst in the house (2 events), e.g. when retiring upstairs to bed
- A movement detector signals a movement whilst the alarm system is activated
- The mains voltage detector signals main electrical failure, or its restoration (2 events)

Now let’s examine, as an example, the functional process that is triggered when I return to the house, with the alarm activated. When I unlock and open the front door, the internal alarm starts ‘beeping’, and I must then enter our PIN code within a pre-set time to de-activate the system and stop the internal alarm ringing. If I do not enter the PIN code before the pre-set time, or I enter the wrong code more than three times, the external alarm also starts ringing. This functional process must have the data movements identified in Table 2.

continues on page 9
Note that the analysis of this functional process is very simple. I do not need to understand the detailed logic of, or the precise sequence of steps of, the process. The size measured accounts for all the data movements arising from all possible paths through the process; the size does not depend on the input values to the process which may determine any particular actual path.

The example illustrates that any software engineer with a clear understanding of the COSMIC-FFP concepts should be able to apply the measurement process to any real-time software requirements, or to the functionality of an existing software system with which they are familiar.

But this example may now raise another objection to the method. Applying the COSMIC-FFP method as described above requires a quite detailed statement of requirements. In a real project, you may well need to estimate a size well before you have the requirements in such detail, i.e. at a high level of granularity, as mentioned above. The Measurement Manual [6] therefore gives guidance on how to construct your own local, approximation-variant of the method that can be used when the requirements are known only at some ‘higher level’, that is, in less detail. See [13] for an example of an approach to early sizing.

4. COSMIC-FFP for Estimating Project Effort and Duration

As of today, if you have mastered the functional sizing step (A), there are three possible approaches to the estimating step (B).

**Approach 1:** The first approach is to re-calibrate your current estimating method by substituting sizes measured in Cfsu for the sizes currently used (most likely estimates of SLOC?). This will require measuring the functional sizes of several existing software products for which the corresponding effort and duration are known. There is really no way of avoiding this step, but measuring some existing software is essential anyway to gain experience and build confidence in the sizing method.

Several academic papers, e.g. [9] and [10], have been published showing results for this approach, including for maintenance and enhancement projects, using the COCOMO model and fuzzy logic. See [11] for an example of a tool to support this process.

**Approach 2:** If you have no established estimating method, then you can measure several existing software products (using COSMIC-FFP) from a domain where there is a high degree of project commonality (e.g. same technologies used, similar time/effort constraints, similar project team skill levels, similar risk, etc.) and then carry out your own statistical analyses so as to develop formulae to predict effort and duration from software size.

**Approach 3:** Finally, it is possible to measure the functional
size of a new requirement set and then use ISBSG benchmark data to produce a first, ballpark estimate of effort. As of early 2006, there are data on around 95 projects measured using COSMIC-FFP in the ISBSG database and a report is available [12] from ISBSG with an analysis of the data. Of course, not many of these projects are yet from the real-time defence software community, but the number of projects submitted to the ISBSG database is growing and a special effort is being made in 2006 to gather more such data. This approach will become much more valuable in time, especially to provide sanity checks on early estimates.

5. COSMIC-FFP Today: Conclusions and Lessons Learned

Over the five years since the first detailed account of the COSMIC-FFP method was published, a lot of practical experience has been gained and a lot has been learned.

The most gratifying aspect of this experience is that the underlying software models have been found to be robust when applied to an enormous variety of types of software. I believe the greatest lessons we have learned have been to recognise the importance of defining the measurement viewpoint, or abstraction, and the level of granularity to be used for a measurement. We have achieved our original goal of being able to measure credible functional sizes of real-time software for the first time, whilst continuing to be able to measure business application software.

The COSMIC-FFP method is a simple, proven, practical method of functional sizing for real-time software and is therefore a key ingredient for effort estimating methods and performance measurement. It is characterized as follows:

- It is stable and standardized. As we learn how to improve our explanation of the method, we continue to refine the Measurement Manual, but the underlying concepts are solid.
- The method is 100% 'open'. The Measurement Manual is available in four languages and translations are scheduled or underway in another three.
- It has been very widely applied in both the business and real-time software domains. In the latter, I am personally aware of its successful application to size software in the fields of avionics, telecommunications, process control, paper copiers, digital cameras, vehicle engine management systems, and scientific instruments. Geographically, there is now substantial experience in Asia, Australasia, Europe and North America.
- Various informal tests have been carried out on the repeatability of measurements by different measurers working from the same specification. The results are both encouraging and predictable. If the specifications are clear, the measured sizes are highly repeatable. If there is ambiguity in the specifications, the results will not be repeatable.

One of our most intriguing and widespread findings is, namely, that applying the COSMIC-FFP sizing method is an excellent method for quality control of software requirements. If requirements cannot be sized, the software cannot be built without the developers having to make some assumptions. Furthermore, if you have a mapping of the functional requirements to the COSMIC-FFP conceptual models you have, in the catalogue of functional processes, an excellent foundation for requirements traceability and test case design.

The necessary infrastructure to support world-wide use of the method is now becoming well-established. This includes benchmark data, organizations that can provide training and consulting services, automated tools to support data collection and estimating (see [4] for more details), and certification examinations [8].

There is now a steady stream of papers at software metrics conferences demonstrating the growing interest by academics in research using COSMIC-FFP based measurements.

Looking Ahead

The future holds some interesting possibilities. Why not use the COSMIC-FFP method to size information processing functionality before the hardware/software allocation is decided? You can see no reason why not, but it needs testing on major systems. (It’s trivial to size the functionality of a pocket calculator without knowing if the functions are embodied in hardware or software.) Why not size software functional requirements automatically if they are held in a CASE tool? Another topic already being explored [7].

If your organisation decides to adopt the COSMIC-FFP sizing method, please submit as much of your data as possible to the ISBSG database. It’s a real-chicken-and-egg challenge, which you can help to solve. The more benchmark data that is made available, the more valuable the method will become, the more data that will be generated, and so on. COSMIC has laid the foundations. Now the method’s users can add the greatest value.

And finally, remember it is an open method. Our documentation and support can always be improved. If you have suggestions or contributions for improvement, e-mail me at mpc-chair@cosmicon.com.

Early Adopters of COSMIC-FFP for Real-time Software

continues on page 11
Nokia first used the COSMIC-FFP method in the worldwide field trials in 2000. Since then its use has gradually been extended among different R&D units within Nokia Networks for managing the development of real-time and embedded software in various network elements. Some R&D units have implemented the method as standard practice for their effort estimation. It is also used as one vehicle for software development productivity measurement and monitoring. The results for improving estimation accuracy by means of COSMIC-FFP based size measurement are very encouraging, especially for the larger software items: the larger an estimated item, the better the result that is achieved.

The software department of the Perkin-Elmer R&D Centre for Process Analytics and Control Technology in the UK has used COSMIC-FFP for around 2 years to provide early estimates of the size of new software projects.

References

[8] See www.gelog.etsmtl.ca/cosmic-ffp

About the Author

Charles Symons is an independent consultant, leading COSMIC (the Common Software Measurement International Consortium) jointly with Professor Alain Abran of the École de Technologie Supérieure, Université du Québec, Montréal, Canada.

He graduated in physics at Birmingham University and began his career with the UK Atomic Energy Authority. From there he moved to CERN (the European Centre for Nuclear Research) in Geneva, Switzerland, to Philips Electronics in the UK and the Netherlands, and to Xerox. After this 25-year 'apprenticeship' in most aspects of computing, he became a management consultant with Nolan Norton & Co and later KPMG in London, advising on IT strategy and on improving the performance of the IT function. He is now semi-retired, living in the UK and the French Alps, relying on the internet to stay in touch with the COSMIC network.

Author Contact Information

Email: [cr.symons@btinternet.com]
1. INTRODUCTION

Outsourcing Information Technology's (IT) software development and maintenance activities has become increasingly popular as a means of enabling an organisation to more effectively focus on their core business activities.

The high cost and risk associated with these IT outsourcing contracts means that they need to be carefully monitored and managed. Both the client and the supplier need to establish a means by which the client’s software assets can be quantified and the supplier’s performance can be evaluated and compared to agreed targets. The most common mechanism for providing these performance measurements is to measure the supplier’s productivity rates in units of software product delivered per unit effort or units of software product delivered per unit cost.

The units of software product delivered are usually measured using a Functional Size Measurement (FSM) method called IFPUG Function Point Analysis (FPA). FPA was developed by Alan Albrecht in the late 1970s and has since been refined by the International Function Point Users Group (IFPUG). Analysis of industry benchmarking repositories shows that the use of FPA has been largely concentrated within the business application domains i.e. software which delivers functionality to the human business user. These applications are typically commercial and management information systems (MIS) software. FPA has been found to very effective in measuring the functionality delivered by these types of applications, particularly when determining the productivity of software development and using the size as input into estimates for project resources and schedules. However, in recent years the need to quantify software has extended way beyond just measuring business applications for these purposes. Many developers are working on real-time embedded and process control software where the users are software in electronic components rather than people. Other developers are building infrastructure software, which enable the business applications software to operate in new technology environments and therefore have other software components as their users. With the advent of outsourcing, the client and supplier need to monitor all software worked on by the development teams, not only the applications that deliver software directly to the business users, but also include software that has other software components or equipment as its primary users. However developers recognise that the functionality delivered by these ‘other types’ of applications does not behave in the same way as functionality delivered primarily to human users.

Many functional size measurement specialists have found that FPA is less effective when measuring software, which delivers functionality to ‘users’ other than the business user compared to when it is applied in its traditional domain for which it was designed. This creates a problem in outsourcing contracts where all the supported software needs to be included in the performance measures and the charging model. This paper explores the reasons for these observed limitations of FPA and shows how they can be overcome using COSMIC-FFP.

Results from industry trials with COSMIC-FFP showed that the COSMIC-FFP measures have the potential to be used successfully for productivity monitoring and estimating, where FPA measures have been tried and were found to be lacking.

2. TYPES OF SOFTWARE APPLICATIONS

The functional domains of software applications, usually included within the scope of the outsourcing contract, can be categorised based on the type of end-user and the services provided by the software to these end users. These categories are illustrated in Figure 1; they include, but are not restricted to, the types identified in Table 1.

Outsourcing contracts may include the support and maintenance of applications in any or all of these categories. For the purposes of this paper, we will describe the software that is not of the type ‘business applications software’ as ‘non-business applications’ software. The non-business applications include real time embedded or control software and infrastructure software i.e. utility software, user’s tools, developer’s tools and systems software.

![Figure 1: Categories of Software](image_url)

continues on page 13
A Method for Sizing  
Continued from page 12.

3. FPA AND COSMIC-FFP CONCEPTS

This section compares the differences between the basic concepts of FPA and COSMIC-FFP. COSMIC-FFP was developed as a collaborative effort of the international functional size measurement community who distilled the ideas and lessons learnt from the preceding FSM Methods into a single FSM method that optimised the strengths of the previous generation methods and addressed their perceived limitations. Although initially developed to take into account functional characteristics specific to real-time software, its concepts have been found to be equally applicable to other types of software particularly where the primary users are not human.

3.1. FPA Concepts

IFPUG FPA was designed, refined and mostly used for Business applications software, which usually constitutes about 70% - 80% of a commercial organisation’s software portfolio. This section describes the basic concepts of IFPUG FPA, which have contributed to it being able to be used to effectively measure Business Applications software over the past 25 years. Full details of the rules for applying FPA can be found in the IFPUG Counting Practices Manual 4.2.

The FPA method measures functionality by quantifying the software’s processes (Inputs, Outputs, Enquires) and data files (Internal and External). It is based on the following fundamental principles:

1. Functional size is the measure of the functionality delivered to the end business user. Only processes that send or receive data to, or from, the external user boundary are included in the measurement of functionality delivered to the user.
2. A process is required to have a predominant role of either entering or extracting data. This predominant role determines the process type (Input, Output or Enquiry).
3. The functional size of a process is directly related to the:
   - Amount of data which crosses the external user boundary of the software during the execution of the process
   - The number of data groups, accessed by the software during the execution of the process

**Table 1: Profile of Software Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Users</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Applications Software</td>
<td>Typically business or commercial (MIS) software which delivers functionality that supports the organization’s core business</td>
<td>Primarily human business users</td>
<td>Payroll applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other business applications</td>
<td>Accounts Receivable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Customer Resource Management systems</td>
</tr>
<tr>
<td>Embedded or Process Control</td>
<td>Delivers functionality, which supports the organization’s core business</td>
<td>Other software components embedded in equipment</td>
<td>Equipment monitoring systems</td>
</tr>
<tr>
<td>Software</td>
<td>Typically operates under strict timing conditions and is often referred to as Real-time software</td>
<td></td>
<td>Telephone switching systems</td>
</tr>
<tr>
<td>Utility Software</td>
<td>Delivers software that provides the infrastructure to support the Business Applications Software</td>
<td>Primarily business applications software</td>
<td>Backup utilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Administrative users</td>
<td>Archiving utilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Installation and conversion software</td>
</tr>
<tr>
<td>Users’ Tools Software</td>
<td>Tools used by administrative users to create the functionality delivered by the Business Applications Software</td>
<td>Primarily business applications software</td>
<td>Report generators, spreadsheets, and word processors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Administrative human users</td>
<td></td>
</tr>
<tr>
<td>Developers’ Tools Software</td>
<td>Tools used by developers to create the functionality delivered by the Business Applications Software</td>
<td>Primarily other applications</td>
<td>Code generators</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IT developers as administrative users</td>
<td>Testing software</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New product generators etc.</td>
</tr>
<tr>
<td>Systems Software</td>
<td>Enable all other types of software to operate and deliver functionality.</td>
<td>Primarily other applications</td>
<td>Operating systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limited interface to Human IT operational staff</td>
<td>Printer drivers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Protocol converters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Presentation software</td>
</tr>
</tbody>
</table>

continues on page 14
4. An extremely complex process of a particular type can only be measured to have, at the most, double the functionality of the simplest process of that type.

5. The size of an individual process is limited to three predefined potential values which have been pre-set at non-linear increments (e.g., 3, 4, 6).

6. Functionality delivered by stored data is a significant contributor to the overall functional size of the software (usually >25%).

7. Functionality changed during an enhancement project is recorded as being the measure for the whole function (process or data group) irrespective of the proportion of the process being changed.

3.2. COSMIC-FFP Concepts

The COSMIC-FFP method measures functionality of the software by quantifying the software’s sub-processes (data movements) within each process. It is based on the following fundamental principles:

1. It measures functional size from the functional perspective instead of the external user view. I.e. measures the functionality required to be delivered by a process to the user of the process not just the functionality experienced directly by the user. Sub-processes that access data in the data groups (reads and writes) are included in the measurement of functionality in addition to the sub-processes required to receive data (entry) and extract data (exit).

2. A process is not required to have a predominant role per se. In order to measure size it is only necessary to identify all the Entry, Exit, Read and Write sub-process data movements.

3. The functional size of a process is determined by measuring the number of unique sub-processes (data movements). The predominant role of a process to either extract or accept data does not influence its resultant size.

4. An extremely complex process of a particular type can be sized accordingly by awarding proportionally more ‘points’. In theory, there is no limit to the number of points awarded for one specific process.

5. The number of ‘points’ awarded to a process (i.e., its functional size) is directly related to the number of unique data movements executed and therefore, the process size can be measured in linear increments of 1 fp unit, with a potential to reach an infinite size. The minimum size is two units with possible sizes in integers from two to infinity units.

6. Functionality delivered by stored data does not contribute to the overall functional size of the software. (I.e. stored data only contributes via the Read and Write sub-process-es); it is not counted as a separate functional type.

7. Functionality changed by an enhancement project is recorded at the level of the sub-processes. Only a part of the process is credited for being impacted by the change (identified by the sub-processes added, changed or deleted).

The COSMIC-FFP FSM Method has taken into consideration current industry practices in the specification of software user requirements from a functional perspective and its constructs map well to the constructs of UML and other specification techniques. COSMIC-FFP introduces new concepts (viewpoints and layers) for measuring functionality delivered by non-business applications software.

4. COMPARISON OF FPA AND COSMIC-FFP FOR NON-BUSINESS APPLICATIONS SOFTWARE

The following section illustrates how the conceptual differences between FPA and COSMIC-FFP affect their capability to measure functional size in non-business applications software.

4.1. Identifying functionality delivered to the end user

In the past, functional size measurements using FPA have usually only included functionality of processes that send or receive data to or from the external user boundary. I.e. FPA is usually used to report the size of the functions delivered to the external business user. In contrast, the end users of non-business software are primarily other software components in either the same or another architectural layer. Many outsourcing pricing models are based on payment for function points delivered (added, changed or deleted) to the end business user. Contention arises when the customer requests infrastructure functionality to be developed or changed and the contract only allows the supplier to charge on function points affected. (E.g. A client request is to improve the performance of all database accesses. The developers do this by modifying the archiving utility software so that its business rules now selectively archive any obsolete data, thus reducing database access time). The quality of the end user functionality (i.e., performance) is improved but the functional size of the business applications software is not considered impacted and therefore, the effort required is not chargeable. FPA specialists consider performance as a general system characteristic rather than functionality changed in software. Issues also arise when a project needs to build infrastructure software in addition to the Business Application software but the infrastructure software is not included in the development project’s functional size. Consequently, the impact on the effort for building the infrastructure continues on page 15

software is not adequately catered for by the adjustment for the technical and quality features in the Value Adjustment Factor of the Business Application. The COSMIC-FFP approach, from the functional user requirements perspective instead of external user view, enables the functionality delivered by non-business software, residing in other architectural layers, to be consistently identified and effectively measured. It provides specific guidelines on how to identify software delivered in different architectural layers and how to size their interactions. I.e. when estimating projects, it enables infrastructure software, which supports the Business Applications, to be effectively sized, and their productivity rates and resource estimates to be separately established.

4.2. Measuring internal and external sub-processes

FPA measures functionality by evaluating the amount of data that crosses the external boundary during the processing of the process. The processes within Business Applications software tend to be primarily involved with entering and extracting data so the amount of data movement is a good indicator of overall process size. However, problems arise with non-business software where there is significant internal processing (e.g. Read and Write data movements) compared to the processing required to move the data into or outside the boundary. This poses a problem in outsourcing contracts where the performance of the suppliers is measured in Function Points delivered. Since any process that has significant internal processing, but little external visibility, will not be sufficiently accredited for the full amount of functionality it delivers, this will result in low-recorded productivity rates for the project. Although considerable effort may have been expended adding or changing the internal logic within a process, if it only accepts a minimal number of data items then its FPA function points will be low. For example, a process, which receives the coordinates of a radar system and only sends out confirmation or an alarm, may have significant internal processing to check positioning and identify exception conditions (e.g. accesses many control and reference tables). FPA only measures the external data movements and restricts the ‘recognition’ of data accesses to a maximum of three, which often does not represent an adequate measure of the functionality delivered by this process.

In comparison, COSMIC-FFP measures all the unique sub-processes within a process, i.e. not only the data entering and exiting the process but the internal sub-processes of reading and writing to the data groups. By measuring and recognising all the functionality that a process is required to deliver, rather than just the external aspects of a process and limited data accesses, it captures the full complement of functionality for estimating and productivity comparisons.

4.3. Categorising processes which do not have predominant role

FPA requires a process to have a predominant role of either entering or extracting data. This predominant role determines the process type (Input, Output or Enquiry). However, the processes within non-business software tend not to have a predominant role making them difficult to categorise unequivocally into one of those process types, since they often:

- Accept data which enters one side of the application boundary, process it and send the results immediately externally across the boundary to another application in another layer. (e.g. translation process in a gateway protocol converter or extraction process in screen scraper software)
- Involve the processing of multiple and variable sets of incoming and outgoing data. Neither side of the process is predominant (e.g. processes in real-time equipment monitoring).

It is difficult to consistently measure these types of processes since the same process may be categorised differently each time it is sized, resulting in a range of reported functional sizes for the same application. I.e. different people will justifiably categorise the same process as an Input, Output or Enquiry each of which will be allocated a different number of function points. Variations in counting results cause contractual disputes, particularly when performance targets fall within the error boundaries of the measures. Variations in count results also undermine the credibility of Functional Size Measurement as a valid means of consistently measuring size.

In comparison, COSMIC-FFP assesses sub-processes within processes. The number of points awarded is independent of the type of sub-process; just its existence is recognised. Since the direction of the data movement for a sub-process is quite obvious, variations in categorisations are uncommon. However, since all sub-processes are attributed one unit of functional size irrespective of type, any errors or variations in categorisation do not affect the measured size of a process. The resultant size tends to be more consistently measured.

4.4. Sensitivity to large variations in functionality delivered by processes

The FPA rating scale used to award function points to a process increases by just over two fold between the points awarded for the simplest process (e.g. Low complexity input is awarded 3 points) and the points awarded for the most complex process (high complexity output is awarded 7 points). The coarseness of the FPA measure is usable and acceptable for business applications where, in most cases, a two-fold difference in size is representative of the range of functionality delivered by most processes. The categorisation of size using ‘ranges’ also assists in the speed of function point counting with the IFPUG method.
A Method for Sizing

Continued from page 15.

However, the processes delivered by non-business software have been found in practice to deliver a much broader spectrum of functionality than a two-fold difference in size. Yet, no matter how complex the functionality delivered by a process IFPUG FPA cannot award it points beyond the maximum upper limit. This can cause significant errors in estimates of effort for a requirement to change a number of very complex processes. This is a concern in an outsourcing contract where a requirement of the contract is to provide fixed price estimates on projects using historical productivity rates.

The COSMIC-FFP measure is not restricted to awarding a maximum number of points to any one process. It is therefore much more sensitive to large variations in size of processes experienced in non-business applications.

4.5. Measuring process-rich, data-poor software

FPA measures the data groups accessed by the processes as one of the major contributors to functional size. FPA is based on the concept that maintaining and reporting on data is the software’s primary role. It includes within its total size the function points awarded to the logical data groups accessed by the application. These data groups also contribute to the functionality attributed to each process that accesses the data groups. However, non-business software does not have the same emphasis on stored and maintained data. Their processes often operate by referencing relatively static threshold values and parameter controls to make decisions on the data to output. The processes may involve multiple unique sub-processes which reference and update fields in these data groups but the data itself is often minimal compared to the many steps to analyse it and react appropriately. The data input to the process is not usually permanently stored but used for the duration of the process. The permanent data that is accessed is relatively simple and usually consists of historical logs, threshold values or parameter controls.

Difficulties arise when using FPA to measure non-business software since FPA uses the amount of stored data as a significant factor in determining the functional size of the application (usually contributes around 25% - 30% of measured size). Where the stored data is simple but the processing of the stored values is complex, the functional size of the application is underestimated. In contrast, COSMIC-FFP determines the size of a process by measuring the number of unique accesses (Reads and /or Writes) to data groups with no upper limit on the number of Data Groups accessed by a process. Thus, COSMIC-FFP gives a better indication of size for applications that have a few groups of simple data but complex use of that data. COSMIC-FFP also measures both the Read and Write accesses as two sub-processes within a Process, compared to IFPUG FPA, which does not recognise additional logic to both extract data from a data group and update it in the same process.

6. CONCLUSION

All software developed and supported by the supplier in an outsourcing contract, needs to be able to be measured in order to provide input into performance monitoring. This software usually includes applications, which do not deliver functionality directly to the human business users. Experience showed that the IFPUG FPA functional sizing technique was not well suited to measuring software that has other software (particularly in other architectural layers) as its primary users. Early industry results have indicated that the COSMIC-FFP technique provides a more effective method of measuring these infrastructure, real-time, embedded and process control software applications (non-business software), than IFPUG FPA.

COSMIC-FFP’s ability to measure consistently across layers, in a repeatable way, by different people, and effectively (i.e. measure all the functionality delivered by these types of applications), makes it a good candidate to be considered when measuring all the software involved in monitoring a contract.

7. BIBLIOGRAPHY


continues on page 17
A Method for Sizing  Continued from page 16.

ISBSG Release 9 Benchmarking Repository – WWW.isbsg.org
Morris, P.’ Similarities and Differences between IFPUG and COSMIC Functional Size Measurement Methods’ IFPUG Arizona, USA, September 2003
Morris, P,’ Impact of Standardization on Functional Size’, AEMES conference, Madrid, Spain, October 2005

About the Author
Pam Morris is the CEO of Total Metrics, founded in 1994 in response to the software industry’s need to better manage and control their software development processes. She has a B.Sc., Graduate Diploma of Computing and Diploma of Education from La Trobe and Monash Universities, Australia.

She is the past president of the Australian Software Metrics Association (ASMA) where she currently holds a position on their Executive and the Benchmarking Database Special Interest groups. She represents Standards Australia as the international project editor of the ISO standard 14143-1 and 2 for Functional Size Measurement. She was the international convenor of ISO/IEC/WG12 group developing FSM standards from 1997 to 2004. She plays an active role internationally in the development of measurement standards and she was a member of the International Function Point User Group (IFPUG) Counting Practices Committee in the USA from 1993 to 2000. She is a member of the COSMIC-FFP Core Group that is responsible for the development of the COSMIC-FFP FSM method. She has been an IFPUG Certified Function Point Specialist (CFPS) since 1994 and a Certified Software Measurement Specialist (CSMS Level 3) since 2006.

Pam has provided consulting services and presented training courses to over 200 corporate and government organizations in the USA, Japan, Australia, India and New Zealand since 1991.

Author Contact Information
Post: Total Metrics
Suite 1, 667 Burke Road
Camberwell Victoria 3124 Australia
Email: pam.morris@totalmetrics.com
Phone: 61(0)3 9882 7611
Fax: 61(0)3 9882 7633
Visit us http://www.totalmetrics.com/
Estimation Models for Software Maintenance Based on Functional Size

By Alain Abran, École de Technologie Supérieure – Université du Québec

Abstract

This paper illustrates the use of functional size measures in building estimation models for maintenance projects implementing small functional enhancements in existing software. More specifically, it reports on estimation models built with 19 maintenance projects on a single real-time embedded software application in the defence industry. Functional size measures were collected using the COSMIC-FFP functional size measurement method and the maintenance projects were classified into two classes of project difficulty to identify sub-sets of projects with greater homogeneity in their relationship of project effort with respect to functional size.

1. INTRODUCTION

Software maintenance costs usually exceed software development costs and Ferens [1] indicates that the attention given to these costs has not been commensurate with their importance: for instance, only a handful of estimation models have been proposed for software maintenance while a significant number have been proposed for development projects. By the early 1980s few attempts had been made at software maintenance estimation. There was the COCOMO-M(aintenance) model, with a single additional maintenance-unique input, annual change traffic[2]. In this model, it is the cost of the whole maintenance life cycle that is estimated, over a period of time; with such a model, no attempt is made to estimate one maintenance project at a time. Ferens [1] reports that SLIM has a single maintenance-unique input, while PRICE-S, SEER-SEM and CHECKPOINT have multiple inputs. However, Ferens points out that the accuracy of such maintenance estimation models in their early analyses was low and that, in general, the performance of maintenance estimation models has yet to be demonstrated [1].

Estimation models are based on the generic concept of productivity defined as the ratio of output to input. In software projects, productivity is defined as the ratio of the software product developed to the resources required to produce it. While ‘effort’ is the generally accepted measure of the input (often measured in person-hours, -days or -months), software size is recognized as a key factor in the construction of models estimating project effort.

In addition to size, it is recognized that other factors affect effort needed for doing maintenance projects, such as type of application, programming language, age of software, quality of the documentation available. In the context reported here, the maintenance projects measured and analysed were carried out on the same software application, in the respective organizations; this means therefore that these specific factors were held constant and did not need to be taken into account in building the estimation models within the same application under maintenance. Still, many other factors can influence maintenance projects effort, in addition to size: for instance, whether or not a full system test is required or not, severe constraints on resources availability, functional complexity, technical complexity, low or high level of reuse, etc. In the study reported here, these individual factors were not investigated independently, but were rather represented through a single factor referred to as ‘project difficulty’.

Generally, factors such as ‘project difficulty’ are represented as categorical variables which are not intended to represent quantitative values. Such categories are used to identify distinct groups of data to be analyzed. The variables, by contrast, are used when categorical factors can affect the positive relationship of variables in regression models. Therefore, they can be considered as explanatory variables in estimation models.

This paper reports on the use of this new generation of functional size measure (COSMIC-FFP) in building productivity and estimation models for measuring the effort involved in small projects implementing functional enhancements in the context of the maintenance of real-time software in the defense industry. The information reported here is a summary of the analysis of the data set from the defence industry reported in full by Abran et al. in [3]. This same reference includes as well examples of estimation models built for a set of 15 maintenance projects on a web-based software application in the linguistic domain. Nagano [4] presents as well estimation models for a set of maintenance projects on a telecom switching system, all measured with COSMIC-FFP.

This paper is organized as follows: the data set is presented in section 2, the simple regression models with function size only in section 3, and the multiple regression models in section 4. A discussion is presented in section 5. While measurement experts will be interested in the details of the equations and of the statistical tests, managers should focus on the approach used, and the graphical interpretations as well as on the impact for estimation purposes in software maintenance.

2. DATA SET

2.1 Context

This data set comes from an organization which designs, develops and implements systems for the defense industry. It is a subsidiary of an international organization, and the software unit of this organization develops and maintains real-time embedded software. This organization did not have a measurement program in place, neither had it developed its own estimation models (either for development or for maintenance projects).
data collected were verified by subject matter experts from the industrial organization.

2.2 Functional size with COSMIC-FFP

The COSMIC-FFP functional sizing method[5,6] was designed to work equally well for ‘data-rich’ business/MIS software and for ‘control-rich’ or ‘real-time’ software, that is, the software typically found in telecoms, avionics and process control, and in embedded and operating systems. The method also makes it possible to size such software in any layer or peer item of a multi-layer and/or multi-tier architecture. In the COSMIC-FFP method, size is measured in Cfsu units (that is: COSMIC functional size units), which is equivalent to one data movement of one data group.

In a maintenance context, the functional size of a change to the Functional User Requirements within each piece of software is calculated by aggregating the sizes of the corresponding impacted data movements according to the following formula:

\[
\text{Size}_{\text{Cfsu}}(\text{Change}) = \text{size(added data movement}_i) + \text{size(changed data movement}_i) + \text{size(deleted data movement}_i)
\]

The COSMIC-FFP method was designed to enable it to discriminate among the sizes of small functions -- it uses a unitary size unit rather than the stepwise functions of the first generation measurement methods. Therefore, this COSMIC-FFP method opens up the possibility of building estimation models for real-time software based on functional size, not only for development projects, but also for small maintenance projects. COSMIC-FFP is supported by the Common Software Measurement International Consortium (COSMIC) formed in 1998 to design and bring to market a new generation of functional size measurement methods. Overall, close to 40 people from 8 countries participated in the design of this measurement method, and the Measurement Manual describing the method is available for free download in English, French, Japanese and Spanish [5]. It was adopted in 2003 as an international standard by ISO (ISO 19761) [6].

For this study, functional size was measured afterwards on completed maintenance projects.

2.3 Effort

The work effort (in person-hours) was obtained from the organization’s time reporting system: this corporate time reporting system allowed us to identify the effort expended on each specific functional maintenance project. However, for some of the projects the effort expended for the analysis phase of a project was not noted; therefore, only the effort expended, excluding analysis, for all projects was taken into account.

2.4 Project difficulty

The project difficulty category for a maintenance project was assigned by the staff who had carried out the maintenance projects. They did this on the basis of project documentation and their own experience; in industry, this is referred to as assignment of values by subject matter experts. At measurement time, four levels of project difficulty were identified on the following scale: 1, 3, 5 and 7, corresponding to no difficulty, difficult, highly difficult and extremely difficult. The use of these four initial difficulty levels in a sample of this size (21 observations) was problematic from a statistical viewpoint, as there were insufficient data observations for certain difficulty values (only 1 or 2 observations), which means that some difficulty categories would not be representative enough when building estimation models. A simpler classification of a categorical variable is then desirable. To do this, the difficulty variable with four levels was reclassified into a two-level classification (low and high).

2.5 Data sample

This industrial data set included information on 21 maintenance projects implementing functional enhancements to the software components of a defence system. Figure 1 presents a graphical representation of this data set, which contained the functional size and effort expended for each of the 21 projects.

When there are good reasons to believe that some outliers are not representative of the data set under study, they should be taken out of the sample being analyzed. Using the statistical tests described in Box 1, two such outliers were identified and dropped out from further analysis.

A visual analysis of this data set of 19 projects without outliers continues on page 20.
Box 1: Statistical tests used for identifying the outliers

The outliers were analyzed with the Grubbs method [7]. The first statistical analysis looked into the distribution of the variables and for the presence of candidate outliers: the Kolmogorov-Smirnov test indicated that the CfSU modified variable was not normally distributed. The Grubbs test indicated that project no. 2 (with 956 hours) and project no. 21 (with 724 hours) were far from the set of observations: the distance of these two projects from the average of the population (on either size or effort) can be measured in terms of multiples of sigma (the standard deviation of the variable), two standard deviations in this case, which means that these projects can be considered as outliers in the population under study. Without these two outliers, the distribution of the sample is closer to a normal distribution, based on the Kolmogorov-Smirnov test. The averages for both functional size and effort are significantly reduced when these two outliers are excluded: from 72 down to 39 CfSU and from 184 to 115 hours on average.

Box 2: Statistical criteria for assessing estimation models

There exist multiple statistical criteria for assessing whether or not a model reflects a data set, and its ability to predict the behavior of the dependent variable. For example, the coefficient of determination ($R^2$) and the coefficient of variation give an indication of the quality of the model’s adjustment to the data. $Pred(X)$ also provides an assessment of the precision of the model. To assess the quality of the estimation models to be built for functional enhancement in a maintenance context, the following statistical elements were selected:

Mean Relative Error (MRE), which gives an indication of the divergence between the values estimated by the model and the real values, as a percentage. Perfect estimates would give a MRE of zero.

Predictive quality of the model: In software engineering, an estimation model is generally considered good when the MRE (Mean Relative Error) is within +/-25% for 75% of the observations [8], or $Pred (0.25) = 0.75$.

Coefficient of determination ($R^2$): Another measure of the quality of a model is its coefficient of determination ($R^2$), which describes the percentage of variability explained by the predictive variable in the linear regression models. This coefficient has a value between 0 and 1; when an $R^2$ is close to 1 it indicates that the variability in the response to the predictive variable can be explained by the model, that is, there is a strong relationship between the independent and dependent variables.

The linear regression model with the functional size as the independent variable gives, for the sample of 19 observations, the following linear equation (Figure 3): Effort = 0.61 x CfSU + 91 ($R^2 = 0.12; n=19$)

This linear model is not strongly positive, with a regression coefficient $R^2$ of only 0.12, which means that only 12% of the total variability of effort in the projects is explained by the variation on its functional size, as measured in CfSU.

See Box 2 for information about criteria for assessing the reliability of estimation models.

Figure 2: Data graph excluding 2 outliers (N = 19)
3.2 Nonlinear regression models with functional size

Other forms of regression model have already been investigated, and these results are presented in Table 1, where R is the coefficient of correlation between the actual values of Y and the values derived from the equation. High values of R (maximum = 1.0) would indicate a high correlation, and R² is the percentage of the variance of the dependent variable which can be explained by the given equation. From Table 1, it can be observed that none of the non-linear regression models represents a significant improvement over the linear model.

### 4. MULTIPLE REGRESSION MODELS WITH TWO INDEPENDENT VARIABLES

#### 4.1 Multiple regression models – additive form

The next regression model constructed takes into account the categorical factor of the low-high level of the difficulty factor:

- Difficulty = 1 -> if in the “high” level
- Difficulty = 0 -> if in the “low” level

Then, an additive model with the low-high difficulty factor gives each level the same importance in the relationship between size and work effort. It takes the following form: y = ax + bz + c

For this sample of 19 projects, the general form of the model is:

\[ Y = 0.92 \times C_{fsu} + 126.12 \times difficulty + 26 \]

This model, with the difficulty level variable, has a coefficient of determination R² = 0.46, which is better than the simple linear regression model, but still not good enough. It can be observed in Figure 4 that both regression lines have the same slope (0.92), and are represented by parallel lines with different points at the origin when C_{fsu} = 0. This is typical of additive models.

#### 4.2 Multiple regression model: multiplicative form

Regression models with multiple independent variables (functional size and number of programs modified) is:

\[ Y = a \times C_{fsu} + b \times (\text{no. of programs modified}) + c \]

\[ Y = 0.78 \times C_{fsu} - 3.62 \times (\text{no. of programs modified}) + 98 \]

This multiple regression model with the number of programs modified has the same value for its R², 0.12, which is not an improvement over the simple linear regression model.

### Table 1: Nonlinear Regression Models (N = 19)

<table>
<thead>
<tr>
<th>Model</th>
<th>N</th>
<th>A</th>
<th>B</th>
<th>R</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y = A \times X^B</td>
<td>19</td>
<td>43.808</td>
<td>0.245</td>
<td>0.50</td>
<td>0.245</td>
</tr>
<tr>
<td>Y = A \times e^{(B \times X)}</td>
<td>19</td>
<td>63.067</td>
<td>0.006</td>
<td>0.39</td>
<td>0.15</td>
</tr>
<tr>
<td>Y = A + B \times \ln(X)</td>
<td>19</td>
<td>44.121</td>
<td>29.29</td>
<td>0.51</td>
<td>0.26</td>
</tr>
<tr>
<td>Y = A + \frac{B}{X}</td>
<td>19</td>
<td>132.463</td>
<td>-48.330</td>
<td>0.32</td>
<td>0.10</td>
</tr>
<tr>
<td>= \frac{1}{(A + (B \times X))}</td>
<td>19</td>
<td>0.022</td>
<td>-8.8E-05</td>
<td>0.31</td>
<td>0.09</td>
</tr>
</tbody>
</table>

When taking the low-high difficulty level into account, the following two models, as illustrated in Figure 4, are obtained:

- If the Difficulty = 0 -> Effort = 0.92 C_{fsu} + 26
- If the Difficulty = 1 -> Effort = 0.92 C_{fsu} + 152

This model, with the difficulty level variable, has a coefficient of determination R² = 0.46, which is better than the simple linear regression model, but still not good enough. It can be observed in Figure 4 that both regression lines have the same slope (0.92), and are represented by parallel lines with different points at the origin when C_{fsu} = 0. This is typical of additive models.

4.2 Multiple regression model: multiplicative form

---

Data & Analysis Center for Software (DACS)
With the additive model, the size effect is not impacted by the values of the low-high difficulty class variable, and hence the impact of the increase in both variables is determined by constant additive values; that is, the impact of the size variable is measured independently of the low-high difficulty variable. To improve this regression model, as recommended in [7], a new variable is added, that is, the interaction of difficulty and size, as represented by the multiplication of the two variables (Difficulty x \(C_{fsu}\)).

The inclusion of the variable in the model makes it possible to recognize the multiplicative impact of these two variables on the positive relation between size and effort. Of course, it eliminates the parallelism of the two lines in the additive model. The general form of the multiplicative model is the following:

\[
Y = X + Z + (X \times Z) + \text{Difficult}\ y \text{ness} + (C_{fsu} \times \text{Difficulty}) + \text{Constant}
\]

The difficulty variable, represented by \(X\), has an influence on the behavior of the size variable \(C_{fsu}\), thereby modifying the slope and the constant of the curve during the analysis of its values of 0 and 1. The general multiple linear regression equation obtained is:

\[
\text{Effort} = 0.64 \times C_{fsu} + 41.94 \ \text{Difficulty} + 3.85 \times (C_{fsu} \times \text{Difficulty}) + 41
\]

With \(R^2 = 0.75\), standard error = 57.83 and \(n = 19\)

This multiplicative model has a coefficient of determination \(R^2 = 0.75\), which is a significant improvement over both the model with one variable and the additive model with two independent variables. Furthermore, the coefficient of the categorical variable linked to both difficulty and size is statistically significant, that is, it has a p value < 0.05. The specific equations for each difficulty level are as follows:

- If difficulty = 0 → Effort = 0.64 \times C_{fsu} + 41 with \(R^2 = 0.47\) and \(n = 8\)
- If difficulty = 1 → Effort = 4.49 \times C_{fsu} + 82.94 with \(R^2 = 0.78\) and \(n = 11\)

These equations are presented in Figure 5 and clearly illustrate the fact that the effort level of projects depends both on their functional size and on their difficulty level as significant variables to be taken into account during estimation.

In addition, the graphical analysis in Figures 4 and 5 shows that the largest project in terms of size (for project no. 1, size = 216 \(C_{fsu}\)), is classified in the low difficulty category, and has a much lower level of effort than projects of smaller size, which is not consistent with the balance of the data set. The behavior of project no. 1 could be considered to be significantly different from the others in terms of some unspecified factor. It could therefore be excluded from the sample and reserved for further analysis to verify its impact on the multiplicative model. On this reduced sample of 18 projects, we apply the general multiplicative model given by \((Y = X + Z + (X \times Z) + \ldots)\), which gives:

\[
\text{Effort} = 1.25 \times C_{fsu} + 56 \ \text{Difficulty} + 3.24 \times (C_{fsu} \times \text{Difficulty}) + 27
\]

With \(R^2 = 0.84\) and \(n = 18\)

Figure 4: Additive Model (N = 19)

Figure 5: Multiplicative Model (N = 19)
The specific models for each difficulty class are then as follows:

If difficulty = 0 \rightarrow Effort = 1.25 \ C_{\text{fsu}} + 27 \text{ with } R^2 = 0.87 \text{ and } n = 8

If difficulty = 1 \rightarrow Effort = 4.49 \ C_{\text{fsu}} + 83 \text{ with } R^2 = 0.78 \text{ and } n = 10

With a regression coefficient $R^2 = 0.84$, this model is better than the previous one, and the $C_{\text{fsu}}$ variable is statistically significant with a $p$ value < 0.05, as is the multiplicative term of both variables (Figure 6). Box 3 presents a comparison of the results of the multiplicative models for both sample $N=19$ and sample $N=18$.

5. DISCUSSION

Software size is recognized as a key factor in the construction of models to estimate project effort. Software size can be measured from either a technical perspective, with Lines of Code for example, or from a functional perspective. While functional size measures have been available for software of the MIS type for the past twenty years, the industry has not yet felt that it was appropriate for real-time and embedded software. The COSMIC-FFP new generation of functional size measures has addressed this domain issue. In addition, the COSMIC-FFP method was designed with a much more granular scale, which enables it to discriminate among the various sizes of small functions – through the use of a unitary size unit rather than stepwise functions in the first generation measurement method. This COSMIC-FFP method thus makes it possible to build estimation models for real-time software based on functional size, and to do this not only for development projects, but for small maintenance projects as well.

This data set included maintenance projects modifying a single software application. The functional size has been measured with the COSMIC-FFP standard.

When the data set is graphically represented ($x =$ functional size, $y =$ effort), there is an indication of a position relationship between size and effort, but this relationship is not strong; both patterns have a wedge shape, which indicates that, for these organizations, at least one other factor has a significant influence on project effort.

Estimation models built with either an average unit cost or a simple regression model provided interesting models for both organizations, but they did not model the relationship between size and effort well for the individual projects. For this data set, even though there is a clear positive relationship between functional size and project effort, such a relation was not strong enough to

---

**Box 3: Comparison of models quality criteria**

The criteria described in Box 2 are used to compare the results of the multiplicative models for both sample $N=19$ and sample $N=18$.

<table>
<thead>
<tr>
<th>Model (Sample size)</th>
<th>$R^2$</th>
<th>PRED (+/-25%)</th>
<th>PRED (+/-30%)</th>
<th>PRED (+/-35%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># Projects   %</td>
<td># Projects   %</td>
<td># Projects   %</td>
</tr>
<tr>
<td>Multiplicative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(N=19)</td>
<td>75</td>
<td>10</td>
<td>52.6</td>
<td>12</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>84</td>
<td>10</td>
<td>55.5</td>
<td>12</td>
</tr>
</tbody>
</table>

In addition to the improvement in the regression coefficient, the MRE has moved down from 0.51 to 0.40. It is, of course, still higher than the 25% recommended by Conte et al. [8], but with a (PRED 25%) of 56% for 10 projects. This is a significant improvement, with some other projects fairly close to this level. With PRED 35%, 77% of the projects are inside this interval, which is quite good in the context of software maintenance where there could be considerable variation due to individuals (which is not mitigated by a team balanced by management for larger development projects).

continues on page 24
derive good estimation models using this single independent variable with either the average unit costs models or the linear and nonlinear forms of simple regression models: both data sets had a graphical distribution on the two axes and were wedge-shaped, illustrating that the single size variable is not enough for building a good estimation model.

Two forms of multiple regression models were then investigated: additive models and multiplicative multiple regression models. While regular multiple regression gives a single model, both the additive and the multiplicative models lead to two models, one for each class of the second independent variable. In the additive model, the two submodels are represented by two parallel lines, with the same slope and a different point of origin; in the multiplicative model, the two submodels do not need to be parallel lines. The multiplicative models graphically mapped the two classes of projects much better (low-high) with respect to the other independent variable, functional size, and their criteria as good models of the relationship between size and work effort were much better.

These results are of particular interest in the development of estimation models for maintenance projects: the functional size of a maintenance enhancement can be measured very early on from the requirements themselves, and therefore this size can be obtained fairly accurately before any programming needs to be done (e.g. functional size does not need to be estimated with an associated level of uncertainty, as input to the estimation model).

These various analyses have provided insights into maintenance work, which consists of small functional enhancements and their relationships with work effort. A process for deriving estimation models has also been illustrated, with both an objectively derived quantitative variable (functional size), obtainable early on in the project life cycle, and a categorical variable, difficulty. Once such models are derived, and their quality analyzed, then they can be quite useful in estimating subsequent maintenance projects involving adding or modifying functions to these software applications.

6. REFERENCES
Estimation Models
Continued from page 24.

About the Author

Dr. Alain Abran is a Professor and the Director of the Software Engineering Research Laboratory at the École de Technologie Supérieure (ETS) - Université du Québec.

He is currently Co-executive editor of the Guide to the Software Engineering Body of Knowledge project. He is also actively involved in international software engineering standards and is Co-chair of the Common Software Metrics International Consortium (COSMIC).

Dr. Abran has more than 20 years of industry experience in information systems development and software engineering. The maintenance measurement program he developed and implemented at Montreal Trust, Canada, received one of the 1993 Best of the Best awards from the Quality Assurance Institute.

About Contact Information

Professor
Department of Software and IT Engineering
École de technologie supérieure – Université du Québec
1100, Notre-Dame Street West
Montréal, Québec
Canada H3C 1K3

Phone: +1 (514) 396-8632
Fax: +1 (514) 396 8684
e-mail: Alain.Abran@etsmtl.ca
URL: www.gelog.etsmtl.ca
As the battered and bruised project manager of several major software engineering projects, by the late 1980s I had had enough. Enough of being expected to deliver to budgets and timelines that were clearly unrealistic. Enough of reporting to project boards that didn’t believe a word I was saying. Enough of the feel-good rhetoric within the software engineering profession about improving productivity and adopting best practice when we had no idea what constituted either.

The cynicism of our customers of the software engineering industry’s ability to deliver was well founded. Around 30% of projects failed to deliver anything and, of those that did complete, the average blow out was in excess of 80%. Business managers were living in fear of failure of their critical software investments.

I had a life-long friend who was much smarter than me. He had decided on a career as a project manager in building construction and, as I often pointed out over a few drinks, his job was much easier than mine. His industry had a large knowledge base of past projects and a specific profession dedicated to interpreting and applying that knowledge so they could predict, manage changes and benchmark productivity. He had a Quantity Surveyor sitting next to him advising him of the differences between fact and fantasy, the likely impact of changes and generally controlling the scope. His situation was similar to that of the long-distance aeroplane pilots of yesteryear who had navigators that told them exactly where to fly to get to their destination while avoiding potential risky situations.

My frustration was being shared globally and resulted in the creation of a number of national not-for-profit industry bodies that were focussed on providing some real data that would improve our ability to manage software engineering projects. The discipline of creating fact-based knowledge to support software engineering was globally known as software metrics.

Creation of a Knowledge Base for Software Engineering Projects

With the growth of these new national software metrics organisations in the late 80s and early 90s, a cooperative global movement started and continues to this day. This resulted in the development of an effective measure of the size of the software engineering product (functional size measure), the definition of ‘productivity’ as it relates to software engineering and the collection of the data of completed projects against these standards. Productivity was defined as the number of hours it takes to deliver one functional unit of software into production. It is expressed as ‘hours per function point’. Specific definition of when a project starts and ends and the effort levels that are included can be found in the data collection questionnaire that can be downloaded at www.isbsg.org. The progress made in this area has been outstanding. In 1990 there was no standard definition of productivity and no publicly available data. Today the International Software Benchmarking Standards Group (ISBSG) has a publicly available repository and learnings based on over 4,000 validated projects from 40 countries. Understanding and applying this knowledge is a specialised skill and not one that all project managers can be expected to have.

By 1995 it became clear that, if the knowledge we had amassed was properly applied, I would be as capable as my friend in building construction in developing early and defendable estimates of size, cost and duration, in managing change and in performance benchmarking.

Time for Change…..The Creation of southernSCOPE

When it came to acquiring custom-built software, government and business had two buying options. The first was ‘fixed price’ where they specify their requirements and assess quotations from suppliers to select the ‘best’ quote. The second was ‘time and materials’ where they pay a preferred supplier (or their internal staff) on the basis of time spent developing the product.

The common approach our industry takes in developing software requirements also seemed to me to be flawed to the point that in most cases it will inevitably result in tears. If I approach an architect to design a new home his first question to me is, ‘What is your budget?’ Based on this budget the remainder of our discussions centre on the features I most need and how these might be delivered within the nominated budget. We don’t do this in software engineering. Our architect starts by helping the customer create a comprehensive ‘dream list’ of anything we can both think of that might be nice. It is an exhilarating experience. Upon completion, we use our individual brands of black magic to arrive at the likely delivery cost (complete with whatever caveats we can think of). It is normal for the cost to be many times what the customer was prepared to pay, and at this point, the honeymoon is well and truly over. The relationship is in demise.

In 1995 with our newfound knowledge in hand, it was time to try to right a few of the old wrongs. The Government of the State of Victoria in Australia developed southernSCOPE, an alternative approach to acquiring custom built software that would try to address some of the imbedded problems. It also established a policy making southernSCOPE the preferred method that agencies of government should use to acquire their custom-build software. In 2000 a study of the extent and impact of its use was undertaken and resulted in the release of a revised approach. Although the number of projects that had used the
approach was small their nature was diverse. The results were outstanding:

- All projects completed within 10% of the initial project budget
- They all had a high customer satisfaction in that the software met the intended business need
- Their cost per unit was in the lowest 25% of comparative industry benchmarks.

Why then is the southernSCOPE method so effective? The method successfully enabled the following problems inherent in the software engineering process to be addressed:

- Realistic cost estimates are provided at project inception
- The functional requirements developed and agreed to are sound and unambiguous
- The customer is able to make objective decisions in language he understands as to what functionality should be provided within the agreed budget
- Scope creep is mitigated

How does southernSCOPE work?

In effect, southernSCOPE is just an aggregation of the practical capabilities made possible through the new knowledge base applied at various places in the system development lifecycle. Whereas for the maximum benefit it should be applied from project inception it can also be applied at several points throughout the project lifecycle with good impact. The approach comprises the following steps (see Table 1):

What is the Role of the ‘Scope Surveyor’?

Critical to the above approach is the engagement of a suitably skilled ‘Scope Surveyor’ that is contracted to the project throughout the development lifecycle by the customer/acquirer. This person provides the early budget estimate (Step 1), leads in the development of the Project Requirements document (Step 2), oversees the completeness of the functional requirements document (Step 3) and the development of the Function Baseline (Step 4), monitors the delivery of the required functionality and any change management issues (Steps 5 and 6) and plays a key role in the finalisation of the project (Step 6). They must be expert in the global software metrics knowledge base and its practical application as well as having a strong background in software engineering. The cost of this resource is normally between 1 and 2 percent of the project cost. Since part of their role is to be a ‘referee’ about what functionality is in and out of scope, there is an escalation process to an independent authority for use in the case of a dispute between the supplier and purchaser. To my knowledge, this escalation has never been enacted. As with the building engineering industry, as our industry matures, for particularly large projects both the supplier and the purchaser may elect to engage Scope Surveyors to provide key expert cost, scope and delivery functions to both parties.

The southernSCOPE approach is particularly effective when the functional requirements are not well known as it provides for the analysis and ultimate selection of the functionality that will best meet the business need. It is like having a ‘time-box of functionality’ to which the project works. An early estimate of functional size for an identified business might indicate that, for example, 800 functional units of software would be sufficient to meet that need. Once the quantity is established, the project activity that follows focuses on identifying what specific functionality should fit into the allocation of functional size and then ensuring it is developed and delivered. As functional size measures (FSMs) have now been developed that are particularly

Table 1: southernSCOPE Acquisition Approach

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Lifecycle stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Customer obtains a realistic estimation of size, cost and development time for a software project that would satisfy the identified business need</td>
<td>Project Initiation</td>
</tr>
<tr>
<td>2</td>
<td>Customer develops a Project Requirements document, a brief and simple document that defines the business need for the software (including the likely functional size), and invites suppliers to make proposals charged on a ‘cost per delivered unit’ basis</td>
<td>Project Initiation</td>
</tr>
<tr>
<td>3</td>
<td>Successful supplier develops a functional requirements document</td>
<td>Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Customer decides which of the identified functionality should be included within the budget package. This is known as the ‘function baseline’ against which the subsequent delivery and any functional changes will be measured.</td>
<td>Analysis</td>
</tr>
<tr>
<td>5</td>
<td>Supplier develops agreed software</td>
<td>Build</td>
</tr>
<tr>
<td>6</td>
<td>Software is accepted by the customer and payment is finalised taking into account any agreed changes</td>
<td>Implementation</td>
</tr>
</tbody>
</table>

The use of a Scope Surveyor is not limited to projects that are using the cost-per-functional-unit acquisition approach but is just as effective on fixed-price and time-and-matters. In fixed-price acquisitions they play a critical role in evaluating proposed responses to identify the level of functional fit and ultimate cost of the competing proposals.

continues on page 28
suited to embedded real time systems (such as COSMIC) the southernSCOPE should have similar relevance in these environments.

**How widespread is its use?**

This is the sad part. This approach was developed ten years ago. Five years ago a study of its effectiveness found it to be both innovative and able to solve some long standing problems. Today, its known use is scattered a little around Europe and in Brazil. Within the Victorian government, there has been scattered use of the approach mainly on smaller projects but there have been some interesting developments.

The Victorian Government has an Office of the Chief Information Officer that has the responsibility for driving best practices across the Victorian government. To this end, since 2002 that Office has funded the engagement of a Scope Surveyor for four projects critical to government. In each case, the story has been the same (see Table 2):

<table>
<thead>
<tr>
<th>Table 2: Observations of Recent Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The project manager agrees that the project will run using the southernSCOPE approach and a Scope Surveyor is contracted. The Scope Surveyor provides some valuable upfront estimating information for which the Project Manager is grateful.</td>
</tr>
<tr>
<td>2. The project manager then decides that he now has good control and, while still calling it a southernSCOPE approach, decides to revert to a fixed price arrangement. Despite persistent contacts from the contracted Scope Surveyor, the project manager says...’Don’t call me, I’ll call you’.</td>
</tr>
<tr>
<td>3. As the projects starts to miss deadlines, questions arise about the amount of functionality that is actually being delivered and the Board starts asking difficult questions. The Scope Surveyor is then called in to clean up the mess.</td>
</tr>
</tbody>
</table>

Although there has been no formal analysis of this pattern of behaviour of project managers, one can speculate on some possible reasons as follows:

- Project managers don’t know how to segment their work to enable the scope surveyor to fit within the team
- They feel threatened and somehow impotent in that someone is doing the job they should do
- They know the old ‘fixed price’ scenario well and are comfortable operating with it
- They react to a pushback from vendors that would much prefer the old fixed price as they also are comfortable with it and it also gives them the flexibility to drive 20 – 80% scope creep in ‘authorised changes’. This method eliminates that.

It seems that thirty years of entrenched habits within what have traditionally been the world’s leading software engineering countries mean that the adoption of new ideas and practices is slow --- very slow. This contrasts with the rapidly evolving new economies that don’t have the same ‘baggage’ and are looking to adopt the current best practice at entry level. A good example of this is South Korea where the government is proactive in driving the best practices that the new software engineering knowledge base has enabled.

**About the Author**

Terry Wright is currently a strategist with the Department of Premier and Cabinet of the government of the state of Victoria in Australia. He is the developer of the southernSCOPE methodology and, since 1990, has been a key driver in the creation and exploitation of the current body-of-knowledge of software engineering projects so that the world can better understand and improve the management of software engineering projects. In 1995 he was the founder and, until 2005 the President, of the International Software Benchmarking Standards Group (ISBSG), the organization that has established a repository of over 4,000 projects from more than forty countries. Over the past decade Terry has been at the forefront of innovation in the way that governments use information and communication technologies to transform the way government operates and delivers services.

**Author Contact Information**

Terry Wright, Strategy and Futures
Office of the Chief Information Officer
Department of Premier and Cabinet
3 Treasury Place, East Melbourne, 3002
Office: 96511182 Mobile: 0412 262 890
Email: Terry.Wright@dpc.vic.gov.au

The southernSCOPE methodology was developed by the Government of Victoria. The methodology and supporting documentation is freely available at http://www.egov.vic.gov.au/index.php?env=-innews/detail.tpl:m1816-1-1-7:l0-0-1-n832-0-0
Sizing and Developing Software Using a Model Driven Architecture (MDA) Approach

By Juan Carlos Molina, CARE Technologies

Introduction

The idea of generating code from models has been implemented in a variety of ways with relative success. Most commercially available tools are UML-based, but the amount of code automatically produced from UML diagrams by those tools is relatively small compared to that of a fully functional application. Consequently, with most UML-based tools, hand-coding is still the most significant phase in the process of applications development and is therefore one of the areas in which most of the effort is put by development teams.

Project sizing and cost estimation are, along with the implementation itself, perhaps the two most crucial issues that a project manager has to face. Project managers have to estimate costs and allocate resources to the development of a system before it is built, so they tend to rely on their experience in having built similar systems in the past, and some even use rules of thumb (e.g. estimate the size of the system to be built based on the number of database tables it will have to deal with).

This article is about an MDA-based approach that provides developers with a technology that not only lets them define and, unlike most tools, automatically produce fully functional applications but that also automatically sizes (using the International Function Point Users Group (IFPUG) functional sizing methodology) the systems to be developed even before the first line of code is generated. The resulting functional size is then used for calculating the cost of code generation. Thus, this is actually a testimony of how one software service provider has developed its business model around the “cost per delivered functional unit” construct.

MDA

The goal of the Model Driven Architecture (MDA), promoted by the Object Management Group (OMG), is to clearly separate the logic of applications from the software platform in which said applications are to be implemented. This separation allows reducing the impact that the evolution of implementation technologies has in the development process, thus allowing the same specification of a system to be implemented in different software platforms.

MDA turns models into first-class citizens of the applications development process, and renders source code less relevant. Models become the most valuable asset because from them, by means of a set of automated transformations, the source code of the applications can be obtained.

MDA does not prescribe that transformations be automated, but it is obviously a desirable feature. But more important than the transformations themselves (and the eventual automation of them) are the languages used to describe the models and transformations. The less precise these languages are, the less productive the transformations are (less code will be produced from models created with these languages). The more concrete these languages are, the more coupled to software platforms (and implementation details) models are.

The choice of the words “precise” and “concrete” here is intentional. They must not be seen as synonyms because they mean different things. We can describe something with a lot of precision but that does not mean we have to be concrete. A modeling language suitable for MDA must be abstract and precise: abstract enough so that it allows developers to describe a system regardless of implementation details yet precise enough so that it allows developers to fully describe all the functionality of said system. In the next section we will discuss a technology based on a methodology which uses a specification language with such features.

Implementing the MDA Approach Using OlivaNova

OlivaNova the Programming Machine (hereafter called OlivaNova) is a technology, developed and maintained by CARE Technologies (www.care-t.com) and sold in the USA by Integranova USA, that enables model-driven software development in a hosted web-based environment. Figure 1 presents its main components.

The process begins when the analyst gathers requirements (1) from the stakeholder(s) and uses them to create, with OlivaNova Modeler, the specification (2) of the application to build. This specification is validated by the Modeler, to ensure it is correct, complete and non-ambiguous. Any error found in the spec by the Modeler is reported to the Analyst, who must fix it. The analyst then (3) chooses a set of options (e.g.: target platform– EJB or .Net, target database – Oracle, SQLServer ... etc.) to transform the validated spec into a full application. This is packaged by the Transformation Services Client (4) into a transformation request which is sent to a Transformation Services web service that selects the appropriate Model Compiler(s) (5) to perform the transformation (6) of the validated spec into the full source code of the application (7) which is packaged in zip files and sent back (8) to the requestor either directly or as a URL from which said files can be downloaded.

The Modeler relies on a formal, object-oriented specification language called OASIS, the complexity of which is hidden by means of a subset of the UML graphical notation (e.g. class diagrams, state machines) coupled with an intuitive yet powerful textual syntax to define, among others, constraints, precond... continues on page 30
tions, triggers and actions.

Users of the Modeler can fully define the functionality of a system from four complementary viewpoints (or models):

- **Object Model**, to specify structural properties of the system to be developed in terms of classes, attributes and relationships, as well as the services or operations to create, delete and modify the state of the system
- **Dynamic Model**, to state the dynamic properties of a system in terms of the valid sequence of services that can be invoked on objects of a class and also in terms of the interactions among objects of the system
- **Functional Model**, to define how the state of objects of the system is modified upon the invocation of services (operations) on them
- **Presentation Model**, to specify in an abstract way how users of the system will interact with it

### Counting Function Points on Models, not on Code

While some of the functional sizing methods apply to code, and some are “guesstimates” using some reference value (such as the number of tables in a database, or the number of classes in a UML model), OlivaNova can actually get the full function point count by doing the count on the model, not on the code. Function Point (FP) is the name of the functional size unit in the IFPUG sizing methodology.

Since OlivaNova produces full applications, there is no actual need for manual coding after the model compilation to implement the functionality. Therefore, because the full functionality of a system is automatically obtained from its specification — created by the developer and validated by the Modeler, as set forth in Figure 1 — an automated (repeatable) counting process can similarly be applied to the system specification to obtain its size in FPs. Although the process is quite complex, in essence it relies on the fact that it can map every construct in a system specification to the different elements that have to be taken into account to determine the number of data and transactional functions to which the FP counting rules will be applied to derive the total FP count for the system.

### Using “Cost per Delivered Function Point” as the Business Model

The cost to the developer to use the OlivaNova services depends on the functional size of the system to be developed. Systems are sized in function points and the billing depends on the number of function points of the application delivered by the model compilation services. The good news is that the developer does not have to actually get the system built by the Model Compilers to know how much it will cost: the functional size can be determined prior to using the services.

The following scenario exemplifies the implementation of this business model from the developers perspective (names of the individuals and company are fictitious).

ACME Corp. develops Java applications with a JSP GUI on an EJB layer which accesses a MySQL database. They bought from Integranova two seats of the Modeler for John Doe and Jack Smith and contracted the use of the JSP and EJB Model Compilers. They were provided with a serial number and a key to register their copies of the Modeler. John and Jack were assigned two passwords to access the model compilation services through the Transformation Services Client, which is packaged with the Modeler.

Jack and John are soon assigned to a project for the development of a sales application. After a series of interviews with the customer, using the Modeler, within a week they had created the first version of the specification of the application, which covers

continues on page 31
pretty much all the requested functionality. Once the spec was validated by the Modeler, they use the Transformation Services Client to get the FP count for their specification (1200 FPs) so they know beforehand how much they will be charged to transform that spec into the full source code of the application. They decide to go forward with generating the code. Jack logs on to the Transformation Services Client – using “ACME” to identify his company, ‘jsmith@acme.com’ as his personal identifier (which is also the reply address) and his password – to send a request to transform their specification into JSP source code, and into Java source code on the EJB platform, using MySql as the database. Within minutes, Jack gets an e-mail with a URL to which he directs his browser. A logon page is presented and Jack uses “ACME” and his password to access the download page from which he gets a zip file containing the JSP code and another zip file containing the EJB code. They compile the source and deploy the application, which is delivered to the customer for rapid feedback.

A couple of days later, the customer calls in. He is happy with the application, though he forgot to mention they are missing a service to calculate the expenses of salesmen. Jack and John then create a new version of their spec which includes this service and use the Model Transformation Services to get the new version of the source code for the application. They are not charged again for the functionality that was previously in the spec, only for the additional 6 FPs that resulted from the addition of the new service to the spec, which now totals 1206 FPs for the two versions of the system. Jack and John may need to go back to the model again, whenever the customer wants to refine or add new functionality to the system, but they know that, after each iteration, their will be charged only for the additions of modifications they make, but not for what they already have built.

Jack and John use a powerful MDA technology that enables them to concentrate on the problem at hand, not on how it will be implemented (the OlivaNova services take care of that) and they increase their productivity. Their boss is happy to use a technology that incorporates a repeatable and sound way of sizing their developments and, better still, he did not have to buy a set of tools that might end up stored in some shelf: he only pays whenever his developers use this technology. Integranova, on the other hand, leverages FP sizing as a reliable way to know how much their services are being used and, hence, how much they have to bill their customers.

**Conclusions**

The benefits that OlivaNova provides to development teams are of two kinds: on the one hand, developers increase their productivity by leveraging an MDA approach to obtain 100% functional applications from models in an automated way. The developers focus on the model, not on code. This results in greater productivity for teams and a greater likelihood of meeting schedule demands. On the other hand, project managers can get the size of the systems to be developed at the very early stages of the project: once they have the initial version of the system specification, before a single line of code is generated or even hand-written. Thus, they know up front what the cost will be for the specified functionality.

**About the Author**

Juan Carlos Molina has a BS in Computer Science from the Technical University of Valencia (Spain) and is currently conducting his Ph.D. in the field of Conceptual Modeling and Model-Based Code Generation. Having worked as applications developer for mainframe and information systems, in 1998 he joined the OO-Method research group at the University where he developed a model compiler which transformed conceptual models into applications in Visual Basic. In late 1999, he was hired by CARE Technologies, where he now coordinates the Research and Development division that develops the technology to automate the transformation of models into full applications in different software platforms and architectures.

**Author Contact Information**

Juan Carlos Molina
Research & Development Manager
CARE Technologies, S.A.
Pol. Ind. Juyarco
Pda. Madrigueres, 44
03700 Denia – SPAIN
Tel: +34.96.643.55.55
Email: jcmolina@care-t.com
Visit us [www.care-t.com](http://www.care-t.com)

**USA Representative for CARE Technologies and Integranova**

Farooque Quarashi, Executive Vice President
2001 Gateway Place Suite 750W
San Jose, CA 95110
Tel:+1 408 687 4200 x 101
Direct +1 408 687-4601
Cell:+1 415 341 3433
Email: fquarashi@integranova.com
Visit us [www.integranova.com](http://www.integranova.com)

“Information superiority is fundamental to the transformation of the operation abilities of the joint force. The Joint Force of 2020 will use superior information and knowledge to achieve decision superiority, to support advanced command and control capabilities, and to reach the full potential of dominant maneuver, precision, engagement, full dimension protection, and focused logistics. The breadth and pace of this evolution demands flexibility and a readiness to innovate.”

—Joint Vision 2020

Today’s technology is compelling all sources to provide information electronically. The TEMS system (Total Electronic Migration System), launched in 2004, is now ready for use in gaining instant, electronic access to the Scientific and Technical Information (STI).

TEMS represents a long-term approach to providing access to electronic documents. Each of the nine IACs sponsored by Defense Technical Information Center (DTIC) generates and maintains a vast repository of information related to its area of expertise. TEMS allows the IACs to store, search, retrieve, and use STI in a convenient, centrally located database. It stores information in a wide range of formats, including text, text and images, sound, and multimedia. TEMS users can then perform simple or complex queries of the entire IAC knowledge base, using any Web browser, running any operating system. For full-text searching, TEMS provides a simple search functionality similar to that found on many Internet search pages. For a more complex, algorithmically based search, TEMS includes a function modeled on a commercial search engine.

Using TEMS

Currently (as of July 2006), TEMS has about 39,500 full documents stored, and over 677,000 document citations for reference or further request. This represents more than double the number of document citations in the last year. The number of documents stored will increase soon, due to the conducting of new document scanning and importing to TEMS.

To register to use TEMS first go to www.dtic.mil and click Registration at the bottom of the page. This brings you to a page with instructions and information about the DTIC registration process. Once registered, you can search for documents and citations using a wide range of TEMS search functions and capabilities.

TEMS provides the tools to perform the following:

- Search all DTIC and individual IAC holdings using full text and taxonomy search tools
- Choose the type of search mode, and:
- Utilize Boolean searches to locate information
- Perform a taxonomy search
- Perform a pattern search for a general subject area
- Perform a concept search on a single term
- Search using wild cards and special operators
- Perform an advanced search on multiple subject areas
- View the total number of results returned by a search
- View hits appearing in metadata from a search as bold, colored text
- Search full metadata and document text
- Display metadata with page of document
- Choose a result-presentation format
- Select document metadata, a single-page Portable Document Format (PDF), or an entire PDF document for review
- Create, edit, and export bibliographies in various formats
- Save queries and bibliographies

The overall operation of TEMS encompasses individual IAC systems (so-called Mini-TEMS) and a centralized CTEMS. To populate TEMS, individual IACs scan their paper documents into electronic format, and then enter metadata related to these documents into their local database (Mini-TEMS). The IACs’ own Information Technology (IT) systems interface with the CTEMS, which acts as an archive for data repositories at the Mini-TEMS. Authorized users can access the metadata and documents stored in CTEMS via the Internet. These users will have different levels of access: some will be able to see actual scanned documents; others will be able to see only the metadata about the documents and will have to use other channels to request copies of these restricted access papers.

TEMS search capability

TEMS allows users access to a sophisticated search engine. With it, users can perform a concept, pattern, or Boolean search over document collections from various sources. Users can save search parameters and results for future reference and, if they wish, even edit and re-execute the query later. What makes this search engine unique is the concept search type. The concept search analyzes query terms as units of meaning. When a query

continues on page 33
is entered, the engine searches not only for exact word matches, but also for related words or concepts. (This is called "word expansion.") This is possible due to the search engine’s built-in "semantic network," comprising approximately 285,000 word meanings and over 2.5 million expansion links between words, compiled from published electronic dictionaries and other lexical sources. The engine can also analyze query terms as a pattern, which tolerates spelling differences in either the body of the text or the queries. This is particularly useful in environments where documents have scanning and Optical Character Reader (OCR) errors.

Because of these features, users do not have to build and maintain complex knowledge bases of their own to establish relationships between topics, nor do they have to formulate complex queries to find information that may be worded differently in the searched documents. In addition to its unprecedented accuracy, the search engine is extremely easy to use—a user may enter queries in plain English without any special operators, complex nesting of statements, or rigid syntax.

TEMS is updated nightly to reflect new IAC holdings; hence, IAC users have access to the newest, most current information regarding emerging technologies. The importance of immediate access to IAC holdings for registered users cannot be overstated; IAC users can now immediately access reports needed for research. Instead of waiting to receive a document via mail to determine its relevance, users can now determine that instantly, and utilize the information straightaway.

TEMS will improve the productivity of researchers, engineers, and program managers in the Defense community by collecting, analyzing, synthesizing, and disseminating STI worldwide and in much shorter times than in the past. Most importantly, TEMS now provides support to the DoD community by enabling fast, real-time access to IAC document libraries by users who need this unique information to fulfill their missions.

Contact Information
TEMS
URL: http://iac.dtic.mil/resources_tems.html
Phone: 703.767.9120
E-mail: tems@dtic.mil
Are you getting All you can from your software investment?

The DACS ROI Dashboard

Impact on Quality (% defect reduction)

75th Percentile: 90%
Oklahoma City Air Logistics Center, Test Program Set and Industrial Automation line observed the defect density decrease 91%, from 3.3 to 0.3 defects per KSOLOC, while achieving CMM Level 4.

Technologies Covered:
- SEI CMM/CMMI
- SEI Team Software Process (TSP)
- SEI Personal Software Process (PSP)
- Inspections
- Reuse
- Cleanroom
And Many More!

Graphs Showing Impact Of Software Technologies On:
- ROI
- Productivity
- Quality

Summarizes Facts From Open Literature

Access the DACS ROI Dashboard!
http://www.thedacs.com/databases/roi/
The first 50 people to send in a completed survey will receive a FREE DoD/IT Acronym CD from the DACS.

This valuable CD-ROM contains over 9,000 Department of Defense and Information Technology acronyms. There are hundreds of acronym lists available but none are as well done as this CD AND specifically targeted towards DoD and Information Technology. This unique-shaped CD-ROM plays in your computer’s regular, hub-mounted, CD drive. You’ll use this great resource over and over again. It’s FREE, just for filling out our brief survey on the next page!

Data & Analysis Center for Software (DACS)

http://iac.dtic.mil/dacs/

Data & Analysis Center for Software (DACS)
Software Tech News Subscriber Survey


2. When did you receive the newsletter? (month/year)

3. How satisfied were you with the CONTENT of the newsletter? (Article Quality)
   - Very Satisfied
   - Satisfied
   - Neither Satisfied nor Dissatisfied
   - Dissatisfied
   - Very Dissatisfied

4. How satisfied were you with the APPEARANCE of the newsletter?
   - Very Satisfied
   - Satisfied
   - Neither Satisfied nor Dissatisfied
   - Dissatisfied
   - Very Dissatisfied

5. How satisfied were you with the OVERALL QUALITY of the newsletter?
   - Very Satisfied
   - Satisfied
   - Neither Satisfied nor Dissatisfied
   - Dissatisfied
   - Very Dissatisfied

6. How satisfied were you with the ACCURACY of the address on the newsletter?
   - Very Satisfied
   - Satisfied
   - Neither Satisfied nor Dissatisfied
   - Dissatisfied
   - Very Dissatisfied

7. Approximately how much of the newsletter do you read?
   - The entire issue
   - Most of the content
   - About half the content
   - Briefly Skimmed
   - Didn’t Read

8. Would you read this newsletter in an E-mail newsletter format?
   - Definitely
   - Probably
   - Not Sure
   - Probably Not
   - Definitely Not

9. How did you request the product or service?
   - Phone Call
   - E-mail
   - DACS Website
   - Subscription Form
   - Other __________________________

10. Would you recommend the Software Tech News to a colleague?
    - Definitely
    - Probably
    - Not Sure
    - Probably Not
    - Definitely Not

11. What topics would you like to see this newsletter devoted to?

    Comments (Optional)

    Contact Information (Optional*)

    Name: Position/Title: 
    Organization: Office Symbol: 
    Address: 
    City: State: Zip Code: 
    Country: E-mail: 
    Telephone: Fax: 
    Functional Role: 
    Organization Type: Air Force Army Navy Other DoD __________________________
    Commercial Non-Profit Non-US US Government FFR&D Other ________________

*Note: You must give us your address to receive the CD.
About the Software Tech News

STN Editorial Board

Philip King
Editor
ITT Industries, DACS

Paul Engelhart
DACS COTR
Air Force Research Lab (IFED)

Morton A. Hirschberg
Editorial Board Chairman
Army Research Lab (retired)

Ellen Walker
ITT Industries, DACS

Thomas McGibbon
DACS Director
ITT Industries, DACS

Article Reproduction

Images and information presented in these articles may be reproduced as long as the following message is noted:

“This article was originally printed in the Software Tech News, Vol. 9, No. 3. Requests for copies of the referenced newsletter may be submitted to the following address:

Philip King, Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil


In addition to this print message, we ask that you send us three copies of any document that references any article appearing in the Software Tech News.

Cover Design by Joseph Barbaccia, ITT Industries

DACS
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: dacs@dtic.mil
URL: http://iac.dtic.mil/dacs/

About This Publication:

The Software Tech News is published quarterly by the Data & Analysis Center for Software (DACS). The DACS is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC). The DACS is technically managed by Air Force Research Laboratory, Rome, NY and operated by ITT Industries, Advanced Engineering and Sciences Division.

To Subscribe to this Publication Contact:

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil
Web: www.dacs.dtic.mil

Distribution Statement:
Unclassified and Unlimited
In This Issue

Tech Views .................................................... 1

Sizing and Estimating for Real-time Software – the COSMIC-FFP Method ...5

COSMIC-FFP – A Method for Sizing All the Software Not Just What the User Sees................................. 12

Estimation Models for Software Maintenance Based on Functional Size ................. 18

With or Without a Navigator? - It’s Your Call ........................................... 26

Sizing and Developing Software Using a Model Driven Architecture Approach ........... 29

TEMS Providing Real-Time Access to Scientific and Technical Information .......... 32

Advertisement

The DoD Software Tech News is now accepting advertisements for future newsletters. In addition to being seen by the thousands of people who subscribe to the DoD Software Tech News in paper copy, the Tech News will also be placed on the Data & Analysis Center for Software’s website (http://iac.dtic.mil/dacs/), exposing your product, organization, or service to hundreds of thousands of additional eyes.

Interested in learning more? For rates, layout information, and requirements contact:

Philip King, STN Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: (800) 214-7921
Fax: (315) 334-4964
E-mail: news-editor@dacs.dtic.mil