

## Estimating for software tasks – the SPECTRE approach

### Legal and Copyright Notices

This document and the software to which it relates are copyright David M Victor. All rights reserved.

Spectre, Construction Points and Construction Point Evaluation are trademarks of Plexsoft Ltd.

Microsoft and Windows are trademarks of Microsoft Corporation. Any other trademarks used are acknowledged as the property of the owning persons or organizations.

### Disclaimer

Errors & Omissions Excepted

Due to a policy of continuous improvement, the software product described by this document is subject to change without prior notice.

### ***Abstract***

*Many and varied are the tools and disciplines that have been developed for Information Technology project management. Indeed, one might be forgiven for thinking that we've reached the point when all the secrets of successful project realisation have been unearthed, and yet "half of all development staff are working on projects that will eventually fail" (Project Manager Today).*

*Furthermore, the same source tells us that "The high rate of project failure...has remained fixed at about 75% for the last decade."*

*In April 2005 we learnt that the Scope intelligence-sharing project was three years late, had encountered problems costing several hundred million pounds and would "now deliver less capability than previously planned and will cost at least 50% more than originally estimated".*

*So, if the available tools for IT project management are anywhere as good as their purveyors claim, why do projects continue to flounder and sink, between the Scylla of unfulfilled promise and the Charybdis of budget meltdown, leaving a flotsam of acronyms in their wake? The inescapable conclusion has to be that something is missing.*

*It is the author's belief that the lack of a reliable **task estimation method** is a major contributor to project failure.*

*His contention is that estimating for single software tasks, **the building-blocks of projects**, generally amounts to little more than systematised guesswork. In the absence of a formal method estimators are inevitably swayed by what they believe management wants to hear and may lack the confidence or experience to defend their views, often in the face of an unrealistic schedule.*

***“If there is one management danger zone to mark above all others, it is software estimation”** (Robert Glass, *Building Software Quality*).*

*The author argues that single software development tasks have their own special estimation requirements, which are rarely appreciated and very seldom satisfied. He identifies these requirements and presents a comprehensive solution.*

## 1. What is a task?

In this article 'task' is defined as a **unit of software development assigned to an individual** encompassing the design, build and test of a program-code solution to a specified problem. The realisation of a task is a small number of unit-tested computer programs, and very often just a single program.

## 2. Task estimation factors

Task development time is conditioned by functional content, complexity, knowledge, experience, testing considerations and the programming language to be employed. Operating system, performance and hardware constraints may also apply.

This article describes a method that takes account of all the above factors and is embodied in a tool called SPECTRE, an acronym for **SPECified Task Realisation Estimator**. The input to SPECTRE is information from the task specification; the output consists of an estimate for development effort in days.

## 3. Construction Points

Construction Point Evaluation™ (CPE) is an original task-based metric developed for the SPECTRE Task Estimator tool (as a matter of interest the prototype version of SPECTRE actually used the time-honoured Source Lines Of Code (SLOC) metric, which worked well enough for line-based languages such as Assembler and Fortran, but was later found wanting - to quote Bill Gates: *"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."* )

There are similarities between CPE and Function Point Analysis (FPA) in the way both metrics are applied, but their respective target entities and their chronological points of application in the development process are very different - and, most importantly, their fundamental objectives. These differences are made clear by the following citations from recognised authorities (the last two may seem mutually contradictory, but Herron and Garmus are discussing a proposed deliverable while Capers Jones is concerned with measuring that which has been delivered):

FPA is used *"as a sizing measure to readily determine rate of delivery, cost per unit, and quality rates...the size measure is **based on the user's view of the functions required and provided by the application. This is done **without regard** to the technology, design, tools or language used"*** (Linda Smith, Predicate Logic, Inc.).

*"The value to be gained from utilizing a functional sizing technique, such as Function Points, is primarily in the capability to **estimate a project early in the development process**"* (Herron & Garmus, The David Consulting Group).

*"Function Points give software engineering researchers a way of sizing software through the analysis of the implemented functionality of a system **from the user's point of view**"* (Capers Jones, chairman of Software Productivity Research).

What CPE and FPA have in common is that a points score is assigned to selected entities in a specification and that this score is adjusted in the light of perceived complexity. The essential difference is that FPA is a project-oriented metric while CPE is a task-oriented one.

Thus, the specification whose Function Point Count is measured will be that of a project (or at the very least a discrete business application); whereas CPE encompasses **a single development task**. FPA is applicable from project inception onward but has no explicit relationship to individual tasks; CPE is relates to **individual tasks only** and its focus is strictly confined to task-specific factors.

For information, a brief description of FPA is given in the Appendix.

## 4. Task Estimation – the SPECTRE method

Task estimation consists of three phases:

1. Construction Evaluation
2. Development Effort Calculation
3. Results Display

Within SPECTRE each phase, and in fact every item, is supported by comprehensive Help Text, some examples of which are used in the following sections.

### 4.1 Phase 1 – Construction Evaluation

Construction Evaluation recognises two classes of function: Input/Output and Processing Logic.

#### 4.1.1 Input/Output

Input/output operations are listed in Table 1.

*Table 1. Input/Output Operations*

<ul style="list-style-type: none"><li>• read from a file</li><li>• write to a file</li><li>• delete from a file</li><li>• update a file</li><li>• select from a table</li><li>• insert into a table</li></ul>	<ul style="list-style-type: none"><li>• delete from a table</li><li>• update a table</li><li>• interface with another program</li><li>• generate a report</li><li>• generate a display</li></ul>
---	--

Each Input/Output operation in the task carries its own Construction Point cost. These costs are accumulated, with repeated occurrences of a feature shaded down in value to take account of the probability of shared code being used (e.g. for error and exception handling), giving a total for the task.

Figure 1 shows the relative Construction Point cost of Input/Output operations; the higher the position in the graph, the greater the number of Construction Points needed (the preliminary count calculated for the task is later adjusted in the light of the task's complexity).

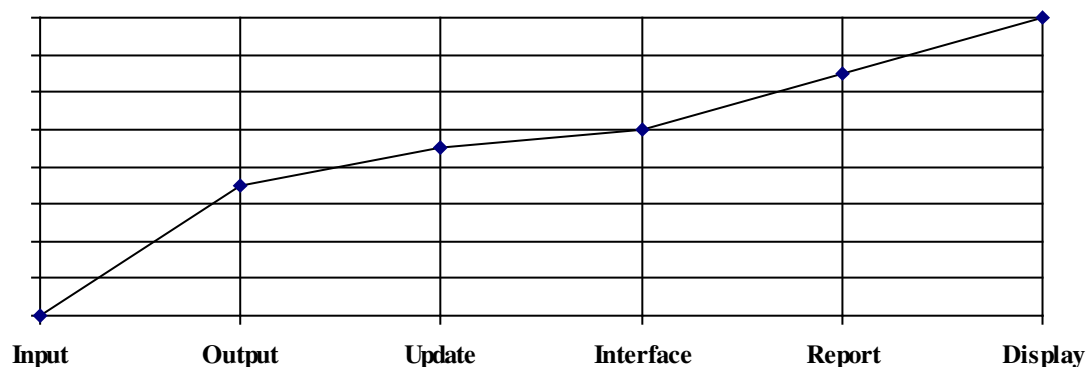


Fig. 1 Construction Point Cost of I/O Operations

### 4.1.2 Processing Logic

Processing logic features are listed in Table 2.

*Table 2. Processing Logic Features*

<ul style="list-style-type: none"><li>• data restructuring</li><li>• conditions</li><li>• linkages with other programs</li><li>• data retrieval</li><li>• calculations</li></ul>	<ul style="list-style-type: none"><li>• sorting</li><li>• data presentation</li><li>• testing considerations</li><li>• performance considerations</li></ul>
--	---

Each applicable feature is given a weighting which represents its degree of complexity (see 4.1.4), from which a Complexity Coefficient is derived for the task.

### 4.1.3 SM

SM (Statement Multiple) is used to convey the coding effort typically required to realise a single Construction Point's-worth in the programming language to be used. The Complexity Coefficient is then applied to arrive at the final SM value.

*The author freely acknowledges the pioneering work of others in the evolution of estimating techniques, in particular the research into the relative power of different programming languages as exemplified by the Programming Languages Table published by Software Productivity Research, Inc. This prompted the realisation that such a relationship should also be capable of derivation for Construction Point assessment, which has proved to be the case. SPECTRE is therefore able to determine the SM required to implement a 'quantum' of Construction in the languages currently in its repertoire. When the requirement arises for SPECTRE to handle a new programming language, it is researched by reference to its ancestry, structure and character, by means of the author's personal contacts within the IT industry, and by investigation of public domain sources such as web sites, the press and available literature. Feedback from SPECTRE users is expected to become more significant as its usage grows.*

The customer will have previously specified all the languages he might wish to use and these will have been conveyed to SPECTRE as part of the registration process. Any one of them can be selected at run time.

#### 4.1.4 Complexity

The degree of complexity is an important characteristic of all aspects of processing logic and therefore requires precise categorisation. Tables 3 and 4 shows extracts from the Help Text which illustrate the classification of complexity as Simple, Moderate or Complex.

Similarly detailed Help Text is provided for all the other processing logic features shown in Table 2 (above). Complexity ratings, as applied internally by SPECTRE, are different for each feature, the range for Testing having the widest spread.

*Table 3. Complexity ratings for Conditions*

CONDITIONS	
<b>Simple:</b>	Straight-through control flow; low proportion of branching logic.
<b>Moderate:</b>	Some non-linear logic consisting mainly of IF/THEN/ELSE or CASE constructs.
<b>Complex:</b>	Heavily conditional; logic affected by timing and/or resource-usage constraints; several logical levels.

*Table 4. Complexity ratings for Calculations*

CALCULATIONS	
<b>Simple:</b>	Mostly addition and subtraction (e.g. spreadsheet type with column and/or row totals); simple multiplication, e.g. compound interest computation.
<b>Moderate:</b>	As above but requiring use of IF/THEN/ELSE or equivalent; straightforward iterative or statistical operations.
<b>Complex:</b>	Computation-heavy; includes recursion, non-linear calculations, fuzzy logic, calculus etc.

## 4.2 Phase 2 - Development Effort Calculation

This phase takes account of the grade and experience of the task assignee. Allowance is made for the possible re-use of existing code and program design. The result is an estimate of the effort in days required from receipt of specification to completion of unit testing (or equivalent acceptance point).

Table 5 shows the parameters that Development Effort Calculation requires.

*Table 5. Parameters for Development Effort Calculation*

<ul style="list-style-type: none"><li>• SM</li><li>• Assignee's Grade</li><li>• Assignee's Experience</li></ul>	<ul style="list-style-type: none"><li>• Assignee's Knowledge</li><li>• Knowledge Required</li><li>• Percentage of Design and Code (that could be adapted from existing sources)</li></ul>
---	---

**SM** has already been estimated (see 4.1.3).

**Grade** is not necessarily a function of job-title; rather, it is an indication of where a neutral observer would place the assignee on a scale ranging from Expert downwards.

**Experience** is weighted by reference to a scale ranging from High downwards. The weightings are grade-specific, e.g. a high-experience senior programmer is rated at slightly less than a low-experience expert. Grade and Experience are combined to give an **Experience Coefficient**.

Help Text guidelines for quantifying Grade and Experience are shown in Table 6.

*Table 6. Guidelines for Grade and Experience*

<b>GRADE</b>	
<b>Junior:</b>	Up to 2 years experience; not expected to function unaided.
<b>Average:</b>	Between 2 and 4 years experience; lacks expertise and likely to require guidance.
<b>Senior:</b>	More than 4 years mostly relevant experience; may lack some detailed technical refinement but likely to be comfortable with the assignment; solutions may sometimes seem unnecessarily complex.
<b>Expert:</b>	Probably at least 6 years relevant experience; has a strong degree of technical competence and is highly regarded; depending on breadth of experience may not always find the 'best' approach but can recognise and appreciate it; produces solutions that look simpler than the problems they address; generates design and code that demonstrate a high degree of maintainability; a significant contributor to installation standards.
<b>EXPERIENCE</b>	
<b>Low:</b>	Recently arrived at this grade and likely to remain there for at least another year.
<b>Average:</b>	Well-established at this level but not yet ready to advance.
<b>High:</b>	Functions very well at this level.

Figure 2 illustrates the relative weightings given to Grade and Experience.

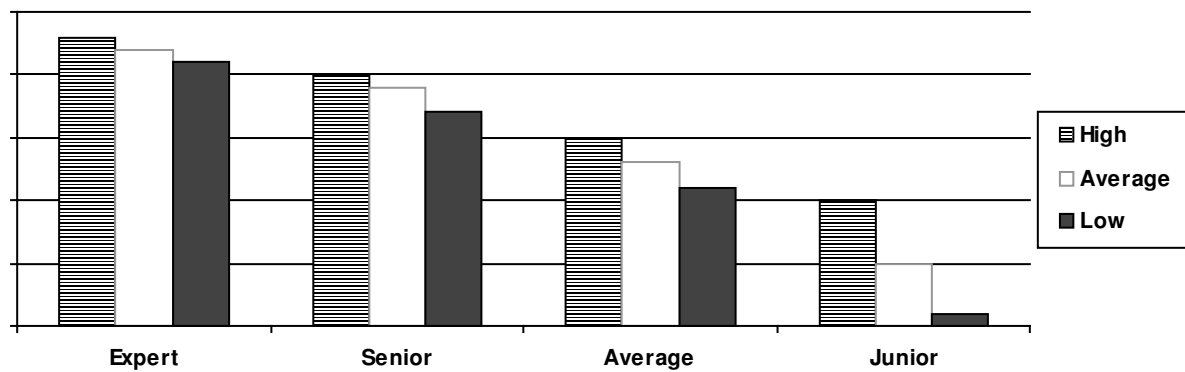


Fig. 2. Relative ability by grade and experience

**Assignee's Knowledge** is an assessment of how much the assignee knows; not only of the task in question but also of relevant related subjects including familiarity with the programming language to be used. It is broken down into a number of descriptors ranging from detailed knowledge of the task and related subjects, to no knowledge of the task and little or no general knowledge of related subjects. The subjects covered are:

- The current task
- Installation Standards
- Operating System
- Operating Hardware

**Knowledge Required** is an assessment of the relevance of the Assignee's Knowledge. This weighting is applied to the Assignee's Knowledge to give a **Knowledge Coefficient**.

Figure 3 illustrates the relative weightings given to the shortfall between Assignee's Knowledge and Knowledge Required (the higher the position in the table, the higher the Knowledge Coefficient):

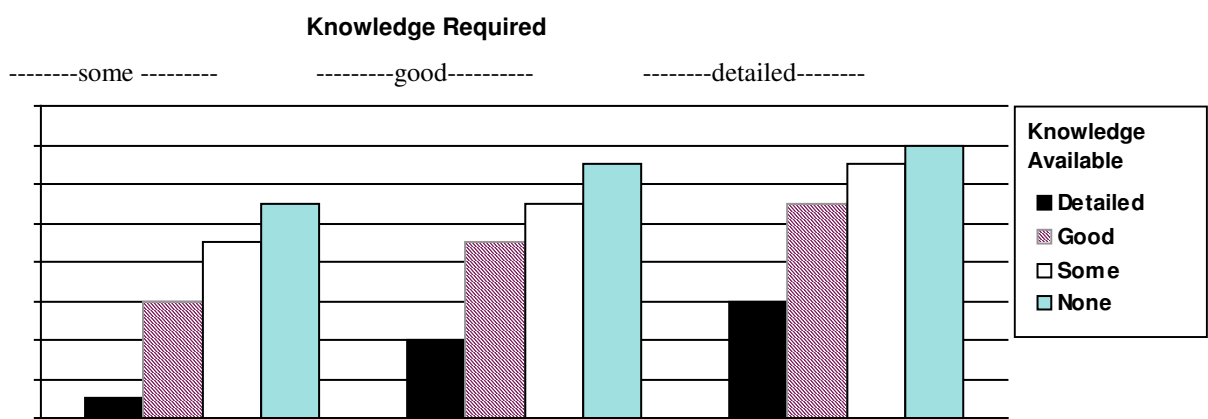


Fig. 3. Relative Knowledge Shortfall

**Percentage of Design and Code** reflects the fact that many applications contain tried-and-tested pieces of design and code (for exception handling, processing transactions against a master-file, etc.). The percentages of design and code that could be adapted from other

sources permit the economies of re-use to be taken into account (with an allowance for customisation).

The SM estimate is divided by a base SM per day value to give an initial number of days to which the Experience and Knowledge Coefficients are applied to give the estimated development effort in days for this assignee/task combination.

Finally, Quality Control and Acceptance Testing Overheads are added. These are contingency allowances, in percentage terms, for possible additional effort involved in Quality Control activities (such as walk-through, peer review) and System and/or Acceptance Testing on the part of the task assignee.

## **4.3 Phase 2 – Results Display**

### **4.3.1 Results - Section 1**

The display is headed by four lines identifying:

- Project/System if provided, or ‘unspecified’
- Developer name if provided, or ‘unspecified’
- Task ID as provided
- Programming language selected for this task

### **4.3.2 Results - Section 2**

This section shows the estimated development effort in days (industry figures indicate that there are approximately 130 working hours in a month in real terms; the value displayed will have been adjusted accordingly).

Figures for estimated effort plus and minus 15% are also displayed to provide a reasonable tolerance.

If the estimate is one that has been retrieved from the database (see Section 5. Other Facilities), the user is given the opportunity, via the Previous Estimate button, to view the database version so that any differences can be noted. This facility is provided when:

- The Save icon is pressed and the database version has not been updated before, or
- Immediately, if the database version has been previously updated

### **4.3.3 Results - Section 3**

This section shows a number of additional items which may be of value or interest.

- Time saving through code/design adaptation
- Complexity rating
- Experience rating
- Knowledge rating
- Construction Point score

SPECTRE evaluates all the above in terms of size, range or scope as appropriate, and appends comments - in particular, any factor considered to be significantly outside ‘normal’ expectation is highlighted. Help Text clarifies all the items shown.

### **4.3.4 Results - Section 4**

This final section identifies the creator of the estimate and, if applicable, the author of the most recent amendment. Both are date/time-stamped.

## 5. Other Facilities

- **User Guidelines** are provided which explain how to tailor usage of SPECTRE to the user's own methods and practices.
- A **Formatted Print** option is provided for the results display.
- SPECTRE's **Database** allows the entire estimate to be saved for reference and also for amendment and re-calculation as needed. The database allows estimates to be grouped, if required, e.g. by project/system as the senior key (junior keys being developer/task/language). The database's display function can show the **total effort** estimated for all the tasks (a) in any group or (b) assigned to an individual developer. Furthermore, for any group it can list all the individual tasks assigned ordered by **developer/task/language**, and for any developer it can list all his tasks ordered by **group/task/language**. These lists have a print facility and can also be output to a Comma-Separated Variables file for importation into a spreadsheet.

## 6. Conclusion

The SPECTRE approach to task estimation is based on accumulated observations over many years of experience in software construction. The result is a rational and realistic approach to task development, sensitive to all the parameters that the author's experience has shown contribute to success or failure, including functional content, component complexity, and the knowledge and experience of the task assignee. It also takes account of opportunities for the re-use and adaptation of design and code.

SPECTRE is at its most effective when applied to a task specification that is complete and confirmed. Nevertheless, it can be used to advantage throughout the whole process of developing the task specification; for example, to give an early feel for notional best, average and worst cases. If a task threatens to exceed reasonable limits of size, complexity or effort, or looks as if it will require a particularly high level of skill, then the earlier these potential exposures are flagged and resolved, the better.

The arguments for using the SPECTRE method can perhaps be best summarised by two favourite quotes, both from Fred Brooks Jr who states memorably that "*adding manpower to a late software project makes it later.*" In answer to his own question: "*How does a project get to be a year late?*" his telling response is, simply, "*one day at a time.*"

## Appendix – Function Point Analysis

A Value Adjustment Factor (VAF) is determined, based on a number of General System Characteristics (GSCs). In FPA Mark II there are 14 of these, as listed in Table 7.

*Table 7. General System Characteristics in Mark II Function Points*

<ul style="list-style-type: none"><li>• Data Communications</li><li>• Distributed Data Processing</li><li>• Application Performance</li><li>• Degree of use of Hardware Platform</li><li>• Transaction Rate</li><li>• On-Line Data Entry</li><li>• End-User Efficiency</li></ul>	<ul style="list-style-type: none"><li>• On-Line Update</li><li>• Application Complexity</li><li>• Reusability</li><li>• Ease of Installation</li><li>• Operational Ease</li><li>• Multiple Sites</li><li>• Ability to Undergo Change</li></ul>
--	--

Each GSC is influence-rated in the range 0 to 5, based on an assessment of five Function Point Components, namely:

- External Inputs
- External Outputs
- External Inquiries
- Internal Logical Files
- External Interface Files

The total of the complexity-rated values of each occurrence of these Components is calculated and then multiplied by the VAF to give Total Adjusted Function Points (details of how the VAF is derived and how complexity rating is performed are outside the scope of this document; the interested reader is referred to the International Function Point Users Group).