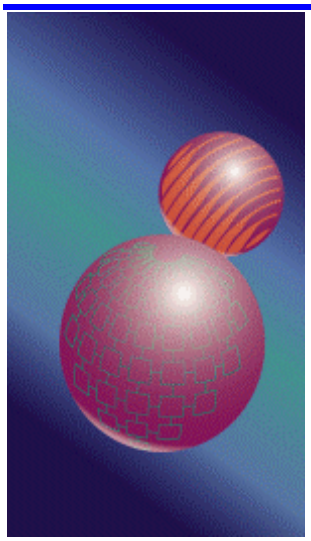

# The Role of Function Points in Software Development Contracts

**Paul Radford and Robyn Lawrie,**
**CHARISMATEK Software Metrics**

info@charismatek.com

## Abstract

*Software development contracts often lead to disputes between the software service supplier and the client commissioning the development. A chief cause of the disputes is the cost of the software development exceeding initial estimates.*

*Software development pricing is based on a 'fee per unit' basis. This paper examines 3 key 'fee per unit' pricing approaches - Time and Materials, Fixed Price for Total Delivery and Fixed Priced per Function Point - outlining the advantages and disadvantages of each to the supplier and the client. The paper also suggests other items to be recorded in the software contract to help ensure a controlled but fair and consistent approach to software development pricing.*

## Introduction

Software development contracts often lead to disputes between the software service supplier and the client commissioning the development. A chief cause of the disputes is the cost of the software development exceeding initial estimates as well as late delivery or non-conformance to what clients perceive as their stated requirements. An appropriate service contract can help to reduce or mitigate such conflict.

Software development pricing is based on a 'fee per unit' basis. The most commonly used 'units' are:

- a **time period**, such as an hour, where the client pays a fixed cost per time period. This is the Time and Materials approach.
- the **total product**, where the client pays a fixed price for the total delivery. This is the Fixed Price for Total Delivery approach.
- a **Function Point**, where the client pays per Function Point delivered. This is the Fixed Price per Function Point approach.

Each different method has its advantages and disadvantages and has situations where it is best used. This paper later discusses each of these methods in more detail and in particular explores the use of pricing based on Function Points.

When estimating price or setting a fixed fee, the supplier will use some assumptions about the solution to formulate the price. The principal project attributes and assumptions about the project impacting the price are:

- Product Size
- Technology - e.g. Platform, Language
- Complexity of the Problem
- Complexity of the Perceived Solution - e.g. User Interface, Performance
- Project Constraints or Goals - Schedule, Cost, Quality
- Project Interruptions and Delays

## Key Project Attributes Impacting Price

### Product Size

The functional size of the software Product to be delivered is a key cost driver. The size may be assessed:

- *formally*, for example, using the Function Point Analysis technique, where an attempt is made to identify and assign Function Points to each function to be delivered or
- *intuitively*, using a micro-estimating technique in which each activity is identified and assigned effort but no quantitative statement of product size is made.

All other factors being equal, the larger the software product, the higher the cost and, in addition, the larger the software product, the higher the $ cost per Function Point.

So, how well product size can be assessed at project outset is a key ingredient for success with any of the pricing methods.

Product size is usually assessed from a Software Requirements Definition. This document provides a statement of the business problems and needs to be addressed by the software. It may also specify the external behaviour of the software product in terms of software functionality as well as other characteristics of the product such as performance, quality attributes, usability and so on.

To assess product size, the functional user requirements in the Software Requirements Definition are analysed in order to extract and / or construct the logical view of a software product which can supply a solution to the business problems.

How precisely this can be done depends both on the nature and the intent of the Software Requirements Definition. Some Software Requirements Definitions include statements which proscribe the required or preferred solutions. Others intentionally include high level statements of the business requirements in order to invite a variety of solutions, e.g.

- *the business will be able to bill electronically*
- *the business rules for all calculations must be able to be specified and maintained by the user*
- *in all decision making functions, the user will be presented with a list of viable options*
- *all components of the distributed system will at all times be operating with the same set of infrastructure data.*

These type of statements are particularly common in requirements for large, technically innovative and / or complex software solutions.

Furthermore, the Software Requirements Definition is a statement of requirements as perceived by the business. Software which may have to be built in order to enable the solution may only be inferred by the Definition because this type of functionality may be outside the business domain or knowledge area of the client. If, for example, the solution needs a *job scheduling system* or a sophisticated *archive / restore sub-system* or *message / data transfer between distributed system components*, and the chosen platform does not support these, then this functionality may have to be sized and built as part of the project, even though it is not specified in the Software Requirements Definition

A good Software Requirements Definition is a necessary base for any contracted software project. Alan Davis[1] , in his book *'Software Requirements - Analysis and Specification'* includes an excellent discussion on the nature of Software Requirement Definitions.

In summary, some Software Requirements Definitions lend themselves well to 'precise' sizing. Others do not and, for these projects, an indication of a size in a size range may be the best that can be done.

For further discussion about product size, use of contingency and management of scope, refer to Lawrie, 1995.

### Technology Choices and Other Factors

In addition to product size, other project attributes will influence the fee. This is by no means an exhaustive list but includes:

- Technology - e.g. Platform, Language
- Complexity of the Problem
- Complexity of the Perceived Solution - e.g. User Interface, Performance
- Project Constraints or Goals - Schedule, Cost, Quality.

Whatever attributes are either true or assumed to be true about the product should be documented in the contract. Should any prove to be untrue or become untrue because of changes to requirements, then there may be a case for contract review.

For example, if a fee schedule is based on the assumption that a 4GL will be used for building the management reporting and a Change Request to the project after the contract signing introduces performance requirements which invalidate the choice of language and necessitate the use of a 3GL, then a review of the fee schedule is obviously indicated.

### Interruptions and Delays

All projects will experience some level of interruptions and delays. What is 'reasonable' is difficult to know. However, 'unreasonable' delays in signing off project documents will impact cost as will interruptions during the project to assess Change Requests.

The contract should attempt to agree how these events will be handled during the project.

# Contract Types

### Time and Materials

In the Time and Materials approach, the client pays a fixed cost per time unit, usually a fee per hour.

In order for the client to make the decision to proceed with the software development project, the supplier is usually asked to provide an estimation of the delivery cost based on the supplier's understanding of the Software Requirements Definition. Increased product functionality emerging as understanding of the Software Requirements Definition increases and the product definition is clarified is assimilated into the project.

The perceived benefit of this approach is that - given that the supplier is professionally honest and employs the most productive and effective development procedures and staff - then the client will receive the most beneficial solution.

The advantages are:

- Changes to requirements are easily absorbed into the project contract
- During the project itself, the lack of pressures associated with pricing constraints means that the project team is free to focus on delivering the software solution. This can be particularly important where the project involves research and development activities, such as the development of algorithms and the use of new technology. These type of activities do not lend themselves well to prediction of associated effort.

The disadvantages are:

- The client is asked to absorb the risk associated with both the product definition and the estimation of delivery cost. If the actual cost of the software product significantly exceeds the estimated cost, the repercussions are many. For example, the cost / benefit justification for the software may be invalidated or the client may not be able to afford completion and end up with nothing.
- There is no cost incentive for the supplier to either deliver in the most cost effective way or to control product size. As a larger outcome of a poor initial cost estimation, the supplier may lose the client's future business.

Where the product definition is well understood, realistic expectations for the project cost should be able to be established.

Where the product definition is not well understood, there is consequent high risk. The product, for example, may be innovative in concept or using new technology. In such circumstances it may be appropriate for the client as the potential beneficiary of the final product to also bear all the risk of development.

Time and Materials is a common contractual arrangement for in-house software development projects. In many such circumstances it is not only the easiest arrangement but highly appropriate since the client and the supplier shares risks as part of the same organisation. However, when all parties are not conversant with the full meaning and allocation of risk and responsibility implied within the contract, this approach can lead to severe acrimony (and be a prime trigger for outsourcing initiatives).

**Fixed Price for Total Delivery**

In the Fixed Price for Total Delivery approach, the supplier is asked to provide a fixed price delivery cost based on the supplier's understanding of the Software Requirements Definition before the project commences.

The Fixed Price per Total Delivery relies on a known product size at project outset when the price is agreed.

The perceived benefit of this approach is that the delivery price is fixed and is known to the client.

The advantages are:

- If product definition is adequate and change minimal, the client can budget with confidence as to total cost and has the ability to make a fully informed decision on purchase from the supplier
- There is incentive for the supplier to control and manage the product size and to deliver in a cost effective way.

The disadvantages are:

- The supplier absorbs the risk arising from the initial estimate
- To ameliorate this risk, the supplier may build contingency into the price which means that the client pays a higher price
- Increased product functionality which may emerge as understanding of the Software Requirements Definition increases and the product definition is clarified becomes a source of contention
- If the supplier has initially underestimated price or 'misunderstood' the required product, the result may be a software product of less than desirable functionality or quality or indeed a supplier out of business
- Changes to requirements are usually the subject of a separate quotation where the supplier may be tempted to load the pricing if initial estimates of cost were low.

  Indeed, it is not uncommon for suppliers to deliberately bid a low fixed price in the sure and certain knowledge that they will be able to significantly overcharge for amendments to requirements uncovered during the development process. This is partially due to the usually poor standard of project definition particular to the software industry.

To reduce disputes arising from increased product functionality emerging with increased understanding of the Definition, some suppliers put into their software contract a ceiling product size. This ceiling product size is derived by assessing product size from the Software Requirements Definition and adding a contingency for 'emerging' functionality.

For example, the initially assessed product size may be 2700 Function Points but the product size actually used for the pricing is 3000 Function Points.

Care must be taken with this ceiling product size approach to indicate in the contract with the client that the intent is to accommodate the functionality as per the Software Requirements Definition, not, as one client interpreted it, any set of *x* Function Points.

**Fixed Price per Function Point**

In the Fixed Price per Function Point approach, the supplier provides a fee schedule for functionality present in the Requirements Definition and for changes to Requirements received from the client during the project.

This approach relies on an assessed product size using the Function Point Analysis technique for assessing Functional Size. In theory, product size does not need to be known at project outset but, in reality, the product is sized or its size at least estimated as early as possible in the project so that the client can evaluate the price against the budget.

The intent of this approach is to build a contract which reduces the risk for both the client and the supplier.

The advantages are:

- The client understands how the fee is derived and how it will vary with changed requirements
- The client can compare pricing against published industry schedules
- The supplier is not disadvantaged when additional functionality emerges as understanding of the stated requirements increases
- The approach can apply across the full development lifecycle. This can short circuit the expensive and tedious tender definition and evaluation process as used by many organisations and almost universally by government.

The disadvantages in this approach are:

- The unit on which the pricing is based is not a 'hard' or 'physical' measurement.

  Function Points are an *indicator* of size based on the 'logical' view of a piece of software. Although Function Points are 'standardized' in that there is a body of rules and guidelines which support the technique, in real life 'expert counters' will not always agree. For their intended purpose of extending a Function Point size using macro-estimating techniques into effort and cost, these variations in interpretation are not significant. However, when there is perceived to be a $Cost associated with each Function Point, these variations can become a source of dispute for which there may be no easy answer.

  In a project, there will be '*cheap*' and '*expensive*' function points when compared with the actual cost of supply. Cost variations per Function Points are to be expected because of the nature of the technique. The reasons for 'extreme' cost variations include e.g. intrinsic complexity of the functions, the constraints and benefits of the chosen technologies, the chosen delivery strategies (are the Function Points reused / bought / built?).

  This disparity in supply costs for some functions can be manipulated by both clients and suppliers.
- The guidelines for the sizing of changes to requirements are not well developed and there are many different interpretations currently in use.
- Function Points are, in general, not well understood by either suppliers or clients and there is usually a need for an external 'independent' assessor to size the project and project changes and resolve disputes.
- The size of the software solution may exceed expectations unless consciously constrained.

*For example, in a project to replace an existing system of about 500 Function Points with one essentially delivering the same functionality but on a new platform, the replacement after re-gathering of requirements was sized at 1200 Function Points, well outside the original budget. The Supplier was asked to review and reduce Functionality.*

- Suppliers may be asked to provide a fee per Function Point in a competitive situation where the required functionality can at best be described as 'indicative'. The supplier's fee is based on a reuse / buy / build strategy which appears to satisfy requirements. Further refinement to the requirements may prove this strategy and its associated cost to be inappropriate.

  *For example, a supplier was able to win business based on a packaged solution which appeared to satisfy requirements. Detailed analysis indicated that more change was required than anticipated by the supplier. In order to ensure original profit forecasts, the supplier convinced the user groups associated with the system to include much more of the pre-existing package functionality (mostly to do with management of presentation, security, etc.) in the "required" functionality category - functionality which was far less extensively detailed in the original indicative outline.*

## Fee Schedules for Fixed Price per Function Point

The format of the fee schedule can take any number of forms. Some examples are included. Note that in these examples, the fees stated are indicative only.

*Example 1*

The following table provides an example of a published schedule of fees used by the US commercial software supplier - RDI Technologies[3].

| Function Point Count | Functional Design Cost per FP | Implementation Cost per FP | Total Cost per FP |
|---|---|---|---|
| 1,501-2,000 | $242 | $725 | $967 |
| 2,001-2,500 | $255 | $764 | $1,019 |
| 2,501-3,000 | $265 | $773 | $1,058 |
| 3,001-3500 | $274 | $820 | $1,094 |
| 3,501-4000 | $284 | $850 | $1,134 |

This table is interpreted as follows:

If the client has a project which is assessed at 2,200 Function Points, the cost per Function Point for the first 2000 Function Points is $967 and the cost for the next 200 is $1,019. In this way, these fees establish in the client's mind that as the product size increases, the fee / FP also increases.

RDI size the product at the end of Functional Design and the fee for the Functional Design is calculated. The functionality to be built becomes the Baseline size for the build and implementation. When the software is completed, it is again sized. The charges at implementation are calculated as:

*Baseline Size * Implementation Cost / FP +*
*(FPs added, changed or deleted during implementation) * Total Cost / FP.*

i.e. Changed functionality attracts not only the Implementation Cost but also the cost of Functional Design.

*Example 2*

Capers Jones[4] suggests using a sliding scale where additional cost is determined by the period of time after contract signing.

| | $ Cost per Function Point |
|---|---|
| Initial 1000 FPs | 500 |
| Features added more than 3 months after contract signing | 600 |
| Features added more than 6 months after contract signing | 700 |
| Features added more than 9 months after contract signing | 900 |
| Features added more than 12 months after contract signing | 1200 |
| Features deleted or delayed | 150 |

*Example 3*

The Department of State Development - Victoria[2] Australia, suggests a single fixed price per Function Point and a table for handling variations expressed as a percentage of the base price:

| | PRE-Acceptance Testing Handover | POST-Acceptance Testing Handover |
|---|---|---|
| Additional FPs | 100% | 100% |
| Changed FPs | 130% | 150% |
| Deleted FPs | 25% | 50% |

*Example 4*

A Victorian State Government project recognized the varying intrinsic complexity, performance and quality requirement (safety critical software) and consequently asked suppliers to respond with varying fee schedules for functionality groups which had been differentiated by complexity/quality/performance characteristics.

## Contracting for Change

As the project progresses, software requirements will in all probability change. Each contract must indicate how change will be charged for.

There are some points about how clients view change which should be taken into account when drawing up the contract:

- Clients understand Change Management - i.e. they understand that changes they request to the functionality will add, change or remove existing functionality and will therefore attract an additional cost.
- Clients do not understand 'internal' growth - i.e. they do not readily understand the situation where the developer, with the benefit of additional project knowledge, 'discovers', for example, that what was thought to be a single function is actually a number of functions, each with a discrete set of processing rules.

- Suppliers and clients can sometimes have difficulty in agreeing when a 'change' is a correction and when it is a change.

  *For example, if the Software Requirements Definition states that 'All user dialogs will support business workflow' and the client requests changes to a dialog to reflect this requirement during Acceptance Testing, is this a correction or a change? Alternately, if the supplier has inadequately analysed the requirement, at what point and to what degree is it the responsibility of the client to ensure the correctness of the description of the requirement?*

  *The business may regard it as the responsibility of the supplier to accurately assess certain existing procedures. On the other hand, the contract might include reference to checkpoints at which the business accepts the manner in which certain tasks are to be performed within the eventual product.*

Definitions of Change must be written into the contract. This is particularly important where function points are used as the basis for charging

## Conclusions

Software development projects, in general, are not straightforward and without risk.

In meeting the business need of providing a constrained but fair and consistent approach where the development risk is shared according to the responsibilities of the client and the supplier, the concept of Fixed Price per Function Point has a number of advantages over alternative contract approaches. The optimum structure of such contracts will be refined from a combination of approaches outlined in this paper but will vary according to the specific type of software development under consideration.

The method also ensures that at least one of the key risk areas - product size - is controlled. However, as with alternative contract types, the key issue of forecasting and meeting optimum schedules requires evaluation and control of a number of other factors. Where practicable, all other constraints must be identified and recorded in the contract.

## References

1. DAVIS, ALAN M.: Software Requirements - Analysis and Specification, Prentice-Hall, 1990.

2. DEPARTMENT OF STATE DEVELOPMENT -VICTORIA: *Draft Acquisition of Customised Software Policy*, September 1996.

3. GARMUS, DAVID and HERRON, DAVID (1996): *Measuring the Software Process*, Prentice-Hall.

4. JONES, CAPERS (1996): Conflict and Litigation Between Software Clients and Developers, Version 2 - July 31, 1996.

5. LAWRIE, ROBYN (1995*): Using Software Metrics to Manage Software Project to Fixed Cost*, Proceedings of 2nd Annual Conference of ACOSM.