COMPUTING SCIENCE

Trade-off Between Cost and Reliability During the Design Phase

R. Burnett and T. Anderson

TECHNICAL REPORT SERIES

No. 534 November, 1995
TECHNICAL REPORT SERIES

No. 534  November, 1995

Trade-off Between Cost and Reliability During the Design Phase

R. Burnett and T. Anderson

Abstract

This paper proposes a method for estimating the development cost of a software module, taking into account the target level of reliability for that module. Our objective is to establish a basis for a model to guide a primary trade-off between cost and reliability during the design phase of the development of a modular software system.

The line of argument developed here is that the operational reliability of a software module can be linked to the effort spent during the testing phase - a higher level of desired reliability will require more testing effort and, consequently, will cost more. A decomposition technique is used to estimate the cost of development, based on an estimate of the number of faults to be found and fixed in order to achieve the required reliability, using data obtained from the requirement specification and historical data. The proposed model is easy to understand and suitable for use by project managers.
Bibliographical details

BURNETT, Robert Carlisle
Trade-off Between Cost and Reliability During the Design Phase
[By] R. Burnett and T. Anderson
(University of Newcastle upon Tyne, Computing Science, Technical Report Series, no. 534)

Added entries
UNIVERSITY OF NEWCASTLE UPON TYNE.
ANDERSON, Thomas

Abstract

This paper proposes a method for estimating the development cost of a software module, taking into account the target level of reliability for that module. Our objective is to establish a basis for a model to guide a primary trade-off between cost and reliability during the design phase of the development of a modular software system.

The line of argument developed here is that the operational reliability of a software module can be linked to the effort spent during the testing phase—a higher level of desired reliability will require more testing effort and, consequently, will cost more. A decomposition technique is used to estimate the cost of development, based on an estimate of the number of faults to be found and fixed in order to achieve the required reliability, using data obtained from the requirement specification and historical data. The proposed model is easy to understand and suitable for use by project managers.

About the author

Robert Burnett is currently a PhD student in the Department of Computing Science at the University of Newcastle upon Tyne, and on leave from Departments of Informatics, CEFET/PR and PUC/PR, Curitiba, Brazil. He is supported by CNPq/Brazil.

T. Anderson is a Professor in the Department of Computing Science at the University of Newcastle upon Tyne.

Suggested keywords

DEVELOPMENT COST SOFTWARE COST MODELLING SOFTWARE METRICS
SOFTWARE TESTING TRADE-OFF COST-RELIABILITY

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.6425 620.0044 658.1552
U.D.C. 681.322.06 519.718 519.866
Trade-off Between Cost and Reliability During the Design Phase

Robert Burnett*  Tom Anderson†
B.C.Burnett@newcastle.ac.uk  Tom.Anderson@newcastle.ac.uk
Department of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne—England NE1 7RU
Fax: +44 91 222 8232

Abstract

This paper proposes a method for estimating the development cost of a software module, taking into account the target level of reliability for that module. Our objective is to establish a basis for a model to guide a primary tradeoff between cost and reliability during the design phase of the development of a modular software system.

The line of argument developed here is that the operational reliability of a software module can be linked to the effort spent during the testing phase—a higher level of desired reliability will require more testing effort and, consequently, will cost more. A decomposition technique is used to estimate the cost of development, based on an estimate of the number of faults to be found and fixed in order to achieve the required reliability, using data obtained from the requirement specification and historical data. The proposed model is easy to understand and suitable for use by project managers.

Key-Words: Software Cost Modelling, Tradeoff Cost-Reliability, Development Cost, Software Metrics, Software Testing

---

*On leave from Departments of Informatics, CEFET/PR and PUC/PR, Curitiba, Parana, Brazil.
†Tom Anderson is Professor and Head of the Department of Computing Science at the University of Newcastle upon Tyne.
1 Introduction

Software project managers recognise the value of techniques which help to estimate the effort (for example: man-months) and cost needed to complete the development of a software system. Then, taking into account the estimation outcome and a specific upper limit for cost, the project cost could be brought under control. Project managers need to be able to estimate how long a software project will take and how much it will cost.

In order to make that estimation it would be helpful to know, before beginning the implementation phase, and taking into account the project profile (such as a required level of reliability), the amount of effort that should be allocated during implementation and testing. Indeed, the cost should be estimated, as accurately as possible, so that the implementation team can be chosen with suitable skills in order to address the constraints of cost and required reliability. To help in this task of software cost estimation, methods have been proposed [1, 2, 3] which yield an estimate of the amount of effort required for development, such as the number of people needed and the development schedule.

Currently available software cost estimation methods do not usually consider the desired level of reliability as a cost driver (see [4]). However, as illustrated, for example, in [5, 6], the cost of a software system is strongly influenced by reliability requirements; that is, a system with high required reliability needs more development effort, and consequently costs more, than if the same system had lower reliability requirements. It can be seen from these papers that the cost of a system increases when the level of reliability required from it increases.

In spite of there being many software reliability models, very few of them provide any guidance on how much effort should be spent (in relation to desired reliability) during the implementation and test phases, before these phases begin. Research has concentrated on “release policies” for software systems, i.e., when to stop testing and deliver the system [6, 7, 8]. Thus these policies do not assist the project manager in advance of implementation: they only provide guidance during testing.

This paper proposes a method for estimating the cost of development of software, given a desired level of reliability for that software. Estimation is carried out during the design phase, that is, before implementation begins, using historical data and the expected number of software faults.

The cost of development \( C_{\text{dev}} \) of a software module is taken in this paper to be specifically the cost spent during the coding and testing phases only. As analysed subsequently, the estimate of the cost of testing is based on the level of reliability \( R \) required for the module, which is linked to the number of faults that are considered to be inserted during the coding phase and removed during the testing phase.

In order to connect cost and reliability a line of argument is developed that achieving a required level of reliability fundamentally depends on the effort devoted to testing, which in turn depends on the number of faults that need to be found and fixed to achieve the required reliability. Each fault requires effort (and therefore cost) to be found and fixed, and hence a relationship between cost and reliability can be established.

As the foremost application of the outcome of this paper we can cite the definition of a tradeoff model between cost and reliability. Using the results of this tradeoff model, a project manager could plan the allocation of resources to the implementation and testing phases so that the estimated total system cost does not exceed the project budget, and the estimated system reliability meets the required target.
To address the points noted above, this work is arranged as follows: In section 2 we define how the cost of development is dealt with. Section 3 proposes a formula that links the number of faults to be fixed to a required level of reliability. Section 4 defines the factors that are involved in the effort of testing and proposes how to estimate the effort of finding and fixing one fault during the testing phase. Section 5 describes the final formula for the estimated cost of development, using the results achieved in the previous sections. Section 6 shows an example of the estimation proposed here, comparing the results with some published data. In section 8 the sensitivity of the established cost formula is analysed, for some parameters utilized; and section 9 provides the conclusion.

2 Cost of Development: Structure

To accomplish the task of estimating the development cost of software during the design phase, we use the most common technique for costing any engineering development project [9], that is, to employ effort estimation. First the number of person-periods (the effort) needed to perform coding and testing (including debugging) is estimated and then a cost is associated with each unit of effort, so that an estimated cost is obtained.

A project manager knowing the outcome of this estimation, namely, effort and cost, and considering a required reliability, could then plan the resource allocation for the coding and testing phases, aiming at avoiding the known problem of cost overrun in these phases.

The estimated cost of development of a software module is defined here as the cost to implement all functions identified in the requirement specification, taking into account a required level of reliability $R$, such that the module can be considered ready for operation. This cost quantifies the effort spent during the coding and testing phases and is represented by

$$C_{dev} = P_{dev} \cdot E_{dev}$$

$$C_{dev} = P_{dev}(E_{cod} + E_{tes})$$  \hspace{1cm} (1)

where

- $C_{dev}$ is the estimated cost of development of a module (taken to be the cost of coding plus testing, taking into account a reliability level $R$), based on the effort of development $E_{dev}$.

- $P_{dev}$ is the cost of development per person-unit time. In this work, it is assumed that the cost spent in either of the coding or testing phases per person-unit time are indistinguishable. Hence, the single cost $P_{dev}$ is used here to estimate the cost of both the coding and the testing phases (of course, if these are to be regarded as different, the $P_{cod}$ and $P_{tes}$ have to be estimated separately).

- $E_{cod}$ is the effort required to implement (i.e., code) the module in, for instance, man-months\(^1\). It is estimated based upon an analysis of previous projects within the organization, and using data from “similar developments”\(^2\) such

---

\(^1\)In, for example, [1, 2, 3] some different procedures to estimated $E_{cod}$ can be seen.

\(^2\)As suggested in [10], “the user defines what this means”. A comprehensive approach on how to define similar software can be seen, for instance, in [9, 11].
that a correlation can be substantiated between the effort expended on those projects taking into account various module sizes and functions.

- $E_{\text{tes}}$ is the effort spent in verification and validation of a module during the testing phase.

Even though the words testing and debugging are often casually used with the same meaning\(^3\), they are, in fact, distinct activities, as is emphasized in [9, 12]. Testing is the activity of finding situations in which the results do not match those expected, that is, a failure of the software has occurred, while debugging is the activity of diagnosing and correcting the fault that produced the failure.

However, $E_{\text{tes}}$ is used here to represent the sum of the effort of testing (considered separate from debugging activities) and the effort of debugging. Thus, the testing and debugging activities are dealt with as a single task. On page 10 some arguments are presented which may clarify why we adopt this approach.

Subsequently the components of equation (1) are expanded, where, it should be highlighted once more, we are interested in finding a way of expressing $C_{\text{dev}}$ in relation to a required level of reliability $R$.

### 3 Underlying Relationship for the Required Reliability

Three quantities of software faults are utilized in our analysis of cost and reliability, as indicated in figure 1, where

\begin{figure}
\centering
\begin{tikzpicture}
\node (N) at (0, 0) {$N$};
\node (F) at (-1, -1) {$F$};
\node (Lambda) at (1, -1) {$\Lambda$};
\node (Found) at (0, -2) {Found during testing};
\node (After) at (0, -3) {After debugging};
\node (Before) at (0, -4) {Before testing};
\draw[->] (N) -- (F) node[midway, above] {\downarrow};
\draw[->] (N) -- (Lambda) node[midway, above] {\downarrow};
\draw[->] (F) -- (Lambda) node[midway, above] {\downarrow};
\end{tikzpicture}
\caption{Quantities of faults}
\end{figure}

- $F$ is the estimated number of faults\(^4\) that will be introduced into the module during coding;
- $N$ is the estimated number of faults that will be found and fixed during the testing phase;
- $\Lambda$ is the estimated number of faults that will remain in the module after finishing the testing phase. It is assumed that $\Lambda = F - N$ (perfect debugging).

\(^3\)Software release policies [6, 7, 8] deal with testing and debugging as a single activity.
\(^4\)In [13, 14, 15] an analysis of how to estimate $F$ can be found.
3.1 Relationship between $N$ and required reliability

It is generally agreed that the reliability $R$ of a module is strongly dependent on
the number of faults $\Lambda$ that remain in the module after testing and debugging have
finished; as $\Lambda$ decreases the probability that the module works according to its
specification will increase, that is, $R$ will vary inversely with $\Lambda$.

For a given level of reliability $R$ we need to estimate the testing plus debugging
effort needed to achieve $R$. This effort is determined by $N$, the number of faults
needed to be fixed to reduce the faults from $F$ to $\Lambda$, where $\Lambda$ is sufficiently low that
the reliability is $R$.

Thus we need to estimate $N$, based on a relationship between $N$ and $R$.

In order to make this association the following arguments are considered here:

1) Suppose a module with $S$ lines of code has $\Lambda$ remaining faults. Assuming that
each line of code can hold just one fault, we have $\Lambda$ faulty lines among the $S$
lines.

2) Let $\beta$ be the probability of an individual faulty line being executed and causing
a failure, where we assume that $\beta$ is the same for each faulty line. $\beta$ then
represents the probability that for one run of the module a specific faulty
line will produce a failure. So, for example, $\beta = 0.005$ (considering just one
faulty line) means that for 1000 executions of the module (using different input
data), on average five failures will be caused by this particular faulty line. This
parameter is briefly analysed below.

This same assumption is adopted, for instance, in the early software reliability
models of Jelinski-Moranda, Shooman and Musa [16, 17]. Results obtained
with this assumption have been claimed to yield, in many situations, an overly
optimistic estimate for the behaviour of faults in a software module, as analysed,
for example, in [18]. Many later models have been proposed to overcome
this deficiency. However, it should be stressed that these models rely on data
collected during the testing phase; this data is not available to be used in the
model developed here.

In spite of the clear limitations of the foregoing assumption, the results ob-
tained from the whole model developed in this paper may still be sufficiently
valid, in this early phase of the life-cycle of a software module.

3) We also assume that the manifestation of a fault does not depend on the
occurrence of other faults (i.e., the remaining faults occur independently).

In general, using $P(B_i)$ to denote the probability that a fault $i$ does not man-
ifest for one run of the program, the probability that the 1st, 2nd, $\cdots$, and
$\Lambda$th faults do not manifest for one run of the program is equal to

$\displaystyle P(B_1) \cdot P(B_2|B_1) \cdot P(B_3|B_1B_2) \cdots P(B_\Lambda|B_1B_2\cdots B_{\Lambda-1})$

and it would then be necessary to estimate the conditional probabilities
$P(B_i|B_1B_2\cdots B_{i-1})$ (probability that $i$th fault does not manifest given that
1st, 2nd, $\cdots$, $i-1$th faults does not manifest as well).

With the simplifying assumption of independence, we have
$P(B_1|B_1B_2\cdots B_{i-1}) = P(B_1)$, and, therefore,
$P(B_1) \cdot P(B_2|B_1) \cdot P(B_3|B_1B_2) \cdots P(B_\Lambda|B_1B_2\cdots B_{\Lambda-1})$ reduces to
$P(B_1) \cdot P(B_2) \cdots P(B_\Lambda)$.
Of course, in practice there is often a knock-on effect; the occurrence of one fault may give rise to the occurrence of another fault. However, we cannot build this possibility (the conditional probabilities) into the design phase, because it is completely unknown at the design phase. how a fault (which has not yet been created) might cause other faults to occur.

Then, considering software reliability to be the probability that a module will operate according to its specification when called and will transfer control correctly when finished (which will happen if none of the remaining Λ faults occurs), we then have that the reliability R of the module, is given by

\[ R = (1 - \beta)^Λ \]  

(2)

So,

\[ \ln R = \Lambda \ln(1 - \beta) \]

As Λ = F - N, then

\[ N = F - \frac{\ln R}{\ln(1 - \beta)} \]  

(3)

Observe that, since the right-hand side of equation (3) must be non-negative, a very small value for \( \beta \) will imply a very large value for R.

3.2 Considerations on \( \beta \)

We do not derive an explicit expression for \( \beta \) in this work. Rather we assume that \( \beta \) is estimated during the design phase, based on the historical data of past projects of the same category as the module under development.

We can estimate \( \beta \) from previous project data if we know Λ and the failure rate per runs for the module. Let \( \eta \) represent the failure rate/runs, so

\[ \beta = \frac{\eta}{\Lambda} \]

Example: Suppose a module contains five faults. Suppose that in 1000 runs of the module 10 failures occur, that is, \( \eta = \frac{10}{1000} = 0.01 \) failures per run.

Then \( \beta \approx \frac{0.01}{5} = 0.002 \). This means that if we have “similar” (see page 3) software we might estimate the “failure rate per fault” (the probability that a remaining fault will produce a failure) as being \( \beta = 0.002 \).

4 Effort of Testing

It is argued in this work that the magnitude of the effort of testing is related to the required level of reliability R. We have to find and fix N faults in order to achieve the level of reliability R required and we can estimate N from R using equation (3), if we have estimates for F and \( \beta \). Then, we have to establish a suitable expression for the effort of testing (finding and fixing N faults during the testing phase).

The overall effort \( E_{test} \) to be spent during the testing phase, such that the remaining faults Λ correspond to the desired reliability R, is thus hypothesized to depend directly on two factors: the number of faults N to be removed (we will use “removed” to mean “found and fixed” in this work) and the effort \( \tau_j \) to remove the
jth fault during testing. On this basis, we express the effort of testing as being the sum of the efforts of removing the first, second, \ldots, Nth faults. So,

\[ E_{\text{tes}} = \tau_1 + \tau_2 + \cdots + \tau_N \]  

(4)

4.1 Estimated effort to remove one fault

The underlying assumption that enables us to estimate \( \tau_j \), as analysed in [19], is that the expected average amount of effort required to remove the \( j \)th fault is proportional to the total effort required to implement the module divided by the expected number of faults. This assumption apparently gives a very reasonable approximation for the sought effort, according to examples shown in [19], and is based on the hypothesis that “if there are \( F \) bugs expected in a software module, one would have to understand to some degree \( \frac{1}{F} \) of the program on average for each bug found”.

This assumption is represented here by

\[ \tau_j = \alpha_j \cdot \frac{E_{\text{cod}}}{F} \]  

(5)

where:
- \( E_{\text{cod}} \) is the effort required to implement (coding) the module;
- \( F \) is the expected number of faults;
- \( \alpha_j \) is a factor of proportionality that is employed here to characterize the effort required to remove the \( j \)th fault.

Since no detailed data are available about \( \alpha_j \), we make an assumption based partly on intuition and partly on mathematical convenience, which allow us to establish an expression for this parameter \( \alpha_j \):

- The effort to remove a fault increases during the testing phase, that is, the effort of removing the \( j \)th fault is bigger than the effort of removing the \((j-1)\)th fault. So, \( \alpha_1 < \alpha_2 < \cdots < \alpha_N \).

At the beginning of testing, for an allocated level of effort (man-months) a certain quantity of faults is removed, leading to a sharp decrease in the number of remaining faults. Later in the testing, for the same allocated effort, a great deal less faults can be removed, due to those faults being more “hidden”.

Thus, it can be said that removing faults during the final stages of testing requires a great deal more effort than at the beginning, in a behaviour that is clearly not linear, as envisaged in figure 2.

As analysed in [12], once the curve shown in figure 2 begins to approach a vertical asymptote, this means that testing is either nearing completion or stuck, without achieving new significant improvements. It does not mean that all faults have definitely been removed, but that the current test method has achieved (almost) the maximum number of corrections possible. So, it is plain, following our line of reasoning, that each fault will require a different (increasing) effort to be removed.

On the basis of the above points, it is proposed that the parameter \( \alpha_j \) should vary exponentially in relation to \( j \), taking the behaviour roughly depicted in figure 2.

Now we have to find a manner of expressing \( \alpha_j \) such that the stated conditions are fulfilled. A suitable expression which fulfils all the required properties is:
Figure 2: Each fault requires a different effort to fix

\[
\alpha_j = \rho \cdot e^{j \cdot s} \tag{6}
\]

where \( s \) and \( \rho \) are constant parameters that control, respectively, the steepness and amplitude of the curve showed in figure 2.

Hence,

\[
\tau_j = \rho \cdot \frac{E_{\text{cod}}}{F} e^{j \cdot s} \tag{7}
\]

In order to establish a final expression for \( \tau_j \), expressions for \( \rho \) and \( s \) need to be derived.

4.1.1 Expression for \( s \)

An expression for \( s \) (the rate at which \( \tau_j \) increases) is now derived using the following line of argument.

From equation (7) we see that the effort of fixing the \( N \)th fault is given by

\[
\tau_N = \rho \cdot \frac{E_{\text{cod}}}{F} e^{N \cdot s}
\]

And the effort of fixing the first fault is given by

\[
\tau_1 = \rho \cdot \frac{E_{\text{cod}}}{F} e^{s}
\]

Then,

\[
\frac{\tau_N}{\tau_1} = \delta = \frac{e^{N \cdot s}}{e^{s}} = e^{(N-1) \cdot s}
\]

\[
s = \frac{\ln \delta}{N-1} \tag{8}
\]

where it is assumed that there exists a parameter \( \delta = \frac{\tau_N}{\tau_1} \) (which can be used for the software under estimation) which reasonably represents the ratio between the
effort of finding and fixing the $N$th and first faults\(^5\). To simplify the representation of $\delta$, it is not denoted here as being linked to $N$.

Further work should address this issue, in order to establish an expression for $\delta$ based on data that are available during the design phase.

### 4.1.2 Expression for $\rho$

In order to express $\rho$, we introduce a factor that enables the estimator (for instance, the project manager) to set up an upper bound for the estimated effort that will be expended during testing phase. So, we can say that

$$
\sum_{j=1}^{F} \tau_j = \gamma \cdot \frac{E_{\text{cod}}}{*}
$$

where the factor marked with $(*)$ represents the effort required to remove all $F$ faults and the factor marked with $(**)$ represents the maximum effort of testing that is envisaged by the estimator. The latter is expressed in terms of the effort of coding $E_{\text{cod}}$, where $\gamma$ characterizes the relationship between the effort of coding and the maximum effort of testing\(^6\). In the same way as noticed for the the parameter $\delta$, it is assumed that the parameter $\gamma$ is available during the design phase to be employed for the software under estimation. Then, substituting equation (7) in equation (9), we have

$$
\rho \cdot \frac{E_{\text{cod}}}{F} \sum_{j=1}^{F} e^{j-1} = \gamma E_{\text{cod}}
$$

where the term marked with $(\dagger)$ is the sum of a geometric progression with rate $e^s$. Then,

$$
\rho \cdot \frac{E_{\text{cod}}}{F} \left( \frac{e^{Fs} - 1}{e^s - 1} \right) = \gamma E_{\text{cod}}
$$

and

$$
\rho = \gamma F \left( \frac{e^s - 1}{e^{Fs} - 1} \right)
$$

### 4.2 Expression for $E_{\text{tes}}$

Considering the expression for $E_{\text{tes}}$ in equation (4), we have that

$$
E_{\text{tes}} = \sum_{j=1}^{N} \tau_j
$$

and

$$
E_{\text{tes}} = \sum_{j=1}^{N} \gamma F \left( \frac{e^s - 1}{e^{Fs} - 1} \right) \frac{E_{\text{cod}}}{F} e^{j-1}
$$

\(^5\)In [1, page 40] it is suggested that $1 < \delta < 10$, for smaller software projects.

\(^6\)Some of the evidence shows that $0.4 < \gamma < 2.5$, as can be seen, for instance, in [20, 9]. Therefore, according to these figures, $0.4E_{\text{cod}} < (E_{\text{tes}})_{\text{max}} < 2.5E_{\text{cod}}$. 

9
\[ E_{tes} = \gamma E_{cod} \left( \frac{e^s - 1}{e^{Fs} - 1} \right) \sum_{j=1}^{N} e^{js} \]

where the factor marked with (\(\dagger\)) is again the sum of a geometric progression with rate \(e^s\). Then,

\[ E_{tes} = \gamma E_{cod} \left( \frac{e^s - 1}{e^{Fs} - 1} \right) \left( \frac{e^{Ns} - 1}{e^s - 1} \right) \]

\[ E_{tes} = \gamma E_{cod} \frac{e^{Ns} - 1}{e^{Fs} - 1} \]  \(\text{(11)}\)

### 4.3 Why not testing and debugging separately

Now the argument risen previously, as to why not link cost and reliability considering testing and debugging separately, can be answered.

The effort expended during the testing phase is composed of two factors:

i) The effort required to check that the software is working according to its specifications. This effort may be seen as being:

- mandatory effort required to attend the demands of \(v&v\) during the testing phase, without taking into account any figure for the required reliability as an element of decision;
- supplementary effort of verification and validation to find sufficient soft-ware failures (and software faults) to achieve the desired reliability.

ii) The effort of relating the software failure with a software fault, resulting in fixing the fault.

The former we can associate with the effort of testing in itself, whereas the latter can be associated with the effort of debugging.

Should testing and debugging efforts were considered individually, it can be concluded that new parameters should be introduced, and therefore estimated, in order to establish the relationship between finding and fixing a fault. If it was the case, that is, if adopted the approach of establishing the relationship between cost and reliability based on just the activities of testing and debugging, it can be said that all procedures developed here would roughly be the same, where the parameters \(\delta, \rho\) and \(\gamma\) utilized here should somehow exist.

It follows that there would not seemingly be any significant advantage of using this approach, even more accuracy in the outcome obtained. This approach would instead produce knotty formulas, with more difficult parameters to be estimated, and without a clear advantage in the final result, we claim.

### 5 Estimated Cost of Development

Substituting the value for \(E_{tes}\) obtained in equation (11) in equation (1) we now obtain the final expression for the cost of development.
\[ C_{dev} = P_{dev} E_{cod} \left( 1 + \gamma \frac{e^{Ns}}{e^{Fs} - 1} \right) \] (12)

Considerations on equation (12):

- Values known in advance:
  \( R \) and \( P_{dev} \)
  The cost \( P_{dev} \) must be available, considering the type of human resources to be allocated in the coding and testing phases.
  The level of required reliability \( R \) for the software module is known in advance.

- Values estimated using formulae developed in this work
  \( N \) (equation 3) and \( s \) (equation 8).

- Values estimated using data from the design specification and past projects
  \( E_{cod}, F, \beta, \gamma \) and \( \delta \).
  There must be historical data of previous projects, using similar characteristics of development in the installation under consideration, that allow us to estimate the parameters above.

- The relationship between the development cost \( C_{dev} \) and the required level of reliability \( R \) is roughly depicted in figure 3 (in section 6 there is a more precise graph and an explanation for why there is a "flat" section in this curve). As we might expect, the cost rises markedly when the required reliability approaches 100%. However, and taking into account the assumptions that have been made here, we can estimate in advance of the coding and testing phases how much that cost will be for a required level of reliability.

![Figure 3: Cost of development versus reliability](image)
• One might question in the established formula, why there is a finite cost $C_{dev}$ to achieve reliability $R = 1.0$. (that is, 100% reliability), when a common feeling says this value would be infinite.

The reason is that, in this work reliability is linked to the number of faults that remain in the module, which is assumed to be finite. So one can estimate this number of faults and the effort required (man-months) to fix all those faults; thus maximum associated cost can then be estimated as a finite value.

The crucial matter, which is not addressed here at all, is how to apply software engineering practices, if any, such that the estimated effort $E_{tes}$ be effectively able to find and fix all of the $N$ sought faults, and certify that this has been achieved. If this matter is not fully dealt with during the testing phase, the cost may indeed turn out to be infinite!

6 Numerical Example

Consider a hypothetical example where we wish to estimate the overall development cost $C_{dev}$ for a module with 4000 lines of code, taking into account three scenarios for the required reliability, which are: $R = 0.70$, $R = 0.90$ and $R = 0.95$. As this is a hypothetical example, we do not have data from real software in order to obtain the parameters required by our formulation. So, we have to make some assumptions to acquire reasonable values for these parameters.

• The value for $E_{cod}$ (in man-months) is estimated based on formula obtained in [20], which was derived through regression analysis for projects characterized as system utilities (e.g., tape management and job scheduling).

In that work\(^7\) $E_{cod} \approx 1.7 \cdot S^{0.82}$.

Hypothesizing that the module under estimation is developed in a similar environment to that in [20], we can very crudely use the same formula for effort of coding.

So,

$$E_{cod} \approx 1.7 \cdot 4^{0.82} = 5.3$$

• We hypothesize

$P_{dev} = £2,000.00$ per man-month; $\gamma = 1.0$; $\delta = 5.0$; $\beta = 0.005$.

* The hypothesized value of $\beta$ may be seen as a reasonable guess, based on the following line of arguments.

  i) It has been suggested [16, 12] that a standard expression for reliability is given by

$$R = e^{\eta t}$$

where $\eta$ is the failure rate in any time interval and $t$ is the length of the time interval in which the reliability is estimated.

---

\(^7\)The formula shown in [20] enables us to estimate the overall effort expended in a system utilities project. This effort is said to be $E = 4.27 \cdot S^{0.82}$, where $S$ is the size of the software in KLOC (thousand lines of code). A percentage break-down of effort by phase is presented, which permit us to say, very roughly, that $E_{cod} \approx 0.4E$. Then $E_{cod} \approx 1.7 \cdot S^{0.82}$. 

12
(ii) Let us consider \( t = 1 \) run, that is, the time interval corresponds to 1 run of the software module. Then, \( \eta = -\ln R \) failures per run.

(iii) Suppose \( R = 0.99 \). Then, \( \eta = 0.01 \) failures per run, which, following the example of estimation of \( \beta \) shown previously, produces \( \beta = 0.002 \).

Thus, it can be said that a hypothesis of \( \beta = 0.005 \) seems sound.

* Although \( \delta \) changes depending on the number of faults to be removed, we assume here that \( \delta \) is constant throughout this example. As can be seen in section 8, \( C_{dev} \) is not very sensitive to \( \delta \). This fact enables us to use, in this hypothetical example, a constant value for \( \delta \), which has not produced significant distortions in the results produced for \( C_{dev} \).

The hypothesized values of \( \delta = 5.0 \) and \( \gamma = 1.0 \) seem to be a sound assumption (see footnotes on pages 9).

Then using equation (12), we have for the three scenarios of reliability the following values.

a) \( R = 0.70; C_{dev} = £11,148.98 \)
b) \( R = 0.90; C_{dev} = £17,290.91 \)
c) \( R = 0.95; C_{dev} = £19,193.32 \)

Considerations

a. In figure 4 the curve \( C_{dev} \times R \) is depicted. With this curve, a project manager would be able to have some scenarios for \( C_{dev} \), depending on the required level of reliability for the software.

b. In figure 4 we see that the cost of development for the 4000-lines-of-code module, is in the range £10,600.00—£21,200.00. The lower end of the range coincides with the situation in which the software has just been coded (and not tested at all); the upper end represents the theoretical situation for reliability equal to 100%.

It is worth emphasizing that the "flat" area of the curve shown in figure 4 indicates that below a certain level of required reliability, the development cost for the required reliability less than this value is nearly the same (and \( \approx C_{cod} \)). The point is that having spent \( C_{cod} \) on coding the software, we obtain \( R \approx 0.6 \). Then, if our requirement is \( R < 0.6 \) for these modules, we still have to spend \( C_{cod} \).

The level of required reliability mentioned above, can be estimated using in equation (3) \( N = 1 \). In this case, the reliability that would be achieved after just one fault having been removed is estimated as follows.

\[
1 = 115 - \frac{\ln R}{\ln (1 - 0.005)} \quad R = 0.564
\]

Hence, when \( R > 0.564 \), each fault removed results in a significant increase in the cost \( C_{dev} \). Below this threshold \( C_{dev} \) is nearly constant.

c. Despite the fact that we are using hypothetical, but as far as possible realistic, values for some parameters, which makes a precise comparison impossible, we present a very rough analysis of the results obtained here, when compared with other data available in the literature, in which, it
Figure 4: Curve $C_{dev} \times R$ for the hypothetical example

should be stressed, a target reliability is not clearly cited. This comparison confirms (again, very roughly) that the results produced here, using the developed formulas, seem to yield a sound outcome. The module with 4000 lines of code is used for this comparison.

Our results are compared with real data collected in [20]. The total effort $E$ of development (design + code + test) has the following formula in [20], which is based on real data gleaned in several different environments.

$E = 4.27 \times 4^{0.82} \approx 13.3$ man-months.

The percentage of effort spent in the coding and testing phases represents approximately 80% of the value calculated above, again according to data shown in [20]. Thus $E_{cod} + E_{c&v} = 13.3 \times 0.8 \approx 10.64$ man-month.

Thus the total cost of coding and testing, using the data acquired in [20] would be $C_{dev} \approx 10.64 \times 2000 = £21,280.00$.

As can be seen in figure 4 the cost of development for a module with 4000 lines of code (considering the characteristics defined for those examples) is around £20,000.00 (for reliability equal to .90 or 0.95).

d. Therefore, it may be said that the result produced here as expected has yielded a reasonably consistent estimate when compared with [20], however, it should again be emphasized, that a required level of reliability was used in the estimate here.
It can be claimed that the representation proposed here for $C_{dev}$ is clearly a reasonable estimate for the cost of development that can be acquired during the design phase, because the relationship between cost and reliability is modelled realistically and produces sound outcomes. Furthermore, it has to be stressed, we are also associating a required level of reliability in that estimate, which, to our knowledge, has not been developed within the same context elsewhere.

7 Sensitivity

An analysis of sensitivity is an attempt to identify combinations of data values within permissible limits that can cause particularly significant variations in the results, as defined in [9].

It is well-known that a partial derivative (gradient) of a function in relation to a variable gives a clear indication of the relative behaviour of the function, particularly the effect that changes in some variables would have on the function. Thus, the partial derivative of the function enables us to find the sensitivity of the function for changes in its parameters. So, the sensitivity of a function defines the proportional effect on the function for changes in a variable.

If $u = f(x_1, \cdots, x_n)$, then the sensitivity of $u$ with relation to $x_i$ is expressed by

$$\Theta_{x_i} = \left| \frac{x_i}{u} \right| \left| \frac{\Delta u}{\Delta x_i} \right|$$

When the variation $\Delta x_i$ tends to 0, then the term $\left| \frac{\Delta u}{\Delta x_i} \right|$ tends to the partial derivative $\left| \frac{\partial u}{\partial x_i} \right|$ [22].

Then, the expression for sensitivity is given by

$$\Theta_{x_i} = \left| \frac{x_i}{u} \right| \left| \frac{\partial u}{\partial x_i} \right|$$ (13)

8 Sensitivity of the Cost of Development

To clarify the sensitivity aspect of the formula (12) for the employed parameters, we calculate the partial derivative of $C_{dev}$. For each parameter a brief discussion is made on the influence that changes in its value would exert on $C_{dev}$. The module with 4000 lines of code is used for illustration.

As seen in formula (12), the ensuing parameters have to be estimated:

- $P_{dev}$; $E_{cod}$; $\beta$; $F$; $\delta$; and $\gamma$.

8.1 Sensitivity due to $P_{dev}$ and $E_{cod}$

Let the sensitivities due to $P_{dev}$ and $E_{cod}$ be denoted, respectively, by $\Theta_{P_{dev}}$ and $\Theta_{E_{cod}}$. Using equation (13) the following results are obtained.

As can be seen in equation (12), $C_{dev}$ varies linearly and in direct proportion to $P_{dev}$ and $E_{cod}$. Thus, a foregone conclusion is that $C_{dev}$ varies in the same proportion to the variations in $P_{dev}$ and $E_{cod}$. So,

$$\Theta_{P_{dev}} = 1; \quad \Theta_{E_{cod}} = 1$$
8.2 Sensitivity due to $\gamma$

Let $\Theta_\gamma$ denote the sensitivity of $C_{dev}$ due to $\gamma$. Using equation (13) it is seen that

$$\Theta_\gamma = \left| \frac{\gamma}{C_{dev}} \cdot \frac{\partial C_{dev}}{\partial \gamma} \right|$$

$$\Theta_\gamma = \frac{\gamma}{C_{dev}} \cdot P_{dev} \cdot E_{cod} \cdot \frac{e^{Ns} - 1}{e^{Fs} - 1}$$

$$\Theta_\gamma = \frac{\gamma(e^{Ns} - 1)}{(e^{Fs} - 1) + \gamma(e^{Ns} - 1)}$$

$$\Theta_\gamma = \frac{\gamma}{\gamma + \frac{e^{Fs}-1}{e^{Ns}-1}}$$  \hspace{1cm} (14)

Applying the values utilized in the example of chapter 5, for a module with 4000 lines of code, it is found

$$\Theta_\gamma = 0.387$$

Thus, it can be concluded that $C_{dev}$ is definitely less sensitive to $\gamma$ than to $P_{dev}$ and $E_{cod}$.

8.3 Sensitivity due to $\beta$

The parameter $\beta$ is used in formula (3), which is repeated below.

$$N = F - \frac{\ln R}{\ln(1 - \beta)}$$

We require $\frac{\partial C_{dev}}{\partial \beta}$. As can be seen, $\beta$ affects $N$, which, in turn, also affects $s$ (formula (8)). And both exert influence on $C_{dev}$.

And,

$$\Theta_\beta = \left| \frac{\beta}{C_{dev}} \cdot \frac{\partial C_{dev}}{\partial \beta} \right|$$  \hspace{1cm} (15)

The result for $\frac{\partial C_{dev}}{\partial \beta}$ is a very complicated expression. An expression for $\frac{\partial C_{dev}}{\partial \beta}$ was obtained using the software MATHEMATICA [23]. When used in equation (15) and employing the same data of example in chapter 5, the result

$$\Theta_\beta = 0.20524$$

was produced.

As can be seen $\Theta_\gamma > \Theta_\beta$. Thus, it may be concluded that $C_{dev}$ is more sensitive to $\gamma$ than to $\beta$. 

16
8.4 Sensitivity due to $F$

This parameter has a direct effect on $N$, that is, $N$ varies in direct proportion to $F$. However, as $C_{dev}$ does not vary in the same proportion to changes in $N$, this implies that $F$ does not affect $C_{dev}$ in direct proportion as well.

$$\Theta_F = \left| \frac{F}{C_{dev}} \cdot \frac{\partial C_{dev}}{\partial F} \right| \quad (16)$$

The software MATHEMATICA was again employed to find $\frac{\partial C_{dev}}{\partial F}$. Applying the values employed in chapter 5 in equation (16), we obtained

$$\Theta_F = 0.20592$$

Thus, it may be concluded that $C_{dev}$ is slightly more sensitive to $F$ than $\beta$, but $C_{dev}$ is less sensitive to these two parameters than to $\gamma$.

8.5 Sensitivity due to $\delta$

In formula (12) it can be seen that $C_{dev}$ varies in inverse proportion to the parameter $s$, that is, if $s$, say, increases the expression $(e^{Fs} - 1)$ will increase more rapidly than $(e^{Ns} - 1)$, which means that $C_{dev}$ will decrease. As $s$ varies in direct proportion to $\delta$ (formula (8)), then it can be concluded that $C_{dev}$ varies in inverse proportion to $\delta$.

$$\Theta_\delta = \left| \frac{\delta}{C_{dev}} \cdot \frac{\partial C_{dev}}{\partial \delta} \right| \quad (17)$$

To verify the above conclusion, the software MATHEMATICA was again employed to find an expression for $\frac{\partial C_{dev}}{\partial \delta}$. When this is applied in equation (17), then using the data previously mentioned, produces

$$\Theta_\delta = 0.06918$$

Comparing the value for $\Theta_\delta$ to the previous analysed parameters, it may be concluded that $\delta$ affects $C_{dev}$ the least.

8.6 Summary of the sensitivities in $C_{dev}$

To show the sensitivity of $C_{dev}$ for the parameters involved in its formulation, the parameters that need to be estimated by the user, namely, $P_{dev}$, $E_{cod}$, $\delta$, $\beta$, $F$ and $\gamma$, were analysed. By doing so, we are able to see which influence an inaccurate estimate for these parameters may exert on $C_{dev}$.

An overall view of the discussion on sensitivity is shown in Table 1, where the following classification is employed:

- Level 1: minimal influence at all ($0 < \Theta \leq 0.1$).
- Level 2: moderate influence ($0.1 < \Theta \leq 0.5$).
- Level 3: steady influence ($0.5 < \Theta \leq 1.0$).
- Level 4: large influence ($1.0 < \Theta$).

On this basis, it may be suggested that $C_{dev}$ is not ill-conditioned and is reasonably sensitive for the parameters involved. It can also be said that
<table>
<thead>
<tr>
<th>$P_{dev}$</th>
<th>$E_{cod}$</th>
<th>$\delta$</th>
<th>$B$</th>
<th>$F$</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0.06918</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0.20524</td>
<td>0.20592</td>
<td>0.3870</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Summary of the sensitivity analysis

- There is no parameter classified as exerting "large influence" on $C_{dev}$.
- An inaccurate estimate for $\delta$ does not affect $C_{dev}$ badly. This fact, as discussed previously, enabled us to use $\delta$ constant throughout the examples shown in chapter 5.
- The parameters $P_{dev}$ and $E_{cod}$ exert a constant influence on $C_{dev}$. Any inaccurate estimates for these parameters $C_{dev}$ will be affected directly in the same proportion.
- The remaining parameters exert an unexceptional influence on $C_{dev}$.

It has to be said that a combination of changes in different parameters may result in a variation of $C_{dev}$, which, seemingly, is not easy to predict. Therefore, the classification indicated in Table 1 should be used just as a crude guideline for the sensitivity analysis only.

9 Conclusion

In line with recent published papers [24, 25], which show that a project manager wishes to utilize simple methods for estimating development cost, this article has proposed an uncomplicated method to estimate the cost of coding and testing a software system. This estimate is based on data available at the design phase, i.e., before beginning the coding phase, and, as the main contribution of our work, taking into account a required level of reliability for each module. We have derived some formulas that enable a project manager to obtain the required estimates, in a relatively straightforward way.

The outcome of this work, despite being still very preliminary, allows us to say that the method proposed here is clearly a step forward in formulating a tradeoff between cost and reliability, during the design phase. Thus, we would claim, considering the approach described, a project manager could deal with various scenarios of cost and reliability, before allocating the resources for the coding and testing phases, which could lead to better management of the software project as a whole\(^8\).

Acknowledgments

We wish to thank Dr. Chris Phillips for his helpful suggestions and useful comments during the development of this work.

\(^8\)This work has been partially funded by CNPq/Brazil (grant no. 200487/92-2) and ESPRIT Basic Research Action PDSC.
References


[22] Jeffrey, A. Mathematics for engineers and scientists, Chapman-Hall, 1992

