https://doi.org/10.5028/jatm.v11.1031 ORIGINAL PAPER

Applying COCOMO II for a DO-178C Safety-Critical Software Effort Estimation

Lucas Pereira dos Santos[1,*], Maurício Ferreira[1]

Santos LP https://orcid.org/0000-0003-1859-4970
Ferreira M https://orcid.org/0000-0002-6229-9453

How to cite

Santos LP; Ferreira M (2019) Applying COCOMO II for a DO-178C Safety-Critical Software Effort Estimation. J Aerosp Technol Manag, 11: e1819. https://doi.org/10.5028/jatm.v11.1031

ABSTRACT: This paper provides a real example of applying COCOMO II as an estimation technique for the required software development effort in a safety-critical software application project following the DO-178C processes. The main goal and contribution of the case study is to support the research on software effort estimation and to provide software practitioners with useful data based on a real project. We applied the method as it is, by correlating the effort multiplier factors with the complexity and objectives introduced by the DO-178C level A application, resulting in an estimated effort. The rationales for each scale factor and effort multiplier selection were also described in detail. By comparing the estimated values with the actual required data, we found a magnitude of relative error (MRE) of 40% and provided alternatives for future work in order to increase the effort estimation accuracy in safety-critical software projects.

KEYWORDS: Software development, Software engineering, Project management, Engineering management, Cost estimates.

INTRODUCTION

The software industry is, by far, one of the businesses that have been introducing most of the major innovations in the modern world (WIPO 2016). Since the creation of the smartphone concept, a variety of opportunities came up for entrepreneurs and software developers in general (Capaldo *et al.* 2003). The way to order a sandwich, call a cab, get a ride from a stranger, and even to find a date have changed completely in the past years. The software development industry is highly responsible for this huge evolution (Chesbrough *et al.* 2006). However, this considerable innovation also requires some additional care regarding the responsibility that the software product carries in our everyday life. Calling a cab by an app on a smartphone, for example, will depend upon a series of safety issues, once the car will likely be unmanned, which will be possible due to the use of autonomous navigation technology (Scaramuzza *et al.* 2014)environment monitoring, security surveillance, and inspection. If they are further realized in small scale, they can also be used in narrow outdoor and indoor environments and represent only a limited risk for people. However, for such operations, navigating based only on global positioning system (GPS. Such caution regarding safety is already a reality for the software development industry, known as safety-critical (Rierson 2013)software technology is changing, projects are pressed to develop software faster and more cheaply, and the software is being used in more critical ways. Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance equips you with the information you need to effectively and efficiently develop safety-critical, life-critical, and mission-critical software for aviation. The principles also apply to software for automotive, medical, nuclear, and other safety-critical domains. An international authority on safety-critical

1.Instituto Nacional de Pesquisas Espaciais – Engenharia e Tecnologia Espaciais – Engenharia e Gerenciamento de Sistemas Espaciais – São José dos Campos/SP – Brazil.

*Correspondence author: lucas.pds@gmail.com

Received: May 06, 2018 | Accepted: Jul. 30, 2018.

Section Editor: Adam Cumming

software, the author helped write DO-178C and the U.S. Federal Aviation Administration\u2019s policy and guidance on safety-critical software. In this book, she draws on more than 20 years of experience as a certification authority, an avionics manufacturer, an aircraft integrator, and a software developer to present best practices, real-world examples, and concrete recommendations. The book includes: An overview of how software fits into the systems and safety processes Detailed examination of DO-178C and how to effectively apply the guidance Insight into the DO-178C-related documents on tool qualification (DO-330, which means that the software is safety-critical for the human being and demands a large amount of money (Pries and Quigley 2011).

In the aviation industry, for instance, the flight control of an aircraft equipped with the Fly-By-Wire (FBW) technology is the full responsibility of the embedded software that manages all actuators and pilot commands. In the past, the biggest challenge in the development of an airplane was to deal with the aerodynamic issues; currently, the software is the item of greatest concern and, in some cases, it is responsible for long delays in launching a new product (Kuznecova 2017). In this context, the effort estimation in software development projects presents itself as an alternative to reduce delays and losses because it offers methods to predict the cost, time, and effort required to achieve the desired results.

This paper presents a specific literature review on effort estimation methods in general, the concept of critical software, and a case study of effort estimation of software development using the COCOMO II (Boehm *et al.* 2000) method in a critical software project that follows the processes defined by the DO-178C (RTCA 2011e) standard.

The next sections of this article are organized as follows: section 2 shows the related work; in section 3, the main methods of effort estimation are presented; section 4 describes the critical software concept in more detail and the DO-178C; the case study of a critical software development project in the aeronautical industry is presented in section 5; a discussion of the results obtained is presented in section 6; and, finally, section 7 presents the final considerations and future work.

RELATED WORK

Estimating effort in software development projects is a subject widely researched by academia. The study reported in have identified 304 articles on this subject in an extensive systematic review (Jørgensen and Shepperd 2007). The various estimation techniques, as well as their imprecision problems, have been the subject of study by several researchers. Several models or variations have been created and improved over the years.

In the paper, Idri *et al.* (2000) proposed an adaptation of COCOMO using Fuzzy Logic. The same approach was discussed by Sheta and Aljahdali (2013) and Sarno *et al.* (2015). In Boehm and Valerdi (2008), many variations of COCOMO over the years and new methods that were derived from it due to specific calibrations for a given segment were presented. In spite of the well-known use of COCOMO, Jørgensen and Shepperd (2007) raised a concern regarding the low number of records of studies and results obtained with the application of COCOMO in the scientific databases.

In de Barcelos Tronto *et al.* (2008) and Finnie *et al.* (1997), the authors proposed a better precision in the effort estimation techniques involving neural networks. And Velarde *et al.* (2016) proposed a method based also on lines of code, one of the entries of the COCOMO method.

Khatibi and Jawawi (2011) also present the results of using COCOMO as a method of estimating effort in a real project.

Due to the complexity of safety-critical software applications and the number of processes and activities involved in the DO-178C standard, different metrics and management approaches are also objects of research.

In Estrada *et al.* (2013), the authors provided different approaches and best practices for model-based design to develop a software certified by DO-178C. Paz and El Boussaidi (2016) also focused on model-based design as a support for DO-178C compliant projects.

Finally, a set of compliance metrics for DO-178C objectives, which aims to prevent delays in the certification schedule of the aircraft, was proposed by Yelisetty *et al.* (2015). It must be mentioned, however, that the software development effort was not the goal of their research, but instead metrics regarding the technical aspects of the software product itself.

SOFTWARE DEVELOPMENT PROJECT EFFORT ESTIMATE

In this article, we adopt the definition of effort estimation used by Basten and Mellis (2011) "*the effort that is most likely to be needed to implement the task that is estimated* (*excluding any risk buffer*)". This definition is also aligned with the recommendation made by Jørgensen (2014); he suggests that the meaning of "effort estimation" must be understood and communicated as a probability-based terminology.

There are several techniques for estimating effort in software development projects. Different authors classify them in several ways. Khatibi and Jawawi (2011), for instance, classify all techniques in only two groups: (a) non-algorithmic and (b) algorithmic methods.

Non-algorithmic methods are based on comparisons, inferences, analogies, or opinion of specialized and experienced people to estimate the effort required in a project. Basically, it does not implement any kind of mathematical equation to obtain the results (Basten and Mellis 2011).

Algorithmic methods are those that use a given mathematical equation to relate software metrics to the effort required (Jørgensen and Shepperd 2007; Abbas *et al.* 2012). In general, the main input of the algorithm is the software's size, measured in number of lines of code, for example, which is then processed by the model and converted into the required effort estimate, usually a measure of the total development hours.

Of the 304 articles surveyed by Jørgensen and Shepperd (2007), for example, 148 (49%) dealt with algorithmic methods. In this category, the well-known COCOMO is included, being developed by Boehm (1981) and the main method to be explored in this paper.

COnstructive COst MOdel

The COCOMO method is named according to the initials in English for *COnstructive COst MOdel*, being originally proposed by Boehm in 1981, and therefore also known as COCOMO'81. The COCOMO II (Boehm *et al.* 2000) was then published from this first version, incorporating about 20 years of evolution of the software engineering discipline in general. To avoid repetition of references, this entire section is based on the book *Software Cost Estimation with COCOMO II* (Boehm *et al.* 2000), except when specified.

The two main formulas of the method are used to calculate the effort (Eq. 1) and the required scheduled (Eq. 3) for certain projects. These formulas are presented as follows.

The measure of effort in PM (person-months) is defined by Eq. 1 and Eq. 2, as follows:

$$PM_{NS} = A * Size^{E} * \prod_{i=1}^{n} EM_i \tag{1}$$

where PM is person-months (1PM = 152h); NS is nominal-schedule; A is constant based on model calibration; Size is the software's size in source lines of code (SLOC) or function points; EM is effort multiplier; and n is number of EM taken in consideration.

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j \tag{2}$$

where B is constant based on model calibration; and SF is scale factor.

The time to development (TDEV) in months parameter is defined by Eq. 3 and Eq. 4:

$$TDEV_{NS} = C * (PM_{NS})^{F} \tag{3}$$

where TDEV is time to development; NS is nominal-schedule; C is constant based on model calibration; and PM is person-months.

$$F = D + 0.2 * 0.01 * \sum_{j=1}^{5} SF_j \ = D + 0.2 * (E - B) \quad (4)$$

where D is constant based on model calibration; SF is scale factor; and E and B were described in Eq. 2.

The term nominal-schedule (NS) in the equations means that it excludes the multiplicative scale factor (SCED), which will be detailed in this paper. The term scale factor (SF) is described as follows.

The COCOMO II method presents five scale factors and they basically determine the savings and costs of the project under development. The scale factors are listed as follows:

- PREC (precedentedness): Defines the level of similarity with previous projects (Khatibi and Jawawi 2011).
- FLEX (development flexibility): Indicates the level of software flexibility according to the development process, interface constraints, requirement, and schedule.
- RESL (architecture/risk resolution): Based on the risk assessment analysis results (Khatibi and Jawawi 2011).
- TEAM (team cohesion): Indicates the integration level of the team and human factors.
- PMAT (process maturity): Related to the process maturity level according to CMMI (CMMI Product Team 2010) or similar.

Each scale factor has six levels, from very low to extremely high, and each level has an associated weight suggested by the method (Boehm *et al.* 2000). In addition to the scale factors, the method also presents 17 multiplicative cost factors. They are organized into four categories:

- Product factors: In this category, there are the factors that may cause variation in the required effort due to the product itself and its characteristics – required software reliability (RELY), database size (DATA), product complexity (CPLX), developed for reusability (RUSE), and documentation needs (DOCU).
- Hardware factors: This category considers the factors related to the hardware in which the software will be executed – execution time constraint (TIME), main storage constraint (STOR), and platform volatility (PVOL).
- Human factors: This factor is related to the influence that the team can have under the required effort – analyst capability (ACAP), programmer capability (PCAP), personnel continuity (PCON), application experience (APEX), platform experience (PLEX), and language and tool experience (LTEX).
- Project factors: In this category, there are the project factors that influence the effort, such as the use of modern tools, location of the team, level of interaction, etc. The factors are use of software tools (TOOL), multisite development (SITE), and required development schedule (SCED).

Each of these attributes determines a multiplication factor that estimates their effect on the software development effort. These attributes are classified on a six-level scale from "very low" to "extremely high", and for each level a weight is also given by the method.

Summarizing the COCOMO method, it can be considered a framework, which groups 17 multiplicative cost factors in four different categories, as described in Fig. 1.

To situate the reader in applying this method in a critical software development project, the next section details the concepts of this type of software and its definition.

SAFETY-CRITICAL SOFTWARE

NASA has a critical software development guide called the NASA Software Safety Guidebook (NASA 2004), which defines critical software as follows: "*software is considered critical to safety if it controls or monitors hardware or software that is dangerous or critical to safety*". In general, such applications are embedded and considered real-time applications, which means that they

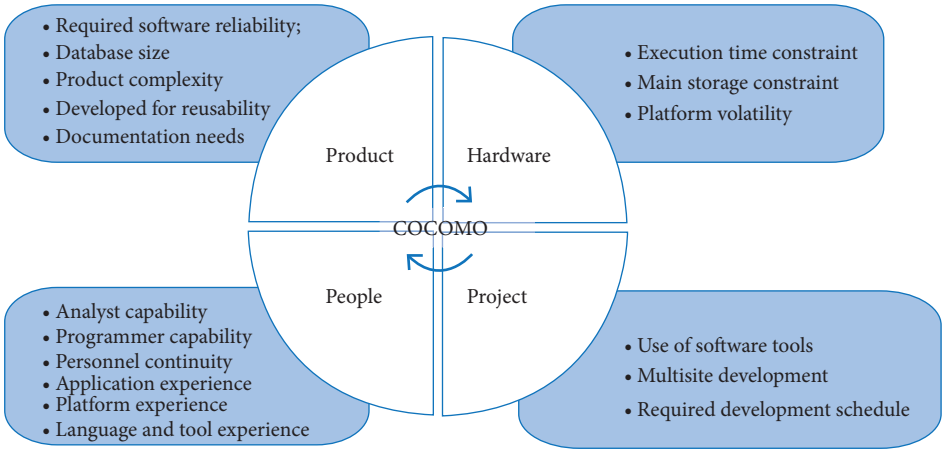

Figure 1. COCOMO II diagram.

operate at an extremely high frequency and have strict response requirements in which a single fault can generate catastrophic events (McCormick et al. 2011). Romani et al. (2010) emphasize the importance of requirements for space software systems according to the NASA criticality scale.

A software for such applications is generally developed following rigorous process standards. In this context, the following are the three of the most used standards in their segments, the DO-178C for the aeronautical industry, the NASA-STD-8719.13C (NASA 2013) for the aerospace industry, and the ISO/IEC62304 (BSI 2006) for the medical segment.Because it is a case study of an aeronautical sector project, the concepts of DO-178C will be further described. To avoid repetition of references, this entire section is based on the standard itself, unless otherwise specified.

Because it is a case study of an aeronautical sector project, the concepts of DO-178C will be further described. To avoid repetition of references, this entire section is based on the standard itself, unless otherwise specified.

As clearly summarized by Marques and Marques da Cunha (2017), "*the DO-178C establishes considerations for developers, installers, and users when designing an embedded equipment*". The DO-178C classifies the criticality of the effect of failures on the crew or plane into five categories. Due to the effects of each failure, the DO-178C then classifies the software in five different levels (A, B, C, D, and E), according to its contribution to the potential system failures, where the level A is the most rigorous. For each software level, the standard presents a set of objectives that must be achieved in order to certify such an embedded application.

The software level determination is done by a system safety assessment process, identification of potential failures, loss of function or malfunction, and their impacts. According to DO-178C, the software level implies that the level of effort required to show compliance with certification requirements varies with the failure condition category. The necessary effort to approve the software increases as the number of objectives increases according to its level. The Table 1 describes the system failure condition associated with the software level and the number of objectives.

The DO-178C standard also presents supplementary documents including special recommendations for tools qualification (RTCA 2011a); model-based development (RTCA 2011b); object-oriented (RTCA 2011c); and formal methods (RTCA 2011d)

Given the DO-178 usage history through its different revisions, in addition to the level of confidence, this document has earned from industry and certification agencies, including the world's leading aeronautical certification agency, the Federal Aviation Administration (FAA), the recognition of this standard and its supplements as a valid mean to show compliance with the regulations for certification of software aspects, by issuing the AC20-115D (FAA 2017).

The case study described in this paper addresses all the concepts presented to the reader up to this moment, being a critical software, following the processes established by the DO-178C, and applying the COCOMO II method as a development effort estimation. Since it is a project that was concluded in 2017, it is possible to present not only the estimated data, but also the actual values demanded by the project.

Table 1. Failure condition categories, software level, and number of objectives.

Failure condition category	Software level	Number of objectives
Catastrophic	A	71
Hazardous	B	69
Major	C	62
Minor	D	24
No safety effect	E	0

CASE STUDY – AERONAUTICAL SECTOR

We report in this article a case study of the COCOMO II method applied in a safety-critical level A software development project of an aeronautical company. It begins with the contextualization of the company Harpia, environment in where the case study was based, following the description of the project and the software development process adopted. Finally, we present the results estimated by the COCOMO II method and the effort that was actually necessary to develop the software under analysis.

THE HARPIA COMPANY

Harpia is the fantasy name of the company responsible for the project and was created especially for this article in order to protect the real name of the company being portrayed in this section for commercial and intellectual property reasons. Harpia has more than 20 years in the software development market in Brazil and it is specialized in software embedded in mission systems for defense aircraft and real-time applications for critical systems. Since its founding, it has grown exponentially and has about 200 software engineers currently. Its main clients are aviation companies from all over the world, which subcontract their services in the defense projects carried out by the Navy, the Army, and the Air force, as well as the development of applications for systems of flight control, called Fly-By-Wire.

In addition to embedded software, the complete company's portfolio includes simulation systems and desktop applications for the planning and debriefing of tactical missions conducted by military Air forces.

THE FLY-BY-WIRE APPLICATION

The project in this case study refers to the software development of the FBW system of the Harpia-H2 program. The complete project development cycle was from June 2013 to December 2017, totaling 4.5 years and consuming at peak 48 software engineers working full time.

PROJECT SCOPE

The embedded software development project of the Harpia-H2 FBW system was split into two companies. A US vendor was hired to develop low-level hardware and platform software, which was responsible for basic hardware interface functions and management of data inputs and outputs through digital communication channels. Harpia was then responsible for developing the control law application, the testability functions, and the failures management.

EFFORT ESTIMATION

During the project-planning phase, at the beginning of development, the COCOMO II method was adopted to assist in the estimating effort and development time required to complete the software development.

The method was then applied in its essence, without performing a dedicated calibration for the company, and with the factors of scale and multiplicative factors of cost being answered considering the complete software application developed by both companies, Harpia and its supplier.

In order to estimate the software's size (the parameter KLOC of the COCOMO II method), by applying the methods based on expert opinion and analogy, the team agreed in considering a value of 140KLOC, based mainly on data from previous project values similar to this one.

The scale factors of the COCOMO II method were analyzed by a team of experienced software engineers and the result is indicated in Table 2. Once the level for each scale factor was identified, its respective value was then automatically obtained as presented by the method itself.

The Table 2 shows all the values for each selection of the scale factors.

Table 2. Scale factor.

Scale factors	Selection	Value
Precedentedness	Nominal	3.72
Development flexibility	Very low	5.07
Architecture/risk resolution	Nominal	4.24
Team cohesion	Very low	5.48
Process maturity	Very high	1.56

By the concepts of the COCOMO II method, the selection of project scale factors is somewhat subjective. In this way, it is necessary to detail the premises and considerations taken into account by the project team during the selection.

The PREC parameter, which indicates the level of similarity of the current project to previous projects, was selected as nominal, since the company has only one similar previously developed project; however, it has conducted a long-term research project in this area. This balancing between years of research, but only one previous project, led the team to select the nominal option.

The FLEX parameter, which indicates the level of flexibility of the software in terms of the development process, the interface constraints, requirements, and deadline, was selected as very low, since the need for certification of such software, as well as compliance with the standard DO-178C, made the process less flexible. In addition, the project was born with a tight schedule, little flexibility to adjust the deadline due to the required date of launching the product on the market, and the fact that it involved the development by an external supplier, which limited the changes of interface.

The RESL parameter, which reflects the result of the risk analysis, was selected as nominal, since the company had a good risk management, but the low precedence index and previous experience, identified by the PREC factor, could introduce hidden and not mapped risks, as well as the involvement of a third-party company as a supplier.

The TEAM parameter, which indicates the level of team integration and human factors, was also selected as very low, since in the project planning phase the team was not yet fully formed, people did not know each other, or they were new to the company, which made it difficult to analyze this parameter. On top of that, the development team includes the supplier personnel, which was based in a different country, with a different culture and background. Therefore, not only technical aspects, but also human factors and culture should be considered in this parameter.

Finally, the PMAT parameter, which indicates the level of maturity of the company's processes, was selected as very high due to the accumulated experience and the maturity of the processes developed and experienced in the research projects prior to the development of this product.

Once the scale factors were defined, it was possible to calculate the parameter "E" according to Eq. 2:

$$E = 0.91 + 0.01 * (3.72 + 5.07 + 4.24 + 5.48 + 1.56) = 1.1107$$

It can be observed that the project presented E > 1.0; in other words, the project presented increasing costs as the scale increases. If the product's size doubled, the project effort is more than doubled.

Following the method steps, the same group of engineers then dedicated themselves to identify the multiplicative factors of project effort. They were all analyzed and answered according to Table 3. Similar to scale factors, once the level for each multiplicative effort factor was identified, its respective value was then automatically obtained as presented by the method itself. The values for each selection of the multiplicative stress factors are presented by the method itself (Boehm *et al.* 2000).

Table 3. Effort multipliers.

Cost drivers (as in COCOMO II)	Selection	Value
Required software reliability (RELY)	Very high	1.26
Database size (DATA)	Nominal	1
Product complexity (CPLX)	Very high	1.34
Developed for reusability (RUSE)	Nominal	1
Documentation needs (DOCU)	Very high	1.23
Execution time constraint (TIME)	High	1.11
Main storage constraint (STOR)	Nominal	1
Platform volatility (PVOL)	Nominal	1
Analyst capability (ACAP)	Low	1.19
Programmer capability (PCAP)	Very high	0.76
Personnel continuity (PCON)	Very high	0.81
Applications experience (APEX)	Low	1.1
Platform experience (PLEX)	Nominal	1
Language and tool experience (LTEX)	Nominal	1
Use of software tools (TOOL)	Very high	0.78
Multisite development (SITE)	Very low	1.22
Required development schedule (SCED)	Low	1.14

As with scale factors, the selection of multiplicative cost factors for the project is also subjective. In this context, it is again necessary to detail the assumptions and considerations taken into account by the project team during the selection.

The RELY parameter, which measures the effect of a software failure on the function which it must run over a period, has been selected as very high; because it was a flight control software, the requirements and reliability and safety of the product are the highest. This was certainly one of the factors that contributed to the multiplication of project costs.

The DATA parameter, which relates to the effect of testing large amounts of data on product development, was selected as nominal to not influence the estimation, since little was known about the amount of test data that would be needed.

The CPLX parameter measures the complexity of the product divided into five areas: (1) operation control; (2) computational operations; (3) devices operations; (4) data management; and (5) user interface operations; it was selected as very high. Although the application is extremely complex, following the method of the project in question did not present the characteristics that would qualify it to be selected as extremely high, since there was no distributed processing or graphical interface.

The RUSE parameter, which relates to the additional effort that will be required to develop the software with the intention of being reused by future projects, was selected as nominal to not interfere with the estimation, since no development was planned to be reused in the future.

The DOCU parameter, which is evaluated in terms of the need for documentation throughout the life cycle of the project under development, was selected as very high, depending on the amount of documents required throughout the life cycle of a software development project level A, according to DO-178C.

The TIME parameter, which is the measure related to the constraint for software runtime, was selected as high, since there was a clear requirement that the software should have consumed a maximum of 70% of the processing throughput by the time of certification.

The STOR parameter, which represents the degree of restriction of the main data storage medium of the system, was selected as nominal to not influence the estimation, as little was known about the amount of data that would be needed.

The PVOL parameter, which is measured from the point of view of the number of platform changes, was selected as nominal to not influence the estimation, since platform software is the responsibility of the vendor.

The ACAP parameter, which analyzes the analyst's ability, efficiency, and accuracy, also the ability to communicate and cooperate with other team members regarding the requirements, architecture, and detail of the project, was selected as low since the team, as already mentioned, was still in training and several members of the team had little experience.

The PCAP parameter, which analyzes the analyst's ability, efficiency, and accuracy, also the ability to communicate and cooperate with other team members regarding programming, was selected as very high, since even if it was a new and recently-trained team, all had prior experience with programming and software development.

The PCON parameter, which is selected in terms of the annual turnover of the project team, the percentage of people leaving the team during development, was selected as very high, which in this case should be considered as a reduction of effort in the project, since the turnover rate of the company is less than 3% per year.

The APEX parameter, which measures the level of applications experience of the project team developing the software system, was selected as low for the same reasons previously mentioned, the team was still under construction and with several newly hired members.

The PLEX and LTEX parameters, which consider the influence on productivity of the development team experience level in the platform in question, were selected as nominal to not influence the estimation, since platform software was the responsibility of the supplier.

The TOOL parameter, which considers the use of sophisticated tools in software development, was selected as very high, since the project used sophisticated tools qualified by the DO-330, such as generated automatic code. The use of such tools in fact should be treated as a cost reduction factor, since its value is < 1.0, which means that it reduces the final value of the product of all multiplicative cost factors.

The SITE parameter, which measures the effect of the globalization and the distribution of development teams, was selected as very low because the development involved three different locations (São Paulo, Minas Gerais, and the United States) and the communication tools were limited, mainly between Brazil and USA.

Finally, the SCED parameter, which considers the scheduling restrictions imposed on the software development team, was selected as low because the project schedule had only 85% of the nominal value.

Once all the necessary entries were defined, it was possible to calculate the estimated number of PM, according to the Eq. 1. In this first moment, the nominal value was calculated by excluding the SCED parameter of the product of all multiplicative cost factors. Thus, n = 16.

$$\prod_{i=1}^{16} EM_i = 1{,}767$$

$$PM_{NS} = 2.94 * (140)^{1.1107*} (1.767) = 1257 \text{ PM}$$

It should be emphasized that the PM parameter refers to the estimated value of person-months. A PM is the amount of time a person devotes working on software development per month. The COCOMO method adopts as a reference value the factor of 152 hours per PM (Boehm *et al.* 2000), which means 1PM = 152 hours. According to Eq. 3, the TDEV development time was calculated in its nominal schedule definition:

$$TDEV_{NS} = 3.67 * (1257)^{F}$$

where, from Eq. 4, we have:

$$F = 0.28 + 0.2 * (1.1107 - 0.91) = 0.32014.$$

Thus:

$$TDEV_{NS} = 3.67 * (1257)^{0,32014} = 36.05 \text{ months}.$$

However, considering the SCED parameter pointing to 85% of the original schedule, we have new TDEV, now considering the SCED influence:

$$TDEV = 36.05 \text{ months} \times 0.85 = 30.6 \text{ months}.$$

Once PM and TDEV parameters were calculated, it was possible to calculate the average team size:

$$\text{Team size} = PM/TDEV = 1257/30.6 = 41 \text{ people}.$$

The Table 4 summarizes the results of all the main estimated parameters.

Table 4. COCOMO II estimated values.

Parameter	Estimated effort
Thousands Lines of Code	140
PM (person-month)	1257
Time to development (schedule)	30.6 months
Full-time software personnel	41 people

By applying the Rayleigh distribution, the model itself suggests the calculation of the full-time equivalent software personnel (FSP), which is about the allocation of the team over the months of project in an approximate way, as presented in Eq. 5:

$$FSP = PM\left(\frac{0.15 * TDEV + 0,7t}{0.25 * (TDEV)^2}\right) e^{\frac{(0.15*TDEV+0,7t)^2}{0,5*(TDEV)^2}} \quad (5)$$

where FSP means full-time equivalent software personnel; PM means person-month; TDEV means time to development; and t means time in months.

Considering the values already calculated for PM and TDEV and applying the Rayleigh equation for values of "t" from 0 (zero) to 30 (thirty), we finally had the estimated distribution presented in Fig. 2.

The Fig. 2 shows the average size of 41 people, as calculated previously, but suggests a peak period, reaching up to 50 people. This type of early analysis helps project managers to budget for overtime or hiring outsourced workers on the right time for development.

The COCOMO method also foresees the distribution of the team throughout the phases of the project. Using as reference the software development model known as Waterfall methodology, the theoretical distribution of the team would be as presented in Table 5.

The percentages of the "plans and requirements" and "transition" phases are additional to the effort estimated by COCOMO. In this way, the sum exceeds 100%.

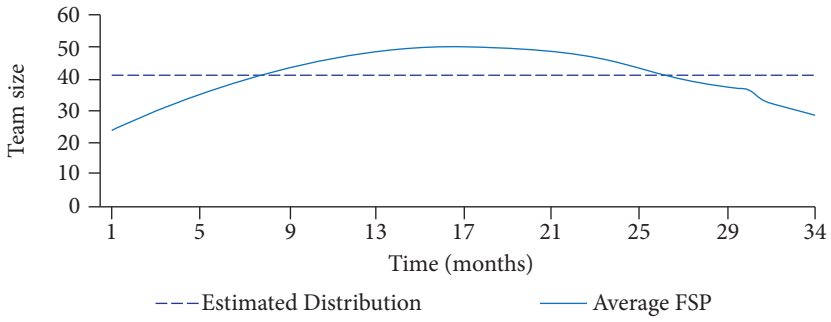

Figure 2. Team average distribution. FSP = full-time equivalent software personnel.

Table 5. Team distribution by phase.

Phase	Effort (%)	Team size
Plan and requirements	7	3
Detailed design	17	7
Coding	52	21
Integration and tests	31	13
Transition1	12	5

It has to be mentioned, however, that the project did not follow a perfect Waterfall model, but a variation in the representation of the waterfall model instead, called the V-model (Pressman 2009), as represented in Fig. 3.

Once the project effort estimate is presented, the next section presents how the project's effort demand was, as well as the variances in the estimate.

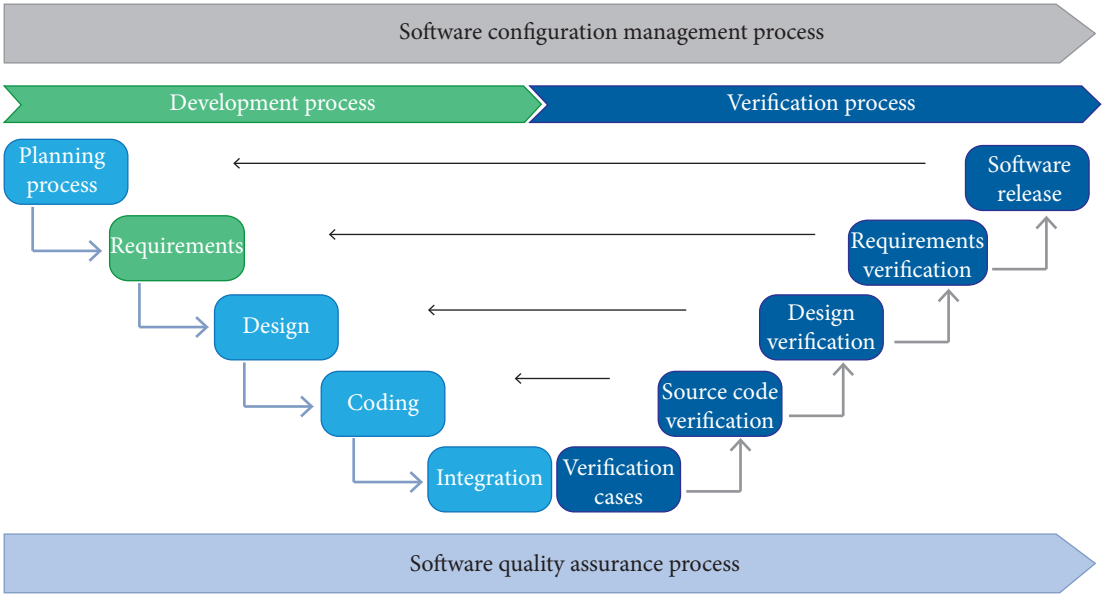

Figure 3. The V-Model adopted by Harpia-H2 Fly-By-Wire development process.

ACTUAL REQUIRED EFFORT

After presenting the estimated effort of the project, this section describes the real effort that was required for the development of the software object of this case study. As presented in the previous section, the COCOMO method uses as reference the software development model known as Waterfall (Boehm *et al.* 2000). The project in question followed the procedure established by standard DO-178C. Thus, before comparing the actual versus the estimated, it is necessary to co-relate the phases of the two development processes, as presented in Table 6, as follows.

Table 6. Waterfall phases and DO-178C processes.

Waterfall phase	DO-178C process
Plan and requirement	Planning requirement
Detailed design	Design
Coding	Coding
Integration and tests	Integration verification
Transition	–

Then, adopting the processes established by DO-178C, the project demanded the following amount of people, according to the Table 7.

Table 7. Actual demanded effort.

DO-178C process	Number of people
Planning	7
Requirement	12
Design	40
Coding	45
Integration	50
Verification	55

Table 7 shows that the peak of number of people occurred in the verification phase, reaching a total of 55 people. The greater need in this process meets the objectives required by the DO-178C; of the 71 objectives, 43 are related to software verification activities, which is the vast majority.

In terms of development time, the project required a total of 4.5 years (54 months) to run through all phases in the life-cycle process, from conception to certification. The Fig. 4 shows graphically the number of people required by the project during the 54 months of development.

After software certification by regulatory agencies, there is a natural fall in the demand for personnel, since the activities are limited to minor fixes or modifications. The maintenance phase of this software is beyond the scope of this case study, so there is no information available on how the personnel was distributed in the later stages.

Considering only the development phase and the number of people from month to month, from the first to the last month of the project, there was a demand for a total of 270.560 hours of development. Adopting the value of 152 hours per PM, it results in a PM value of:

$$PM = 1780 \ (1780 = 270.560/152).$$

The average team size is obtained in the same way as in the estimation method, PM/TDEV. So, we have 1780/54 months, which results in an average value of approximately 32.9 people.

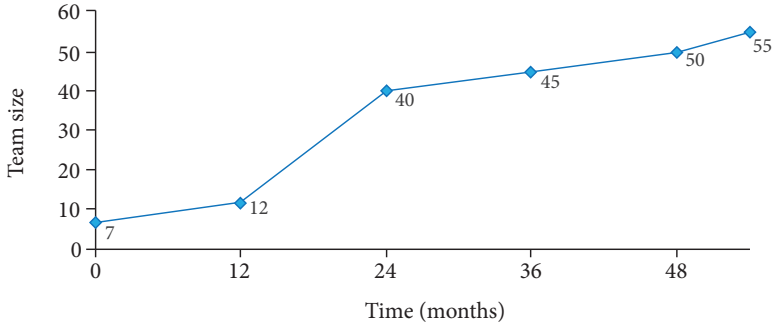

Figure 4. Actual effort during the full life-cycle project.

During the 54 months of the project, a total of approximately 170KLOC was developed and all DO-178C objectives were reached, a relatively close number to the 140KLOC used as input of the COCOMO model in the software size estimation.

Table 8 summarizes the results of all the main parameters of the project.

Table 8. Summary of actual effort.

Parameter	Actual effort
Thousands lines of code	170
PM (person-month)	1780
Time to development (schedule)	54 months
Full-time software personnel	32.9 people

After presenting the estimated effort and the real effort required by the project, the following section presents a discussion of the results obtained with the application of COCOMO II method in the project studied.

DISCUSSION

As adopted by several authors (Jørgensen 2010; Khatibi and Jawawi 2011; Jørgensen and Sjoberg 2001; Basten and Mellis 2011; Grimstad and Jørgensen 2007; Tronto *et al.* 2006; Jørgensen and Shepperd 2007), the magnitude of relative error (MRE) is also adopted in this paper, as a measure to evaluate the accuracy of the estimation model in relation to actual results (Eq. 6). where MRE is the magnitude of relative error.

$$MRE = |actual\ effort - estimated\ effort|/actual\ effort \quad (6)$$

Table 9 presents the values obtained by the estimate and the values actually demanded by the project.

By applying the calculation of MRE to analyze the precision of the estimated effort value (PM), it results as following:

$$MRE = |\ 1780 - 1257|/1780 = 0.40$$

Based on numbers only, it can be observed in Table 9 that the project meets the statistics presented by The CHAOS Report (Standish Group 1995), as well as Moløkken and Jørgensen (2003), because there is clearly a project delay and an over-budget

Table 9. Estimated × actual analysis.

Parameter	Estimated effort	Actual effort
Thousands lines of code	140	170
PM (person-month)	1257	1780
Time to development (schedule)	30.6 months	54 months
Full-time software personnel	41 people	33 people
Team size peak number	50 people	55 people

when compared to the estimated values. In many cases, such difference between the expected 30,6 months and the actual 54 months required for completion could lead the project to failure or even to the collapse of the company. However, it should be considered that the "delay" in the project is due to an over-optimistic estimate, which may lead to a decrease in project quality (de Barcelos Tronto *et al.* 2008).

The MRE calculation shows that the estimate presented a relative error of about 40% in relation to the actual project. Such value, when compared to the results obtained by other projects, can be considered as a poor estimate, since it is not difficult to find in the literature values of MRE below 25% (Khatibi and Jawawi 2011; Jørgensen et al. 2003; Grimstad and Jørgensen 2006). Some techniques of statistical analysis that consider the performance of the estimation of several projects together, such as PRED(25), do not consider in its sample projects with MRE > 25% (Basten and Mellis 2011); in this case, this project would be discarded from the sample.

On the other hand, a value of 40%, as MRE based on COCOMO II method without any type of calibration and not considering the fact that it is a safety-critical application, should not be totally discarded and considered a poor estimation, since in the literature there are also several examples with MRE values in the range of 30% or even > 50% (Velarde *et al.* 2016; Grimstad and Jørgensen 2007; Hughes *et al.* 1998).

Our hypothesis for such estimation error is that the COCOMO II does not address the aspects related to a safety-critical software development that follows the DO-178C processes, especially the number of necessary activities to meet all the 71 objectives for a software level A. Such activities (e.g., requirements review, code review, test results review, structural coverage, etc.) are extremely time consuming and, in some cases, require people independence, which means that the verification activity must be performed by a person other than the developer of the item being verified.

Finally, this case study presents a real use of COCOMO II in a safety-critical software development project following the processes of DO-178C level A. Even considering a reasonable result, the need of an adaptation to the method is evident in order to consider the objectives and specific characteristics of safety-critical projects that must follow the DO-178C.

CONCLUSION AND FUTURE WORK

Finding the root cause for the various fails and delays in software development projects has been the object of study by several researchers in recent years. Properly estimating project costs and deadlines before starting is an important step in the development process that should not be ignored.

In this paper, we present a summary of the main methods of estimating effort in software development projects and the case study of a critical software development aeronautical project that applied COCOMO II in the effort estimation.

The result showed a 40% error in relation to the actual effort, however during the case study it was observed that even an estimate with such error is better than a project without any estimate. The numbers offered by COCOMO II served as a basis for the project management team and a reference to cost and time considered reasonable.

The great contribution of this article is to present the researchers the result of an actual application of the effort estimation method, as well as the factors considered for the selection of each of the levels of the multiplicative factors of cost.

The next step of this research is to make an adaptation of the COCOMO II method so that it is more adherent to the aspects that involve the development of a safety-critical software guided by the processes defined by the DO-178C standard, in order to obtain more accurate estimates for safety-critical software development projects.

AUTHORS' CONTRIBUTION

Both authors contributed equally to the manuscript.

FUNDERS

There are no funders to report for this submission

REFERENCES

Abbas SMA, Liao X, Rehman A, Azam A, Abdullah MI (2012) Cost Estimation: A Survey of Well-known Historic Cost Estimation Techniques. Journal of Emerging Trends in Computing and Information Sciences 3(4):612-636.

Basten D, Mellis W (2011) A Current Assessment of Software Development Effort Estimation. 2011 International Symposium on Empirical Software Engineering and Measurement; Banff, Canada. https://doi.org/10.1109/esem.2011.32

Boehm BW (1981) Software Engineering Economics. New Jersey: Prentice-Hall.

Boehm BW, Abts C, Brown AW, Chulani S, Clark BK, Horowitz E, Madachy R, Reifer DJ, Steece B (2000) Software Cost Estimation with Cocomo II. New Jersey: Prentice Hall.

Boehm BW, Valerdi R (2008) Achievements and Challenges in COCOMO-Based Software Resource Estimation. IEEE Software 25(5):74-83. https://doi.org/10.1109/ms.2008.133

Capaldo G, Iandoli L, Raffa M, Zollo G (2003) The Evaluation of Innovation Capabilities in Small Software Firms: A Methodological Approach. Small Business Economics 21(4):343-354.

CCMI Product Team (2010) CMMI® for Development, Version 1.3. Massachusetts: Software Engineering Institute.

Chesbrough H, Vanhaverbeke W, West J (2006) Open Innovation: Researching a New Paradigm. Oxford: Oup Oxford.

de Barcelos Tronto IF, da Silva JDS, Sant'Anna N (2008) An investigation of artificial neural networks based prediction systems in software project management. J Syst Software 81(3):356-367. https://doi.org/10.1016/j.jss.2007.05.011

Estrada RG, Sasaki G, Dillaber E (2013) Best practices for developing DO-178 compliant software using Model-Based Design. AIAA Infotech@Aerospace (I@A) Conference; Boston, United States of America. https://doi.org/10.2514/6.2013-4566

British Standards Institution – BSI. IEC 62304:2006: medical device software – software life-cycle processes. Brussels, Belgium: BSI, 2006.

Federal Aviation Administration. AC 20-115C: airborne software assurance. Washington, DC, EUA, 2013.

Fifty Sky Shades. 2017. MRJ First Delivery Expected to Be Delayed Again. Japão: Kuznecova K; [accessed 2017 Jan 20]. https://50skyshades.com/news/manufacturer/mrj-first-delivery-expected-to-be-delayed-again.

Finnie GR, Wittig GE, Desharnais J-M (1997) A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models. J Syst Software 39(3):281-289. https://doi.org/10.1016/s0164-1212(97)00055-1

Grimstad S, Jørgensen M (2006) A framework for the analysis of software cost estimation accuracy. Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering – ISESE '06; New York: ACM Press.

Grimstad S, Jørgensen M (2007) Inconsistency of expert judgment-based estimates of software development effort. J Syst Software 80(11):1770-1777. https://doi.org/10.1016/j.jss.2007.03.001

Hughes RT, Cunliffe A, Young-Martos F (1998) Evaluating software development effort model-building techniques for application in a real-time telecommunications environment. IEE Proceedings - Software 145(1):29. https://doi.org/10.1049/ip-sen:19983370

Idri A, Abran A, Kjiri L (2000) COCOMO Cost Model Using Fuzzy Logic. 7th International Conference on Fuzzy Theory & Techniques; New Jersey, United States of America.

Jørgensen M (2014) Communication of software cost estimates. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering – EASE '14. New York: ACM Press.

Jørgensen M (2010) Selection of strategies in judgment-based effort estimation. J Syst Software 83(6):1039-1050. https://doi.org/10.1016/j.jss.2009.12.028

Jørgensen M, Indahl U, Sjøberg D (2003) Software effort estimation by analogy and "regression toward the mean". J Syst Software 68(3):253-262. https://doi.org/10.1016/s0164-1212(03)00066-9

Jørgensen M, Shepperd M (2007) A Systematic Review of Software Development Cost Estimation Studies. IEEE Transactions on Software Engineering. 33(1):33-53. https://doi.org/10.1109/tse.2007.256943

Jørgensen M, Sjoberg DIK (2001) Impact of effort estimates on software project work. Inform Software Tech 43(15):939-948. https://doi.org/10.1016/s0950-5849(01)00203-8

Khatibi V, Jawawi DN (2011) Software Cost Estimation Methods: A Review. Journal of Emerging Trends in Computing and Information Sciences. 2(1):21-29.

Marques J, Marques da Cunha A (2017) Verification scenarios of onboard databases under the RTCA DO-178C and the RTCA DO-200B. 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC). St. Petersburg, United States of America. https://doi.org/10.1109/dasc.2017.8102030

McCormick JW, Singhoff F, Hugues J (2011) Building Parallel, Embedded, and Real-Time Applications with ADA. Cambridge: Cambridge University Press.

Moløkken K, Jørgensen M (2003) A review of software surveys on software effort estimation. Proceedings of the 2003 International Symposium on Empirical Software Engineering – ISESE 2003; Rome, Italy. https://doi.org/10.1109/isese.2003.1237981

National Aeronautics and Space Administration. NASA-STD-8719.13: software safety standard. Washington, D.C., EUA, 2004.

National Aeronautics and Space Administration. NASA-STD-8719.13C: software safety standard. Washington, D.C., EUA, 2013.

Paz A, El Boussaidi G (2016) On the Exploration of Model-Based Support for DO-178C-Compliant Avionics Software Development and Certification. 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW); Ottawa, Canada. https://doi.org/10.1109/issrew.2016.12

Pries KH, Quigley JM (2011) Testing Complex and Embedded Systems. Boca Raton: CRC Press.

RTCA INC. DO-330: software tool qualification considerations. Washington, EUA, 2011.

RTCA INC. DO-331: model-based development and verification supplement to DO-178C and DO-278A. Washington, EUA, 2011.

RTCA INC. DO-332: object-oriented technology and related techniques supplement to ed-12c and ed-109a. Washington, EUA, 2011.

RTCA INC. DO-333: formal methods supplement to DO-178C and DO-278A. Washington, EUA, 2011.

RTCA INC. DO-178C: software considerations in airborne systems and equipment certification. Washington, EUA, 2011

Rierson L (2013) Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance. Boca Raton: CRC Press.

Romani MAS, Lahoz CHN, Yano ET (2010) Identifying dependability requirements for space software systems. J Aerosp Technol Manag 2(3):287-300. https://doi.org/10.5028/jatm.2010.02037810

Sarno R, Sidabutar J, Sarwosri (2015) Improving the Accuracy of COCOMO's Effort Estimation Based on Neural Networks and Fuzzy Logic Model. 2015 International Conference on Information & Communication Technology and Systems (ICTS); Surabaya, Indonesia. https://doi.org/10.1109/icts.2015.7379898

Scaramuzza D, Achtelik MC, Doitsidis L, Friedrich F, Kosmatopoulos E, Martinelli A, Achtelik MW, Chli M, Chatzichristofis S, Kneip L et al (2014) Vision-Controlled Micro Flying Robots: From System Design to Autonomous Navigation and Mapping in GPS-Denied Environments. IEEE Robotics & Automation Magazine 21(3):26-40. https://doi.org/10.1109/mra.2014.2322295

Sheta AF, Aljahdali S (2013) Software Effort Estimation Inspired by COCOMO and FP Models: A Fuzzy Logic Approach. International Journal of Advanced Computer Science and Applications 4(11). https://doi.org/10.14569/ijacsa.2013.041127

The Standish Group. The standish group report: chaos. New York, USA: Project Smart, 2014.

Tronto IFB, Silva JDS, Sant'Anna N (2006) Uma Investigação de Modelos de Estimativas de Esforço em Gerenciamento de Projeto de Software. Simpósio Brasileiro de Engenharia de Software; Florianópolis, Brasil.

Velarde H, Santiesteban C, Garcia A, Casillas J (2016) Software Development Effort Estimation based-on multiple classifier system and Lines of Code. IEEE Latin America Transactions 14(8):3907-3913. https://doi.org/10.1109/tla.2016.7786379

World Intellectual Property Organization. Who filed the most Pct patent applications in 2015? Genebra, Suíça: WIPO, 2015.

Yelisetty SMH, Marques J, Tasinaffo PM (2015) A set of metrics to assess and monitor compliance with RTCA DO-178C. 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC); Prague, Czech Republic. https://doi.org/10.1109/dasc.2015.7311480