Investigating Functional and Code Size Measures for Mobile Applications: A Replicated Study

Filomena Ferrucci[1], Carmine Gravino[1], Pasquale Salza[1], and Federica Sarro[2]

[1] Department of Computer Science, University of Salerno, Italy
{fferrucci, gravino, psalza}@unisa.it
[2] Department of Computer Science, University College London, United Kingdom
f.sarro@ucl.ac.uk

Abstract In this paper we apply a measurement procedure proposed by van Heeringen and van Gorp to approximate the COSMIC size of mobile applications. We compare this procedure with the one introduced by D'Avanzo et al. We also replicate an empirical study recently carried out to assess whether the COSMIC functional size of mobile applications can be used to estimate the size of the final applications in terms of lines of code, number of bytes of the source code and bytecode. The results showed that the COSMIC functional size evaluated with van Heeringen and van Gorp's method was well correlated to all the size measures taken into account. Nevertheless, the prediction accuracy did not satisfy the evaluation criteria and turned out ot be slightly worse than the one obtained in the original study and based on the approach proposed by D'Avanzo et al.

Keywords: Functional Size Measurement, COSMIC, Android Mobile Applications, Code Size Measure, LOC, Empirical Study

1 Introduction

Software sizing is used in software engineering to estimate the size of a software application or component in order to implement other software project management activities, such as effort estimation and productivity benchmarking.

Lines of Code (LOCs) represent one of the most extensively used code size measure and the main input to parametric software cost and effort estimation tools. Nevertheless, LOCs as well as other code size measures are not available early in the development process when effort/cost estimations are needed.

Functional Size Measurement (FSM) methods have been introduced to overcome the limitations of LOCs. They have been widely investigated in software engineering research and also applied in industry for sizing software systems in terms of the functionality provided to the users [14]. The obtained functional size can be used to estimate LOCs using backfiring FSM/LOCs ratios based on earlier projects. Then, the calculated LOCs can be used in the parametric software cost models, e.g., COCOMO [4].

The Function Point Analysis (FPA), was the first FSM method to be introduced in 1979. Several variants have been then proposed (known as 1st generation

FSM methods) to improve the size measurement or extend its application domain. COSMIC [2] is a 2nd generation FSM method, being the first to comply to the standard ISO/IEC14143/1 [14]. It is based on fundamental principles of software engineering and measurement theory, and conceived to be applicable to business, real-time, and infrastructure software (or hybrids of these) [2].

Recent studies have investigated the applicability of 1st and 2nd generation FSM methods to mobile applications [1, 8, 13, 23–26]. This domain is rapidly growing and new software engineering processes, including functional size measurement and estimation methods [23], might be required to improve the quality of these applications. The International Function Point User Groups (IFPUG) has proposed guidelines for the application of IFPUG FPA to mobile applications [24, 25] and some software companies have used them [26]. As for COSMIC, at the best of our knowledge, three proposals to size mobile applications have been reported in the literature [1, 8, 13]. A proposal focused on the measurement of mobile games apps [1], while the other two [8, 13] proposed some guidelines for an approximate and quick sizing of mobile apps. In particular, D'Avanzo et al. [8] focused on apps that use internal data storage, while different basic assumptions where made by van Heeringen and van Gorp [13]. The approach proposed by D'Avanzo et al. has been recently applied by Ferrucci et al. [11] to size 13 Android applications, while no study has been reported in the literature on the use of the other existing appraches so far.

In this paper, we apply the measurement approach proposed by van Heeringen and van Gorp [13] to compute the COSMIC size of 13 mobile Android applications. In particular, we replicate the empirical study proposed by Ferrucci et al. [11] to assess (1) how this size relates to some size measures of the source and compiled code, (2) if it can be used to predict the final application code size in terms of LOCs or number of bytes of the source code and bytecode, (3) which approach between [13] and [8] provides better estimates of code size measures for mobile applications?

It is worth noting that the idea of estimating code size in terms of bytes has been recently proposed by Lind and Heldal [19] that presented a practical approach to estimate the size of compiled C code of embedded applications. Their study highlighted a better correlation between bytecode and COSMIC size with respect to LOCs. They argued that this was due to the fact that the compiler behaves always in the same way by filtering differences in programming style (like 'condensed' programs with many operations per line, few comments vs. only one operation per line, many comments). They also encouraged further studies considering different types of applications and from other domains in order to conclude that bytes can be used as code size measure [18].

To carry out the empirical study we employed the same applications and methodology used in our previous work [11] and the measurement method proposed by van Heeringen and van Gorp [13].

The rest of the paper is organised as follows. Section 2 provides background information on COSMIC. Section 3 introduces and compares the two COSMIC measurement approaches proposed in literature to size mobile applications. Sec-

tion 4 explains the design of the empirical study and reports its results. Section 5 discusses future work and concludes the paper.

2 COSMIC

Functional size is defined as the size of the software derived by quantifying the Functional User Requirements (FURs) [2]. FURs describe what the software is expected to do for its users. COSMIC defines a standardised measure of software functional size expressed in COSMIC Function Point (CFP) units.

A functional process is one of the main concepts underlying COSMIC. It is defined as a set of data movements representing an elementary part of the FURs. A functional user is defined as a (type of) user that is a sender and/or an intended recipient of data in the FURs. Thus, a functional user can be a human or, for instance, an external device as well. A boundary is a conceptual interface between the software being measured and its functional users. With these definitions, it is possible to focus on four different data movement types: an Entry (E) moves data from a functional user to a functional process; an Exit (X) moves data from a functional process to a functional user; a Write (W) moves data from a functional process to persistent storage; a Read (R): moves data from persistent storage to a functional process.

1 CFP unit is given per each data movement and their sum represents the measured size. COSMIC defines a measurement process that consists of three phases: the Measurement Strategy Phase, the Mapping Phase, and the Measurement Phase. Each of them is explained in the following.

Measurement Strategy Pattern is a concept introduced in the last current version (4.0) of COSMIC [2]. Previous versions do not refer to any strategy patterns. The Measurement Strategy Phase sets the key parameters of the measurement: the purpose, defining what the measurement result will be used for; the scope defining which pieces of software (in terms of FURs) have to be measured; the level of granularity which describes how much detailed the documentation about the software is (e.g., in terms of the requirements description or also the structure description). The complete list of parameters can be found in the COSMIC Context Software Model and it is necessary to carefully define them.

In the *Mapping phase* the measurer extrapolates the functional processes from the available FURs of the software being measured. This is a technical work in which the principles and, above all, the rules of the COSMIC method (reported in the COSMIC Generic Software Model [2]) have to be carefully followed. The measurer identifies the potential functional processes inside the FURs by looking for each functional process that is started by a triggering Entry and comprises at least two data movements: an Entry plus either an Exit or a Write. The triggering Entry is the Entry of the functional user that starts the functional process. Data manipulations inside a functional process are not counted as CFP [2], i.e., they are considered associated to the corresponding data movements. The object of interest is defined as any 'thing' that is identified from the point of view of the

FURs; it may be any physical thing, as well as any conceptual object or part of a conceptual object in the world of the functional user about which the software is required to process and/or store data. Each Entry, Exit, Read, or Write is a movement of data group of a single object of interest. There are only two exceptions: the triggering Entry which can start a functional process without data movement, e.g., in specific enquiry for a list of items; the error/confirmation message which is defined as an Exit for the attention of a human user that either confirms only that entered data is accepted, or only that there is an error in the entered data.

The *The Measurement Phase* defines how to count data movements, consisting in associating a CFP to each data movement. The amount of all data movements represents the functional value of the measurement. It is worth noting that in cases (differently from our work) of aggregating measurement sizes (software stratified into different layers) or when measuring the size of software changes, this phase may become more complex [2].

The COSMIC community has also proposed approaches for counting the size of software in terms of COSMIC by exploiting approximate countings. There are a couple of situations when a need arises in practice to measure a functional size approximately [3]: it can happen either early in the life of a project before the Functional User Requirements (FURs) have been specified in detail ('early sizing'), or when a measurement is needed but there is insufficient time or resources to apply the standard detailed method ('rapid sizing'). These motivations are not mutually exclusive and contribute to reach a trade-off between a correct measurement and time and budget available.

There exist some proposals for a quick and approximate COSMIC sizing (e.g., [3,9,10]). The use of COSMIC approximate methods has also been suggested recently in the context of mobile apps [8, 13], with the aim of helping measurers to size mobile apps in a fast and accurate way.

3 Towards Applying COSMIC to Mobile Applications

In this section we summarize the approach proposed by van Heeringen and van Gorp to size mobile apps in terms of COSMIC. We also compare this approach with the one presented by D'Avanzo et al. [8] and experimented in our previous work [11] for code size estimation purposes. The comparison is made in this section in terms of the guidelines of the two approaches applied on a set of functional requirements of a mobile application.

3.1 van Heeringen and van Gorp Approach

The approach proposed by van Heeringen and van Gorp [13] is based on a set of assumptions: i) A mobile app is considered to be an application layer that is developed on top of one or more data layers; ii) No persistent data is stored in the application layer because they consider storing data on the device as a technical solution. Thus, no Reads and Writes are expected; iii) Because mobile apps are

considered to be business applications, they include possible error messages in every functional process as 1 Exit, together with 1 Entry when the messages come from a data layer.

The proposed measurement phase consists in identifying the type of each functional process and quantifying the parameters involved for the identified type of functional process. Five different functional process types ($\mathcal{A}1$–$\mathcal{A}5$) can be identified as follows.

$\mathcal{A}1$. View Functionality. Data is presented to at least one of the functional users, with a minimum of 6 COSMIC Functional Points (CFPs): 1 triggering (start) Entry; 1 question for information Exit to the data layer; a couple of 1 Entry and 1 Exit for reception and show of data respectively; a couple of 1 Entry and 1 Exit for reception and show of error messages. The ability of changing the view of the displayed data does not afflict any data movement but other additional data groups add 2 CFPs (1 Entry to receive data and 1 Exit to show). In this case, a separate data movement for the question for information to the data layer is not necessary because it can be included in the previous one. For each data group which shows calculated or derived data, another 1 Exit for showing data is added to the functional size.

$\mathcal{A}2$. Data Manipulation Functionality. It is used to manipulate information (add/change/delete) about a data layer, with a minimum of 4 CFPs: 1 start Entry; 1 Exit to provide information to data layer; a couple of 1 Entry and 1 Exit for reception and show of error messages. Further 2 CFPs for any other data group manipulated are considered (1 Entry for entering data by the user and 1 Exit to provide them to the data layer). When the manipulated data is also shown to the functional user, the size is increased of 3 CFPs: 1 question Exit to data layer, 1 Entry for receiving data and 1 Exit for data showing. Furthermore, if there is a validation process involving the data layer, 1 Entry for questioning the data layer and 1 Exit for receiving data are also added.

$\mathcal{A}3$. Enquiry Functionality. It shows the data that can be manipulated and it is considered equal to the view functionality ($\mathcal{A}1$).

$\mathcal{A}4$. User Supporting Functionality. If it is mandatory in order to complete a manipulation process, it is reduced to $\mathcal{A}2$. On the other hand, when it can be avoided but the user can exploit it, e.g. a calendar view to simplify the insertion of a date, it is considered as a separate functionality and it is reduced to $\mathcal{A}1$.

$\mathcal{A}5$. Special Functionality. It includes some special cases:

- Dynamically generated menus, when the view of the menu depends on the information from the data layer. $\mathcal{A}1$ is applied;
- Log in, with a total of 5 CFPs: 1 start Entry, 1 Exit for credentials providing to the data layer, 1 Entry for log status and a couple Exit/Entry for error messages. The log out functionality has only a couple of Entry/Exit for messages showed from the application layer. If the log out is recorded into the data layer, another 1 Exit to provide credentials and 1 Entry to receive error messages are expected;
- Help functionality, $\mathcal{A}1$ is applied;
- Invoking External Functionality, it is outside the scope of the app being measured. Therefore, no CFPs are considered.

3.2 D'Avanzo et al. Approach

D'Avanzo et al. [8] propose guidelines to measure business mobile applications where the persistent storage is considered as an internal database, so it is accessible by Read and Write data movements. Moreover, differently from van Heeringen and van Gorp [13], in the case of external applications, these guidelines consider only the data movements between the application being measured and the external one. The guidelines can be summarised as follow:

$\mathcal{B}1$. Open the application and see info on home screen. Data movements are: 1 triggering Entry, by opening the application; 1 Read from persistent storage; 1 Exit to show data. If other data is expected an additional 1 Read and 1 Exit are counted, or only 1 Exit is counted in case the data shown is the result of a calculation.

$\mathcal{B}2$. See details. Same as $\mathcal{B}1$ except for the additional data movements.

$\mathcal{B}3$. Create/set/delete data. Data movements are: 1 triggering Entry; 1 Write to persistent storage; 1 Exit for error/confirmation messages.

$\mathcal{B}4$. Update data. It is a combination of the functional process to enquiry the data ($\mathcal{B}2$) and the one to update them ($\mathcal{B}3$), with at least 6 CFPs expected.

$\mathcal{B}5$. Process input and stored data to provide an output. Data movements are: 1 triggering Entry; 1 Read from persistent storage; 1 Exit to show the result. Further couples of Entry/Exit are added for each further data visualisation, not read from the database, and 1 Exit if error/confirmation messages are expected.

$\mathcal{B}6$. Share data with an external application. Data movements are: 1 triggering Entry; 1 Read from persistent storage; 1 Exit towards external application. Any possible error/confirmation messages are considered associated to the external application functional process and they are not measured.

$\mathcal{B}7$. Import data from an SD card to the database. Data movements are: 1 triggering Entry; 1 Entry from an SD card; 1 Exit for error/confirmation messages; 1 Write to persistent storage; 1 Exit for error/confirmation messages. Another Exit is counted if the data is also shown to the user.

$\mathcal{B}8$. Export data from the database to an SD card. Data movements are: 1 triggering Entry; 1 Read from persistent storage; 1 Exit to SD card; 1 Exit for error/confirmation messages. Another Exit if the data is shown to the user.

3.3 Example of Application and Discussion

To better understand the differences between the two considered approaches, an example of application is given. We considered an app realising an academic transcript manageable by the user, whose FURs are described in Table 1. From here on we will refer to the van Heeringen and van Gorp approach as $\mathcal{A}$, while $\mathcal{B}$ will denote the D'Avanzo et al. approach.

Since R1 is an opening functionality which lies on data visualisation, $\mathcal{A}$ suggests to apply rule $\mathcal{A}1$ with 6 basic CFPs and 1 additional Exit for calculated and shown statistic data, with a total of 7 CFPs. $\mathcal{B}$ applies rule $\mathcal{B}1$ with 4 CFPs, which also include 1 Exit for calculated data. The main difference in this case regards the different point of views about the role of the persistent storage,

Table 1. Functional User Requirements.

FUR	Description
R1	User opens the application to see on home screen the principal info R1 included in his transcript, i.e., the list of exams, the number of exams, the number of credits and the average mark.
R2	User clicks on the icon button 'new' to insert data about a new exam in the database. The system provides error/confirmation messages.
R3	User selects an exam from the list in the home screen and clicks on the R3 button 'delete' to delete it from the database. The system provides error/confirmation messages.
R4	User clicks on the button 'delete all' to delete all the exams data from the database. The system provides error/confirmation messages.
R5	User selects an exam from the list in the home screen and clicks on the R5 button 'update' to update its data in the database. The system provides error/confirmation messages.
R6	User selects an exam from the list in the home screen and clicks on the button 'details' to see detailed info.
R7	User clicks on the icon button 'projection average' and the system shows a new box containing the current average mark and a form to specify the number of future exams, their credits and the expected mark. The system provides the expected average mark given by the input data values and the current exams.
R8	User clicks on the button 'export exams' to export exams from database to SD. The system provides error/confirmation messages.
R9	User clicks on the button 'import exams' to import exams from SD to the database. The system provides two error/confirmation messages, one for the input from SD and another for the writing on the database. The system shows the list of exams after importing.
R10	User sets the lode value for the statistics ($30 + 0$, $30 + 1$ etc.). The system provides error/confirmation messages.
R11	User sets maximum credits value. The system provides error/confirmation messages.
R12	User clicks on the button to read change log.
R13	User clicks on the button to read FAQ.
R14	User clicks on the button to read application license.
R15	User clicks on the button to read info to donate a payment to the developer.

considered as a distinct functional user in $\mathcal{A}$ and as belonging to the application boundary in $\mathcal{B}$. This means that every Read in $\mathcal{B}$ matches a couple Exit/Entry in $\mathcal{A}$. Moreover, $\mathcal{A}$ expects always the presence of error messages movements, then a couple Entry/Exit is needed to read/show the messages.

R2, R3, R4, R10, R11 refer to data manipulation functionality, then $\mathcal{A}$ applies rule $\mathcal{A}2$ with a total of 4 CFPs while $\mathcal{B}3$ is applied for $\mathcal{B}$ obtaining 3 CFPs. Thus, $\mathcal{A}$ and $\mathcal{B}$ behave in a similar ways because 1 Write of $\mathcal{B}$ matches 1 Exit of $\mathcal{A}$ towards persistent storage and 1 Exit to show error messages is considered for both. $\mathcal{A}$ includes an additional Entry to read error messages from data layer.

$\mathcal{A}$ and $\mathcal{B}$ interpret R5 in different ways. The FUR is an update activity based on data already present into the persistent storage. $\mathcal{A}$ applies rule $\mathcal{A}2$ for data manipulation. On the other hand, $\mathcal{B}$ applies $\mathcal{B}4$ which is a mix of $\mathcal{B}2$ and $\mathcal{B}3$, relying on the fact that the data has to be read before writing the modification. This difference brings $\mathcal{A}$ to count 4 CFPs against the 6 CFPs obtained with $\mathcal{B}$.

R6 is treated as a visualisation functionality in both the approaches. Because there is just data extrapolation from persistent storage, without any calculation, $\mathcal{A}$ applies rule $\mathcal{A}1$ with a score of 6 CFPs. $\mathcal{B}$ identifies a specific rule (i.e., $\mathcal{B}2$) for this kind of FUR, where the main intent is to show details about a selected item, with a score of 3.

In R7, the data are first shown after user's triggering button. Thus, $\mathcal{A}$ requires to apply rule $\mathcal{A}1$ with a score of 6 CFPs. After this, the user can edit 3 different boxes and enquiry the persistent storage on any change. Rule $\mathcal{A}1$ states that for each new enquiry, other 2 CFPs has to be considered. This leads to a total of 12 CFPs. In the case of $\mathcal{B}$, rule $\mathcal{B}5$ is applied adding 3 Entry/Read couples to a base of 4 CFPs, with a total of 9.

R8 and R9 are respectively the FURs for exporting to and importing from the SD card. In $\mathcal{A}$, two different rules are applied: view functionality ($\mathcal{A}1$) and manipulation data functionality ($\mathcal{A}2$). The data is moved from persistent storage to the SD card and vice versa by first reading (i.e., $\mathcal{A}1$), with a total of 5 CFPs. Then, $\mathcal{A}2$ provides the writing functionality with a score of 3 CFPs, excluding the starting Entry expected for the rule. In the case of import, R9 expects data to be also shown on the screen. So, 9 CFPs are counted for R9 and 8 CFPs for R8. As for data direction, for $\mathcal{B}$ there are two different rules: for R8 is applied rule $\mathcal{B}8$ obtaining 4 CFPs while for R9 6 CFPs are obtained by applying rule $\mathcal{B}7$ and considering an extra Exit to show data on the screen.

R12, R13, R14 and R15 belong to the type of help functionalities which are considered equivalent to the view functionality. Thus, $\mathcal{A}$ applies rule $\mathcal{A}5$ (equal to $\mathcal{A}1$ in this case) with a total of 6 CFPs. $\mathcal{B}$ applies $\mathcal{B}2$ with a score of 3 CFPs.

3.4 Discussion

Taking into account the measurement performed on a set of 13 mobile apps, including the above-mentioned example, we were able to perform a comparison of the approach $\mathcal{A}$ and $\mathcal{B}$ in terms of guideline applicability. The achieved outcomes and some insights originated from the analysis are summarised in the following.

$\mathcal{A}$ defines the boundary of the application layer in a more strict way than $\mathcal{B}$: the persistent storage is considered as a separate functional user and any interaction with it is measured as Entries or Exits rather than Writes or Reads as done by $\mathcal{B}$. This is reflected also on the constant presence of error messages in $\mathcal{A}$. In our opinion, persistent storage needs a proper definition and it might be associated to every simple data structure as key/value maps or stored files. In that case, the use of Read/Write, as $\mathcal{B}$ does, could be more appropriate. In the presence of more complex structures as SQLite database, requiring a standardised SQL communication, even if it is stored as a normal file, considering them as separate data layers (as done in $\mathcal{A}$) matches better.

$\mathcal{A}$ expects the presence of error messages in any rule. As far as we are concerned, it regards also confirmation messages, but only for communication with the data layer and not with the persistent storage. However, possible error messages due to the persistent storage are external to the functional processes since they are generally handled by the mobile operating system.

Moreover, $\mathcal{A}$ lacks of any interaction between different data layers. For instance, in many cases we observed some operations of import/export data from/to an SD card. While $\mathcal{B}$ has its proper guidelines facing this kind of situation, to cover this case with $\mathcal{A}$ a combination of rules has to be considered with a risk of introducing redundant data movements. On the other hand, $\mathcal{B}$ does not consider communication with a remote data storage (e.g., a web service). Even if not explicitly mentioned, in $\mathcal{A}$ this can be seen as a data layer. Communication with a remote data storage is generally included in mobile apps, thanks to the mobility feature of devices, so it should be carefully considered in the data layer way.

Both $\mathcal{A}$ and $\mathcal{B}$ also address the situation in which there is an interaction with an external application. $\mathcal{A}$ considers it as an external application. However, excluding any data movement from the application being measured might be misleading. Let us consider a generic app which shares a list of items with an external e-mail app. Because the e-mail app handles onyl textual input, the shared data needs to be converted in a text form in the first place. The conversion as well as sending the data to the e-mail app are in charge of the generic app itself. The example and the derived discussion suggest that some effort is needed to elaborate a more flexible approach for measuring the functional size of mobile apps. In particular, the simplicity of $\mathcal{A}$ in the definition of its rules and the readiness of $\mathcal{B}$ with persistent storage interaction might inspire a future and improved version of the measurement process.

4 Empirical Study

In this section we present a replication of our previous work [11] where we empirically analysed whether the COSMIC functional size of mobile applications obtained by following the guidelines provided by D'Avanzo et al. (approach $\mathcal{B}$) [8] can be exploited to estimate the size of the final applications in terms of lines of code and number of bytes of the source code and bytecode. Differently from the original study, in this replication we have exploited the measurement approach

proposed by van Heeringen and van Gorp (approach $\mathcal{A}$) [13] to size the mobile apps in terms of COSMIC. In the following we first describe the design of the replication and then the results we achieved. With the aim of the comparison with the original study, we also reported the results we achieved in our previous work [11] (approach $\mathcal{B}$).

4.1 Design

As in the original study, the research questions we investigate are:

- **RQ1:** Does COSMIC measure relate with code size measures for mobile applications?
- **RQ2:** Can COSMIC measure be used to estimate code size measures for mobile applications?

Since we want to compare the results achieved by employing the functional sizes obtained with the two measurement approaches proposed in [13] (i.e., $\mathcal{A}$) and [8] (i.e., $\mathcal{B}$), in the present study we also investigate a new research question:

- **RQ3:** Which approach between $\mathcal{A}$ and $\mathcal{B}$ provides better estimates of code size measures for mobile applications?

In the following, we provide details about the employed data set, the estimation technique, the validation method, and the evaluation criteria. Threats that could affect the validity of the empirical study are also discussed.

Data Set and Variables. We have employed a data set including information[3] on 13 Android mobile applications randomly downloaded from Google Play Store. For each application one of the authors collected the requirements in a Functional User Requirements (FURs) document. Then, following the guidelines proposed by van Heeringen and van Gorp [13], COSMIC was applied to the FURs document obtaining the results reported in Table 2. In particular, $\text{CFP}_{\mathcal{A}}$ is the independent variable of our study denoting the number of COSMIC Function Points (CFP) obtained by applying the the guidelines proposed by van Heeringen and van Gorp [13]. $\text{CFP}_{\mathcal{B}}$ denotes the number of CFP obtained by applying the guidelines proposed by D'Avanzo et al. [8].

Table 2 also shows the information about the 13 mobile application code sizes. Since in Android the Java code is mainly used to develop functionalities, while XML is used to design the user interfaces, we considered both Java and XML size, to analyse apart the components constituting the interface of the mobile apps. We did not measure any other raw asset files, such as images or local database files, because they do not directly relate to the application functionalities. We considered each Java class involved (except for external libraries) and only the XML layouts that are needed to visualize the application. For example, we discarded those XML files that describe additional resolutions for other devices,

[3] Requirements and CFPs data are publicly available on https://goo.gl/Nj6mAO

Table 2. Descriptive statistics of mobile app code sizes and COSMIC sizes.

Descriptive statistics	Java		XML		Bytecode	$CFP_{\mathcal{A}}$	$CFP_{\mathcal{B}}$
	kB	LOC	kB	LOC	kB		
Min	19.00	473.00	12.00	167.00	23663.00	18.00	15.00
Max	322.00	7514.00	94.00	1695.00	272 775.00	220.00	145.00
Mean	122.50	2786.60	42.30	590.70	111 892.30	70.62	49.30
Median	91.50	2295.50	38.00	487.50	94 298.50	58.00	40.00
Std Dev	96.85	2157.30	23.20	427.40	73 823.36	51.79	35.03

such as tablets, since these are optimisations and do not describe functionalities. As for the LOC, the variables Java_{LOC}, XML_{LOC}, and $\text{Total}_{\text{LOC}}$ represent the lines of code for the Java code, XML code, and their sum, respectively. As for number of bytes, we considered both the source and the compiled code. The variables Java_{kB}, XML_{kB}, and Total_{kB} represent the Java size, XML size, and their sum in terms of kilobytes, respectively. The variable $\text{Bytecode}_{\text{kB}}$ denotes the size of the compiled code in terms of kilobytes. These variables represent the dependent variables of our empirical study. The information about these variable were collected by using the apps APK files downloaded from the official store. The code size was measured with the 'du' UNIX command and the lines of code with the 'CLOC' tool.

Correlation Test and Estimation Technique. To assess (RQ1) the relationship among the independent variable (i.e., $CFP_{\mathcal{A}}$) and the dependent variables (i.e., Java_{kB}, Java_{LOC}, XML_{kB}, XML_{LOC}, Total_{kB}, $\text{Total}_{\text{LOC}}$, and $\text{Bytecode}_{\text{kB}}$) we applied the nonparametric association statistics Spearman's rho [12], which is widely employed in the literature. This statistic ranges from $+1$ to -1, where $+1$ indicates perfect correlation and -1 indicates a perfect inverse correlation, while 0 indicates no correlation.

To verify whether or not the functional size of a mobile application can be exploited to predict the corresponding code size (RQ2), expressed both in terms of bytes and lines of code, we built a prediction model for each dependent variable (e.g., $\text{Total}_{\text{LOC}}$) using $CFP_{\mathcal{A}}$ as independent variable, by applying the Linear Regression (LR) analysis. To evaluate the goodness of fit of a regression model, we exploited the square of the linear correlation coefficient, R^2, that shows the amount of the variance of the dependent variable explained by the model related to the independent variable. Other useful indicators are the F value and the corresponding p-value (denoted by Sign F), which high and low values, respectively, denote a high degree of confidence for the prediction.

To answer RQ3, i.e., to compare the accuracy of $CFP_{\mathcal{A}}$ and $CFP_{\mathcal{B}}$ in predicting code sizes, we employed the results of the original study [11].

Validation Method and Evaluation Criteria. To validate the built estimation models we carried out a cross validation, meaning that the original data set was divided into different subsets of training and validation sets. Training sets

were used to build models with LR and validation sets were used to validate the obtained models. In particular, we applied a leave-one-out cross validation, i.e., the original data set was divided into $n = 13$ different subsets of training and validation sets, where each validation set has one observation.

As for evaluation criteria, we applied the Spearman's rho test to verify whether the predicted size is a useful estimation of the actual size. Furthermore, we employed some summary measures to assess the accuracy of the obtained estimations, namely MMRE, MdMRE and Pred (l) [7], which have been widely used in empirical studies similar to ours (see e.g., [16]). In the context of effort estimation, where these measures were proposed [7], l is widely set to 0.25 and a good estimation model should have a MMRE ≤ 0.25 and Pred $(0.25) \geq 0.75$, that is, the mean estimation error should be less than 25 %, and at least 75 % of the estimated values should fall within 25 % of their actual values [7]. In this study we used $l = 0.25$. In the future, we will further analyse this point. We employed summary measures also to compare the results achieved herein with those obtained in the original study. Moreover, we tested the statistical significance of the results by using absolute residuals, i.e., to establish whether one approach provided significantly better code size estimations than the other employed [15]. Absolute Residuals (AR) is defined as $|Actual - Predicted|$, where *Actual* is the actual code size (e.g., Java_{LOC}) and *Predicted* is the estimated code size. In particular, we performed the Wilcoxon signed rank test [6] to verify the following null hypothesis 'the two considered populations of absolute residuals have identical distributions'. For all the statistical tests, we accept a probability of 5 % of committing a Type-I-Error [12].

Threats to Validity. Reliability of the data and lack of standardisation should be taken into account for the internal validity [17, 20]. We did our best to collect information in a uniform fashion. The construct validity can be biased by the collection of the information used to determine the size measures. The measurement task of the functional size is crucial. One of the authors, with previous experiences in measuring software in terms of COSMIC, performed the measurement task. Another author cross-checked the information obtained. As for the measurement of the code sizes, we manually inspect Java classes and XML files to remove noisy content (e.g., third part libraries). As for the conclusion validity, we carefully applied the estimation method and the statistical tests, verifying all the required assumptions. Another threat to conclusion validity could be the few number of applications composing the data set. However, observe that 'a rule of thumb in regression analysis is that 5 to 10 observations are required for every variable in the model' [22]. Furthermore, this kind of studies can contribute to provide useful indications that can be further validated in subsequent studies.

4.2 Results and Discussion

The results of the Spearman's rho test revealed that all the considered code size measures were positively associated with the independent variable $\text{CFP}_{\mathcal{A}}$, with a

statistics greater than 0.8. Furthermore, the results of the performed tests were statistical significant as the p-values of the statistics were less than 0.05. This means that when the value of the $CFP_{\mathcal{A}}$ increases, the value of the code size measures (e.g., $Java_{kB}$) increases as well. The dependent variables having the highest association with independent variable were XML_{LOC} and $Total_{kB}$ that were characterised by a statistics grater than 0.9.

According to these results we can positively answer RQ1: for the considered mobile apps COSMIC sizes obtained with the measurement approach A well relates to the considered code size measures.

To answer RQ2 we built size estimation models by exploiting LR. To this aim, we first verified the assumptions underlying its application: linearity (i.e., the existence of a linear relationship between the independent variable and the dependent variable); homoscedasticity (i.e., the constant variance of the error terms for all the values of the independent variable); residual normality (i.e., the normal distribution of the error terms), and residual uncorrelation (i.e., error terms are uncorrelated for consecutive observations).

It is worth noting that we also verified the presence of influential observations (i.e., extreme values which might influence the models obtained from the regression analysis) by using the residuals plot and Cook's distance and performing a stability analysis as suggested by Mendes and Kitchenham [21]. According to this analysis no transformation of the original data was performed and no observation was removed.

Table 3. Results of the linear regression using $CFP_{\mathcal{A}}$ as dependent variable.

Dependent Variable	R^2	F	Sign. F (p-value)
$Java_{kB}$	0.777	38.22	< 0.001
$Java_{LOC}$	0.759	40.35	< 0.001
XML_{kB}	0.895	94.13	< 0.001
XML_{LOC}	0.840	57.86	< 0.001
$Total_{kB}$	0.896	94.78	< 0.001
$Total_{LOC}$	0.845	60.07	< 0.001
$Bytecode_{kB}$	0.813	47.82	< 0.001

Table 3 shows the results of the LR analysis. We can observe that the models are characterised by a high R^2 value, i.e., greater than 0.8 except for the models having $Java_{LOC}$ and $Java_{kB}$ as dependent variables for which the value is very close to 0.8 (i.e., 0.777 and 0.759, respectively). Furthermore, a high F value and a p-value (Sign. F) less than 0.001 were obtained, indicating that the prediction is possible with a high degree of confidence.

We have performed a leave-one-out cross validation to evaluate the accuracy of the obtained estimates of the code sizes with respect to the actual code sizes.

Again, we applied a Spearman's rho test to establish whether the predicted size is a useful estimation of the actual size.

The results revealed that the predicted size is always statistically positively correlated with the actual size for all the considered code size measures, with statistics greater than 0.8, except for XML_{kB}. Thus, the obtained size predictions can provide a good indication of the actual sizes. To quantify the accuracy of the obtained estimates, we computed the summary measures MMRE, MdMRE, and Pred (0.25) (see Table 4, results using $CFP_{\mathcal{A}}$). We can observe that no model was characterised by values satisfying the thresholds of Conte et al. [7]. The best result in terms of summary measures was obtained with $Bytecode_{kB}$, having MdMRE equals to 0.25 and Pred (0.25) equals to 0.54. On the other hand, the worst result was obtained for the models predicting $Java_{kB}$. Thus, as answer to RQ2 we can state that $CFP_{\mathcal{A}}$ did not provide quite good estimations of the source code sizes for the considered mobile apps.

Table 4. Results of cross validation in terms of summary measures.

Dependent Variable	Using CFP_A			Using CFP_B		
	MMRE	MdMRE	Pred (0.25)	MMRE	MdMRE	Pred (0.25)
$Java_{kB}$	0.61	0.36	0.31	0.46	0.46	0.54
$Java_{LOC}$	0.56	0.30	0.46	0.43	0.24	0.54
XML_{kB}	0.38	0.34	0.31	0.43	0.29	0.31
XML_{LOC}	0.31	0.28	0.38	0.32	0.28	0.46
$Total_{kB}$	0.36	0.25	0.46	0.29	0.19	0.77
$Total_{LOC}$	0.46	0.27	0.46	0.35	0.20	0.62
$Bytecode_{kB}$	0.44	0.23	0.54	0.33	0.21	0.54

We also compared the results achieved in the replication study presented here with those obtained in the original study. For this reason in Table 4 we also reported results obtained using $CFP_{\mathcal{B}}$. The comparison suggested that better code size predictions were obtained with $CFP_{\mathcal{B}}$, for all the considered code size measures. However, the performed Wilcoxon test revealed that the differences were not statistically significant. To conclude, we can answer RQ3 saying that $\mathcal{B}$ [8] provided slightly better (not statistically significant) code size estimations than $\mathcal{A}$ [13].

5 Conclusions and Future Work

In this paper we presented a replication of our previos study [11] that exploited functional sizes in terms of COSMIC to estimate code size measures about 13 mobile applications. The two studies differ in the independent variable, representing the functional size in terms of COSMIC, since two different measurement approaches have been employed. In the replication we applied the approach proposed by van Heeringen and van Gorp [13], while the original study used the approach proposed by D'Avanzo et al. [8]. On the other hand, the original and

the replication studies share the research questions RQ1 and RQ2, the set of mobile applications, the set of dependent variables, the estimation technique, the validation method, and the evaluation criteria. In the replication we considered a further research question (RQ3) to investigate which of the analysed approaches provided better code size estimations.

The results of the replication highlighted that, for the considered mobile apps, the COSMIC functional size was well correlated to all the size measures taken into account, thus, confirming the results of the original study and the findings by Lind and Heldal [18, 19]. Nevertheless, the prediction accuracy did not satify the evaluation criteria and turned out to be slightly worse than the one obtained in the original study based on the approach proposed by D'Avanzo et al. [8].

The results of our study should encourage the use of functional size measurement methods, in particular COSMIC, to size mobile applications and employ the obtained measure for implementing other software project management activities, such as effort estimation and productivity benchmarking.

In the future we intend to apply the considered approaches on larger data sets and different kinds of mobile applications to confirm/contradict the results obtained so far also to understand the reasons for the low prediction accuracy. In particular we would like to analyse if it depends on the approximated methods of COSMIC measurement or on other factors. We will also investigate whether a unique/unified approach is suitable to measure different kinds of apps, or different approaches are needed. The contribution of single BFCs could be also investigated [5] to provide a better comparison with the approach proposed by van Heeringen and van Gorp [13].

Finally, the collection of effort data could be useful to derive effort/cost estimation models.

References

1. Abdullah, N.A.S., Rusli, N.I.A., Ibrahim, M.F.: Mobile game size estimation: COSMIC FSM rules, uml mapping model and unity3d game engine. In: 25th IEEE Conference on Open Systems (ICOS). pp. 42–47 (2014)
2. Abran, A., Baklizky, D., Desharnais, J., Fagg, P., Gencel, C., Symons, C., Jayakumar, K.R., Lesterhuis, A., Londeix, B., Nagano, S.I., Santillo, L., Soubra, H., Trudel, S., Vogelezang, F., Woddward, C.: The COSMIC Functional Size Measurement Method, Measurement Manual, Version 4.0.1 (2015)
3. Abran, A., Londeix, B., O'Neill, M., Santillo, L., Vogelezang, F., Desharnais, J., Morris, P., Rollo, T., Symons, C., Lesterhuis, A., Oligny, S., Rule, G., Toivonen, H.: The COSMIC Functional Size Measurement Method, Advanced and Related Topics, Version 3.0 (2007)
4. Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B.: Software Cost Estimation with COCOMO II. Prentice Hall Press (2009)
5. Buglione, L., Gencel, C.: Impact of base functional component types on software functional size based effort estimation. In: Product-Focused Software Process Improvement, Lecture Notes in Computer Science, vol. 5089, pp. 75–89. Springer Berlin Heidelberg (2008)

6. Conover, W.J.: Practical Nonparametric Statistics. Wiley, 3rd edition edn. (1998)
7. Conte, S.D., Dunsmore, H.E., Shen, V.Y.: Software Engineering Metrics and Models. Benjamin-Cummings Publishing Co., Inc. (1986)
8. D'Avanzo, L., Ferrucci, F., Gravino, C., Salza, P.: Cosmic functional measurement of mobile applications and code size estimation. In: 30th ACM/SIGAPP Symposium on Applied Computing (SAC). pp. 1631–1636 (2015)
9. De Marco, L., Ferrucci, F., Gravino, C.: Approximate cosmic size to early estimate web application development effort. In: 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). pp. 349–356 (2013)
10. De Vito, G., Ferrucci, F.: Approximate cosmic size: The quick/early method. In: 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). pp. 69–76 (2014)
11. Ferrucci, F., Gravino, C., Salza, P., Sarro, F.: Investigating functional and code size measures for mobile applications. In: 41st EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). pp. 365–368 (2015)
12. Gibbons, J.D.: Nonparametric statistical inference. Marcel Dekker Inc. (1986)
13. van Heeringen, H., van Gorp, E.: Measure the functional size of a mobile app: Using the cosmic functional size measurement method. In: 24th International Workshop on Software Measurement (IWSM) and the 9th International Conference on Software Process and Product Measurement (MENSURA). pp. 11–16 (2014)
14. ISO/IEC: ISO/IEC 14143-1:2007: Information technology - Software measurement - Functional size measurement - Part 1: Definition of concepts (2007)
15. Kitchenham, B., Pickard, L., MacDonell, S., Shepperd, M.: What accuracy statistics really measure. IEE Proceedings Software 148(3), 81–85 (2001)
16. Kitchenham, B., Mendes, E.: Software productivity measurement using multiple size measures. IEEE Transactions on Software Engineering 30(12), 1023–1035 (2004)
17. Kitchenham, B., Pickard, L., Pfleeger, S.L.: Case studies for method and tool evaluation. IEEE Software 12(4), 52–62 (1995)
18. Lind, K., Heldal, R.: On the relationship between functional size and software code size. In: Workshop on Emerging Trends in Software Metrics (WETSoM). pp. 47–52 (2010)
19. Lind, K., Heldal, R.: A practical approach to size estimation of embedded software components. IEEE Transactions on Software Engineering 38(5), 993–1007 (2012)
20. Martin, W., Harman, M., Jia, Y., Sarro, F., Zhang, Y.: The app sampling problem for app store mining. In: 12th Working Conference on Mining Software Repositories (MSR). p. 123–133 (2015)
21. Mendes, E., Kitchenham, B.: Further comparison of cross-company and within-company effort estimation models for web applications. In: 10th International Software Metrics Symposium (METRICS). pp. 348–357. IEEE press (2004)
22. Menzies, T., Chen, Z., Hihn, J., Lum, K.: Selecting best practices for effort estimation. IEEE Transactions on Software Engineering 32(11), 883–895 (2006)
23. Nitze, A., Schmietendorf, A., Dumke, R.: An analogy-based effort estimation approach for mobile application development projects. In: 24th International Workshop on Software Measurement (IWSM) and the 9th International Conference on Software Process and Product Measurement (MENSURA). pp. 99–103 (2014)
24. Preuss, T.: Mobile Applications, Functional Analysis, and the Customer Experience, chap. 22. Auerbach Publications (2012)
25. Preuss, T.: Mobile applications, function points and cost estimating. In: International Conference on Cost Estimation and Analysis Association (ICEAA) (2013)
26. Sethumadhavan, G.: Sizing android mobile applications. In: 6th IFPUG International Software Measurement and Analysis Conference (ISMA) (2011)