DEVELOPING METHOD FOR ASSESSING FUNCTIONAL COMPLEXITY OF SOFTWARE INFORMATION SYSTEM

Andrey Solodovnikov
Department of Medical and Biological Physics and medical Information Science
Kharkiv National Medical University
4 Nauki ave., Kharkiv, Ukraine, 61166
andreysldvnk@yandex.ru

Abstract

Solution of problems for improvement of methods and technologies of software configuration remains important and requires the development of the existing information technology to provide customization of software in terms of changing the end user requirements. Changing requirements stipulate the use of iterative software lifecycle. As part of the life cycle, it is necessary use additional methods to simplify of software architecture and obtaining software with a minimum of functional complexity. This is necessary in order to avoid increasing the time and labor costs for design, development and support of software.

To address the disadvantages of existing methods it is proposed to use a method of assessing the functional complexity of the software information system, which is based on the existing graph multilevel model of software architecture. The method is based on FP-metrics calculation for each architectural element and a corresponding level of the graph model. Metrics values allow choosing software modules with a minimum of functional complexity in configuring the software architecture to satisfy the functional requirements of the end user.

Keywords: graph model, functional complexity, function points, software configuration, multilevel form.

DOI: 10.21303/2461-4262.2016.00157

1. Introduction

Existing methods of software building are focused primarily on efficient software development with the requirements, predetermined prior to development [1, 2]. At the same time, in today's software market conditions there is a necessary of adaptation or its customization of the user needs, taking into account the specifics of a particular company and a particular workplace [3].

The need for software customization leads to the fact that the functional requirements are subject to change in the process of developing and maintaining software. Thus, there is a problem of adaptation to the needs of the end user of the Information Systems (IS), caused by the gap between the capabilities of the existing design technologies and practical necessity to software adaptation in the conditions of time-varying functional requirements of the user.

Solution of this problem lies in the use of formal graph models of IS software architecture and methods of their dynamic configuration according to the end user requirements [4, 5]. Modeling software architecture must consider the dynamic aspect of the end user requirements, in connection with which there are difficulties in software building under changing requirements. The major challenges are keeping the functional and non-functional requirements.

It follows from the above, that the problem of IS software structural synthesis, based on the graph model of the software architecture is relevant and requires the development and improvement of appropriate methods of accounting functional requirements of the user for software architecture design [6].

2. Materials and methods

2. 1. Description of graph multilevel model of IS software architecture

Graph model is formed with excess functionality for a given subject area (SA) and provides flexible configuration of the system in terms of evolutionarily changing requirements.

Specification data specification of requirements to the final software product, information obtained at the stages of pre-research, existing architectural solutions and patterns are required to build the model. This implies the presence of existing software modules and SA model. Graph multilevel model of IS software architecture is built on this basis [7]:

$$M = \left\{V_{init}, X_{init}, D_{in}, D_{out}, V^N, X^N, V^M, X^M, VC, PM\right\}, \quad (1)$$

where V_{init} – set of vertices v, which are matched software modules; X_{init} – set of directed arcs $x_{ij} = (v_i, v_j)$ of the G_{in} graph; D_{in}, D_{out} – plurality of input and output data of program modules; $V^N = \{v_i^N\}$ – subset of vertices, which are matched the new software modules and X^N – subset of the links between them; $V^M = \{v_i^M\}$ – subset of vertices, which are matched the modified modules and X^M – subset of the links between them; VC – set of V_{init} vertices characteristics; PM – set of functions, which are matched V_{init} vertices.

Graph model is a set of vertices, which are matched the software architectural elements, tentatively called software modules. Software modules executed sequentially or independently of each other.

Oriented initial graph G_{init} is a set of vertices V_{init} and a set of directed arcs X_{init} and has the form [7]:

$$G_{init} = (V_{init}, X_{init}). \quad (2)$$

Formation of a set of vertices and arcs of the graph (2) takes place on the basis of selection of relevant functional subsystems of investigated object in SA and the establishment of data flow between them. Directed arcs between the vertices of the graph (2) determine the type of connection from the data which are assigned to the set D_{in}, D_{out}. In other words, two data vector are allocated for each directed arc $x_{ij} = (v_i, v_j)$ of the graph (2):

$$D_{in} = \left\{d_1^{in}, d_2^{in}, ...d_k^{in}\right\} \quad (3)$$

– vector of input data invariable during operation and

$$D_{out} = \left\{d_1^{out}, d_2^{out}, ...d_k^{out}\right\}; \quad (4)$$

– vector of output data that had been modified in the process of the program module operation.

D_{in}, D_{out} sets include variables, data structures, links to the data structure and message packets.

A subset of new software modules $V^N = \{v_i^N\}$ and a subset of modified modules $V^M = \{v_i^M\}$ allow configuring the software architecture for the new version of the requirements, defining a modification of the architecture and functionality expansion. For the formation of a new version of the software architecture of the graph (2) a subset of vertices is excluded, which is replaced by a subset V^M. So, software modules of older version are removed. The same operation is done for directed arcs.

For the graph model (1) the set of the characteristics for vertices is defined in this way:

$$VC = \left\{m_1^{VC}, m_2^{VC}, ..., m_n^{VC}\right\}, \quad (5)$$

where $m_i^{VC} = \{isSubG, Comp\}$ – subset consisting of two elements: isSubG – logic variable that determines whether the nested sub-graph for supervertex; subset Comp – lists the number of vertices belonging to the strongly connected component. Supervertices are formed after applying the method of association of vertices in graph model [7] and replace the part of the graph G_{init} (2), and strongly connected components are determined using Kosaraju's algorithm [8]. If set $G_i^{SC} = \langle V^{SC}, X^{SC} \rangle$ (where V^{SC} – set of vertices, and X^{SC} – plurality of directed arcs) as the i-th component of the strong connectivity $G_i^{SC} \subset G_{init}$ that are found in the similar search, then the set Comp will coincide with the set V^{SC}, i. e. Comp= V^{SC}, where $V^{SC} \subset V_{init}$.

Set of functions PM, matched with vertices, is defined as follows:

$$PM = \left\{p_1^{PM}, p_2^{PM}, ..., p_q^{PM}\right\}, \quad (6)$$

where

$$p_i^{PM} = \{UID, PName, PMName, PUPath, PDes, SPath\} \quad (7)$$

subset of the elements necessary to describe software components or modules that perform these functions. To do this, we define the UID as the unique identifier of the computing process; PName – the process name that used by builder in determining the logical connection between the vertex of the graph model and a software module; PMName – the real name of the program module (or library); PUPath – physical path to the file of the software module; SPath – physical path to the specification file, which defines the main characteristics of the module, and is a reference to the module specification; PDES – additional information about the item. In fact, attribute Pname is syntactic unique name of the software module, and PMName – real physical name (semantic) of the process or module that can be referenced by multiple syntax names.

Developer, describing the software architecture correlates the vertices of the SA graph-model with program modules that perform their function, and deals primarily with the attributes PName, PMName, PUPath and SPath. Attribute Unique value of attribute UID is generated automatically and assigning values of PDes attribute is optional, that is, if the additional data is required by the software module. Values of PName and PMName theoretically can be the same, but in practice this is not the right approach. Since it is often possible to find the same type of functions for various IS software subsystems that can be performed by a single software module, keep one copy of the module using in other cases a reference to it, as is customary in the practice of programming in object-oriented languages with dynamic or static libraries. However, use of different syntactic names by developers for the same program module driven by a desire to simplify the submission and processing the graph model, reducing the number of connections between vertices, an excessive amount of which could lead to additional components of strong connectivity, and therefore supervertices.

The document that provides the necessary information for developer about software architecture synthesis is agreed on the file format with one of the XML-standards (e. g., GraphML language).

2. 2. Development of method for assessing the functional complexity of the software

User, solving the problem, may have to select a different method of solution that is different from the existing one. To do this, it must reconfigure the functionality of the workplace and the replacement of the existing software module by the new, which functionality corresponds to the next version of the stated user requirements. In this case, the developer must provide for such situation by including redundant functionality in IS software architecture that supported by the proposed graph multilevel model of software architecture, and allow the user to form component architecture for its subsystems.

Subgraphs of user tasks relevant to their workplace are allocated in the graph model of architecture (1) or generated graph of the whole application architecture is formed. Working with the resultant set of graph model vertices, which care matched the respective software modules, the user replaces or adds new software modules in accordance with subsets $V^N = \{v_i^N\}$ or $V^M = \{v_i^M\}$ and forms a new subgraph of problems associated with the criterion of functional complexity. To do this, it calculates the number of Functional Points according to the formula proposed by A. Albrecht [9]:

$$FP = TotalPoints \cdot \left(0,65 + 0,01 \cdot \sum_{i=1}^{14} F_i\right), \quad (8)$$

where FP – the total number of functional points, $TotalPoints$ – the total value received in the calculation of FP-metric based on rank and evaluate the complexity of the external inputs and outputs, queries, internal logic and external interface files for the current software module, and F_i – the adjustment coefficients of complexity, taking values in depending on the characteristics of the system application settings (software module).

The values TotalPoints and F_i are calculated in accordance with standard FP-metric calculation algorithm [9, 10] and allow to estimate the functional complexity of software modules of selected subgraphs or graph as a whole.

In the case of using graph model of IS software architecture for each vertex of the model it is necessary to store the calculated FP-metric, which value is used in the formation of the graph or subgraph of a new functional configuration. Overall functional complexity of the software FS_{curr} is based on the amount of FP-metrics of all components included in it:

$$FS_{curr} = \sum_{i=1}^{n} FP_i, \quad (9)$$

where n – total number of graph or sub-graph vertices, FP_i – the total number of functional points (8) for the i-th vertex.

The problem of selection of software modules with a minimum of functional complexity is solved when taking into account the non-functional requirements of software modules on each level of a directed graph (2) that is:

$$FSL_k = \sum_{i=1}^{m} FP_i, \quad (10)$$

$$FSL_k \to \min, \quad (11)$$

where FSL_k – value of functional complexity for the k-th level of the graph model, m – the total number of vertices in the k-th level for the current level of graph multilevel model, FP_i – the total number of functional points for the i-th vertex.

Based on this, calculation of the total functional complexity for the whole graph model of IS software architecture is as follows:

$$FS_{curr} = \sum_{k=1}^{p} FSL_k, \quad (12)$$

$$FS_{curr} \to \min. \quad (13)$$

To calculate the functional complexity of each software module based on FP-metrics it is necessary to determine the rank and evaluation for five information characteristics based on a standard method [9]. To do this, the value of half-degree incomes of corresponding graph vertex (2) of graph multilevel model of software architecture (1) is assigned to the number of external inputs. The value of the half-degree outcomes of the graph vertex is assigned to the number of external outputs. The number of internal logical file or logical data groups that are part of a database or a separate file is calculated based on a plurality D_{in} (3). Set (4) is used to calculate the number of external interface files. PDes of subset (7) to describe software components containing more information about the software module is additionally used for obtaining the values of the two last informational characteristics. The sets of input and output data D_{in} are used to determine the types of data for software modules that are matched vertices of the graph model.

Thus, all the information about the IS software architecture contains in the graph model of software architecture is used for the automatic calculation of informational characteristics and FP-metrics, Assessing the functional complexity of configurable software is performed on their basis.

3. Results

The proposed method of assessing the functional complexity of IS software allows creating software architecture with a minimum value of the functional complexity based on the graph model (1) and includes the following steps (**Fig. 1**):

Step 1. Formation of subgraph of user task for the graph (2).

Step 2. Formation of subsets of vertices $V^M = \{v_i^M\}$ and $V^N = \{v_i^N\}$ that will be added to or replace the existing IS software architecture model (1) for the corresponding level of the model.

Step 3. Calculation of the FP-metric (8) for a set of vertices V and that formed based on subsets V^M and V^N of the current level.

Step 4. Calculation of the total functional complexity (10) for the current level of the graph model (1).

Step 5. Checking the value of the indicator (10) for functional user requirements (11). If the answer is negative, the algorithm is repeated from Step 2.

Step 6. When a satisfactory value of indicator (10), the transition to a new level, and repeat of the algorithm from Step 2.

Step 7. Calculation of functional complexity indicator of the whole subgraph of the user problem at the finishing step of the processing of all levels of the graph model (12).

Step 8. If an unsatisfactory (13) value of the indicator (12), return to the original level of the graph model and repeat the formation of subsets of new vertices from Step 2. Completion of the algorithm in the case of failure to carry out further minimization of the functional complexity of software.

Step 9. Compilation of the generated IS software architecture based on an updated graph model (1).

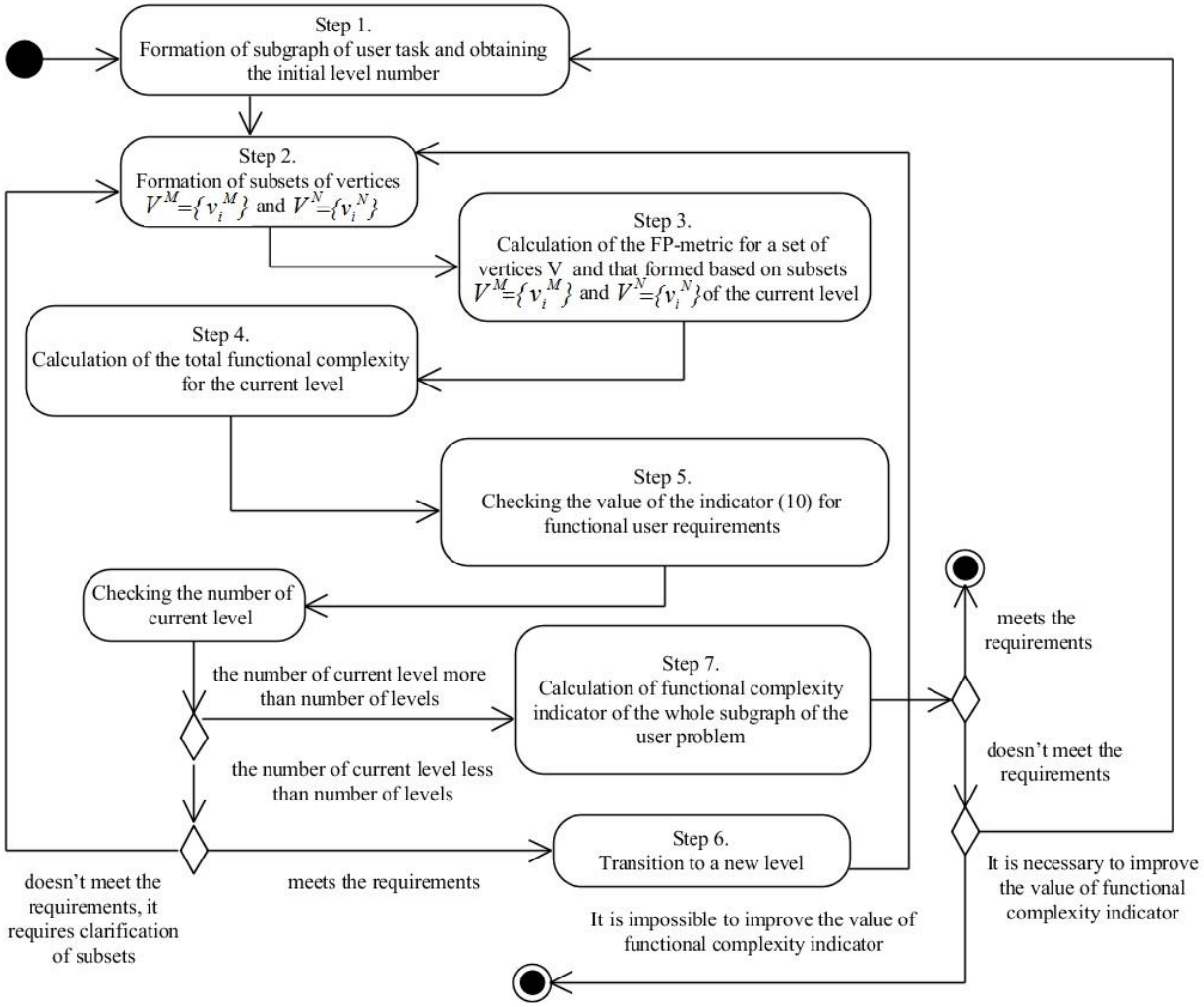

Fig. 1. Steps of method for assessing the functional complexity of IS software

Functionality configuration of the workplace is done on the basis of the proposed method for the current version of the end user requirements, taking into account assessment of the functional complexity of software modules and software in general. The advantage of the proposed method is that the method makes it possible to obtain IS software architecture with the lowest allowable functional complexity, to evaluate compliance of the formed software with the functional requirements of the user and to provide support to the working version of the software. The disadvantage of this method is that when calculating the functional points, adjustment complexity indexes and system settings of the program modules are determined subjectively and depend on the level of expert qualification.

4. Conclusions

Improvement of methods and technologies of software configuration remains important and requires the development of the existing information technology through the use of software architecture models. At the same time, these models require the use of methods of forming a rational architecture and minimize the functional complexity of the developed software. The proposed method allows evaluating the functional complexity of the software modules based on the information about the links for the data between program modules and types of the data. This information is taken from the graph multilevel model of IS architecture and allows calculating FP-metrics. The method includes the steps of determining the functional characteristics of the program modules and calculation of functional complexity criteria, which in turn allows assessing compliance of formed software by functional requirements of the end user with a view of minimizing the FP-metrics values.

Method for assessing the functional complexity of IS software is applicable for dynamic program configuration developed for solving problems in several ways. User, taking into account the current requirements, chooses the appropriate way of solution and dynamically configures the system to their needs. For example, in the case of forming the material needs in solving production problems, the user can select the method of calculating the economic indicators required in this situation. At the same time, the system automatically selects the program modules corresponding to the selected method of solutions and meets the parameters of efficiency of the whole system. Parameters of system efficiency are defined in the user requirements and the selection of modules and their configuration takes place on the basis of graph multilevel architecture model and method for assessing the functional complexity of the software.

The practical significance of the proposed method is to provide the possibility of forming a rational IS software architecture, its dynamic configuration based on the variable functional requirements of the end user.

Further research and improvement of the proposed method is aimed at the rejection of the subjective component in the calculation of functional points of software modules.

References

[1] Smart, J. F. (2015). BDD in Action: Behavior-driven development for the whole software lifecycle. NY: Manning Publications, Shelter Island, 384.

[2] Lavryshcheva, E. (2013). Software Engineering kompiuternykh system. Paradyhmy, tekhnolohyy i CASE-sredstva prohrammyrovanyia. Kyiv: Naukova Dumka, 283.

[3] Levukin, V., Evlanov, M. (2013). Model architecturnogo freimvorka uskorennoy razrabotki informatsionnoy sistemy. Novi Technologii, 1-2 (39-40), 51–57.

[4] Detlef, P. (2012). The Design of GP 2. In Proc. International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2011), Electronic Proceedings in Theoretical Computer Science, 82, 1–16.

[5] Poskitt, C. M., Plump, D. (2014). Verifying Monadic Second-Order Properties of Graph Programs. Lecture Notes in Computer Science, 33–48. doi: 10.1007/978-3-319-09108-2_3

[6] Kääriäinen, Ju., Pussinen, P., Matinmikko, T., Oikarinen, T. (2012). Lifecycle Management of Open-Source Software in the Public Sector. A Model for Community-Based Application Evolution. ARPN Journal of Systems and Software, 11 (2), 279–288.

[7] Chainikov, S., Solodovnikov, A. (2016). Information Technology of Software Architecture Structural Synthesis of Information System. EUREKA: Physical Science and Engineering, 4 (5), 25–32. doi: 10.21303/2461-4262.2016.00125

[8] Sedjvik, R. (2002). Fundamentalnie algoritmy na C++. Part 5. Algoritmy na graphah. Moscow: DiaSoftUP, 496.

[9] Orlov, S. A. Tsilker, B. Ya. (2012). Technologii razrabotki programmnogo obespechenia. Saint Petersburg: Piter, 608.

[10] Albrecht A. J. (1979). Measuring Application Development Productivity. Proc. IBM Application Development Symposium, 83–92.

INVESTIGATION OF THE STABILITY FOR ESTABLISHED FLOWS IN OPEN PSEUDOPRISMATIC CHANNELS

Petro Venherskyi
Department of Applied Mathematics and Informatics
Franko National University of Lviv
1 Universytetska str., Lviv, Ukraine, 79001
petro.vengersky@gmail.com

Yaryna Kokovska
Department of Applied Mathematics and Informatics
Franko National University of Lviv
1 Universytetska str., Lviv, Ukraine, 79001
yaryna.kokovska@gmail.com

Abstract

In this article the equations of channel flow of fluid are considered. They are described by the continuity equation and the momentum equation depending on the rate of flow and cross-sectional area of flow.

It is shown that natural phenomena significantly affect the flow stability and received a condition that ensures the stability of unstated flow at perturbation in initial and boundary conditions. Conditions of stability are checked on test examples, which depend on the following parameters: established movement parameters and middle and bottom slope angles.

It is shown that in such assumptions the system of equations of channel flow of fluid is stable, if received conditions is executed.

Keywords: stability, S-stability, L-stability, channel flow, incompressible fluid, pseudoprismatic channels, perturbations.

DOI: 10.21303/2461-4262.2016.00151

1. Introduction

One of the important tasks for the practice of open-flow stability is investigation of stability in pseudoprismatic channels with big slope angle. The problem is to get the conditions under which the wave that formed in channel as a result of small perturbations will increase or the conditions under which such increase will not.

This problem is different from the usual formulations of stability problems, where deviations are considered at a given point of the flow that will damped during some time.

It should be note that damping of small waves along the channel is a necessary condition but not a sufficient condition for damping waves of finite amplitude, from instability flow at a point follows instable flow in channel, though the stable flow at a point not mean stable flow in general.

Therefore, further along with classical Lyapunov stability, we consider different concepts of stability of open channel flow.

2. Materials and Methods

2. 1. Analysis of the basic phenomena that affect the stability of channel flow

Investigation of the stability of turbulent channel flow [1] associated with the following problems: on the one hand, we can get full information about the stability of the spatial formulation of the hydrodynamics problem; on the other hand, the stability significantly affects the hydrody-