ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 10, Issue 2, pp: (87-90), Month: April - June 2022, Available at: www.researchpublish.com

Optoelectronic control systems: Technological principles to achieve software quality

Omar Salim Abdullah[1], ORAS ABDUL RAZZAQ EYADA[2]

Bilad Alrafidain university college Iraq[1], Basic Education College University of Diyala Iraq[2]

DOI: https://doi.org/10.5281/zenodo.6619281

Published Date: 07-June-2022

Abstract: **The development and application of automated optoelectronic control systems (OESC) [1], which make it possible to increase the efficiency of process control, are extremely in demand in such areas of human activity as industrial production, transportation, security surveillance, etc.**

Keyword: **OESC, subsystem, computers, telecommunication connections.**

1. INTRODUCTION

OESK includes support: technical, mathematical and software. The hardware is represented by an optical-electronic subsystem, computers, telecommunication connections, actuators and can be successfully implemented on the basis of a large number of various and available technical means [2].

The basis of the mathematical software of OECS is the technology of image processing and pattern recognition [3]. The main stages of representation and processing of informative signals that can be performed in OECS include: input and digitization of a video signal, filtering, segmentation, classification of an image and its fragments, recognition of graphic images, decision-making based on the results obtained at each of the stages.

Strict requirements are imposed on the software (software) of OESK, taking into account the conditions of their operation. In this regard, an important problem to be solved in the process of creating the OESC is to ensure the high quality of its software. The complexity of the structure of the OESC software, designed to perform non-trivial data processing procedures, necessitates the solution of new scientific problems in the field of design, implementation and quality assurance of software facilities (PS).

Existing standards and quality management systems (for example, TQM [4]) used in traditional industries cannot be used to ensure the quality of software systems (OESK software in particular) due to the specifics of the software development process. On the other hand, the principles and techniques of management and quality assurance formulated within the framework of well-known models and systems for quality assurance of PS (CMM, PSP, TSP, Cleanroom, MSF [5,6]) are, as a rule, of a general organizational nature, and therefore can be applied in mainly only in the field of PS quality management.

The problem of quality assurance of OESC software considered in the paper is relevant and is due to insufficient development or low efficiency of known theoretical foundations and principles for designing high-quality software.

A comprehensive model of quality assurance, including modern approaches and methods for ensuring the quality of software systems [7] and allowing to guarantee the high quality of design and implementation of OESC software, is schematically shown in Figure 1.

The quality of the OESC software consists of two components: the quality of the design phases and the quality of the software product being developed (the quality of the products obtained at the output of each design phase). The formation of the life cycle (LC) of the OESC software and the construction of a high-quality design process should be carried out in accordance with the current international standards, and for each company it is necessary to form its own profile of

standards for the development and quality assurance of the PS. The creation of an effective life cycle and high-quality processes for designing OESC software is possible only on the basis of modern technological principles for the development of high-quality PS, subject to the maximum use of CASE-tools [8] that automate the corresponding phases of the life cycle.

The main technological principles applied within the framework of the integrated quality assurance model for OESC software are:

- formation of a set (profile) of PS quality assurance standards that regulate the activities and approaches used in the framework of the OESC software design process;

- formation of the model of the life cycle of the software OESK; selection of modern approaches and effective methods that can guarantee the proper level of quality of all phases of the life cycle of the OESC software;

- application of quality control mechanisms in order to reduce defects and errors in the OECS software throughout the entire design process;

- the use of computer automation tools at the stages of the life cycle to increase the efficiency of design and ensure the proper level of quality of the OESC software.

The basic concepts and principles of the structural organization of the life cycle of the PS are given in the ISO 12207 standard (the Russian analogue of GOST R ISO / IEC 12207). An analysis of existing approaches and standards for building a life cycle has made it possible to form a model of the life cycle of the OESC software, schematically shown in Figure 2.

The resulting model reflects the sequence of the main stages of designing the OESC software and information products obtained at individual phases of the life cycle, as well as the roles involved in the design process and their main responsibilities.

The basis of the OESC software life cycle model is an iterative-incremental approach and a unified development process [12]. The iterative-incremental model is close to the spiral approach and ensures the construction of the basic version (prototype) of the OESK software already at the first iteration. The iterative-incremental approach has the necessary degree of flexibility to modify existing and add new requirements to improve the efficiency and functionality of the OESC software. The unified software development process allows overlapping and parallel execution of the corresponding phases of the OESC software life cycle (for example, overlapping the specification and design formalization, overlapping the implementation and testing phases). One of the advantages of a unified development process is positioning on an object-oriented architecture [13] of software systems, support for formal architectural design models and UML [14] notation. To minimize the number of errors and defects in the software system associated with an insufficient degree of formalization of the development phases, in the process of building the life cycle of the OESC software, genetic and formal approaches are used that regulate the use of a formal specification of requirements, a mechanism for analytical verification of the OESC PM and reuse of components. The OESC software life cycle model takes into account the concepts of software quality improvement used in extreme programming [15]. These concepts include: test-oriented programming; use of design, implementation and testing patterns; collective code ownership; accounting for coding notations. related to the insufficient degree of formalization of the development phases, in the process of building the life cycle of the OESC software, genetic and formal approaches are used that regulate the use of a formal specification of requirements, a mechanism for analytical verification of the OESC PM and reuse of components. The OESC software life cycle model takes into account the concepts of software quality improvement used in extreme programming [15]. These concepts include: test-oriented programming; use of design, implementation and testing patterns; collective code ownership; accounting for coding notations. related to the insufficient degree of formalization of the development phases, in the process of building the life cycle of the OESC software, genetic and formal approaches are used that regulate the use of a formal specification of requirements, a mechanism for analytical verification of the OESC PM and reuse of components. The OESC software life cycle model takes into account the concepts of software quality improvement used in extreme programming [15]. These concepts include: test-oriented programming; use of design, implementation and testing patterns; collective code ownership; accounting for coding notations. The OESC software life cycle model takes into account the concepts of software quality improvement used in extreme programming [15]. These concepts include: test-oriented programming; use of design, implementation and testing patterns; collective code

ownership; accounting for coding notations. The OESC software life cycle model takes into account the concepts of software quality improvement used in extreme programming [15]. These concepts include: test-oriented programming; use of design, implementation and testing patterns; collective code ownership; accounting for coding notations.

The profile of standards and a set of modern approaches that take into account the specifics of the life cycle and allow guaranteeing the high quality of individual phases of building OESC software must comply with the international standards of the ISO 12207 and IEEE 1074 series that characterize the life cycle processes. The stages of analysis and construction of requirements specification for OESC software should take into account the IEEE 830 standard, which contains guidelines for the preparation of requirements specifications. The process of designing and documenting the developed design solutions should be consistent with the IEEE 1016 standard, which is a guide for compiling and maintaining design documentation. To improve the style of programming and improve such quality indicators as understandability, learnability, mutability and testability of the program code, in the process of implementing the OESC software, it is recommended to use the provisions of the Hungarian notation, which has become the de facto standard in the field of programming.

To formalize the requirements for OESC software, it is recommended to use the logical approach to describing the semantics and the OCL object constraint language. The construction of a clear architectural concept and correct design models of the OESC software is possible on the basis of an object-oriented paradigm. The use of UML graphical notation will effectively formalize architectural requirements and build correct object-oriented diagrams. The use of a single toolkit (for example, MS Visio or Rational Rose packages) in the process of specification and architectural design of the OESC software will minimize errors associated with transitions of the "specification→formal specification→architecture" type. At the stage of implementation, design decisions, algorithmic support and writing of test equipment for OESC software, it is recommended to use high-level languages.

2. OPEN ARCHITECTURE SOFTWARE OESK

The development of OESC software with high values for the characteristics of efficiency, flexibility, modifiability and extensibility is possible only on the basis of a clear, well-planned and unified architectural concept.

The OESK software architecture is built on the principle of modular decomposition. Features of the modular decomposition of the OESK software are laid down by certain block diagrams. The main element of the OESC software is the kernel manager, which performs all control and coordinating actions related to the computational process. The main stages of the functioning of the recognition process are implemented in the algorithmic subsystem of the OESC. To collect information and perform control procedures, the OESK software interacts with the hardware represented by sensors and actuators (AM) through drivers that control the corresponding devices.

To ensure a high degree of flexibility, modifiability and extensibility, the kernel manager maintains a strictly unified layer of interfaces with the technical and algorithmic support of the OECS. A dedicated interface layer makes it possible to standardize interaction with the corresponding subsystems, as well as to connect various elements of the OECS, regardless of the specific features of their implementation.

To perform its functions, the core of the OESK software interacts with the operating system (Windows 2K or higher) that controls the computer. Interaction is carried out by accessing OS services and calling system functions. To store a priori data and collect statistical information, the OESK software architecture provides an interface with the repository.

In order to meet the requirements of efficiency, performance and real-time operation, the OESK software architecture uses multi-threading, event-based mechanisms, organizes work with external devices, and monitors specified time intervals. The main element of the OESK software core is the control and decision-making block. His responsibilities include managing the progress of calculations, synchronizing the joint work of system elements, making and executing decisions, calibrating devices, setting algorithmic support parameters, and managing the data flow of OESK software.

The event mechanism is implemented in the message processing block. It dispatches hardware interrupts coming to the core, messages transmitted by the OESC algorithm, user control signals, monitors and processes a number of internal system signals. To store current and historical information in the core of the OESK software, a data collection and storage unit is provided that contains input information about the working scene and the results of algorithmic support calculations. The OESK software architecture provides a graphical user interface.

Further development of the ideas proposed in the framework of structural diagrams was carried out on the basis of an object-oriented methodology, on the basis of which the static and dynamic views of the system were designed. The main component of the OESK software is the kernel manager, which structurally consists of the kernel and dispatchers.

The specified architecture of the core of the OESC software contributes to an increase in the degree of flexibility due to the localization of changes caused by the modification of the architecture of subsystems. The OECS Software Managers are Connection Manager, Solution Manager, and User Manager, and are linked to the kernel by a composite aggregation relationship indicating that they are constituent parts of the kernel. The core functions include system initialization, control of the dispatcher part, organization of multi-threaded operation of the corresponding OESC services, support for real-time mode.

The behavior of the OECS software is considered using a finite state machine, which can be in one of the states: receiving messages (waiting), processing messages, making decisions, managing subsystems, and executing the flow of algorithms (identification process). In the state diagram, two parallel states are distinguished: the execution of algorithms (computational thread) and all other states implemented within the message processing thread. Synchronization between them is carried out using kernel events and algorithms.

The analysis of the OESC software schedulability and provision of real-time operation were carried out on the basis of the parallel cooperation diagram of the OESC software subsystems, which reveals the presence of parallel aperiodic tasks of the OESC software, which can lead to blocking and need to be synchronized. These tasks include: the task of processing user messages, the task of collecting data, and the task associated with the identification of control objects. With the help of performance testing of the OESC software and the theory of event sequence analysis, the schedulability of a mixture of tasks and compliance with the real-time mode were confirmed.

REFERENCES

[1] Eremin S.N., Malygin L.L., Rachinsky E.V. Optoelectronic control systems. Processing algorithms , Cherepovets: ChVIIR, 1999. -Ch. 2. - S. 21-22.

[2] Nikulin O.Yu., Petrushin A.N. Television surveillance systems: Educational and reference manual. - M.: Obereg-RB, 1997. - 168 p.

[3] Forsyth, Pons. Computer vision. Modern approach. - M.: Williams, 2004. - 928 p.

[4] Based on the materials of the websites www.citforum.ru, www.standart.ru.

[5] Based on the materials of the websites www.sei.cmu.edu, www.rol.ru.

[6] Humphrey, Watts S., Introduction to the Personal Software Process (SEI Series in Software Engineering), Reading, MA: Addison-Wesley, December 1996.

[7] Lipaev V.V. Methods for ensuring the quality of large-scale software.-M.: SINTEG, 2003. -520 p.

[8] Vendrov A.M. CASE - technologies. Modern methods and means of designing information systems. - M.: Finance and statistics, 1998.

[9] Nepomnyashchy V.A., Ryakin O.M. Applied methods of program verification / Ed. A.P. Ershova. - M.: Radio and communication, 1988. - 256 p.

[10] Myers G. The art of program testing , M.: Finance and statistics. 1982. - 176 p.

[11] Kotlyarov V.P., Kolikova T, Programming technologies. Workshop on software testing in C# , St. Petersburg: SPbGPU, 2004. - 132 p.

[12] Braude E. Technology of software development. - St. Petersburg: Peter, 2004. - 655 p.

[13] Butch G. Object-oriented analysis and design with examples of applications in C ++ / Per. from English - M.: Binom, 1999. - 2nd ed. - 560 p.

[14] Buch G., Rambo D., Jacobson A. UML language. User's Guide , DMK, 2000. - 432 p.

[15] Beck K. Extreme programming. - St. Petersburg: Peter, 2002. - 224 p.

[16] Balykov E.A., Tsarev V.A. Solving the problem of constructing difficult-to-formalize specifications based on expert assessments. Microsoft Technologies in Theory and Practice of Programming, SPbGPU, 2005. - S. 153-154.