International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V69I12P233
Volume 69 Issue 12, 280-286, December, 2021

Component Design of The Complex Software Systems, Based On Solutions' Multivariant Synthesis

Nikita Alexandrovich Ryndin[1], Sergey Vladimirovich Sapegin[2]

[1]*Voronezh State Technical University, Voronezh, Russian Federation*
[2]*Center for Applied Research, Design and Development of Information Systems, Voronezh, Russian Federation*

Abstract *- One of the most critical problems in software development is to find a balance between the allocated resources, quality, and planned functionality of the developed system. For large projects, it is hard to assess risks at the initial stage of development and allocate resources in such a way as to achieve an acceptable result. At the same time, the professional design of the developed system plays a significant role in achieving the result at the initial stage, which determines a realistic sequence of development stages. The article discusses issues of optimal design of complex software systems (CSS) based on a multivariable synthesis of design solutions. The existing methods of CSS design, their disadvantages related to the subjective approach to determining the parameters of the future system and significantly affecting the process and development result are considered. Method for selecting CSS components based on evaluations of conditional probabilities of sharing subsystems, third-party components and documents, calculation of multivariable integration entropies, and their minimization is proposed. The system architecture, which is optimal for this indicator, will help carry out the development under the terms of reference, at the specified time and with acceptable quality.*
The main objective of the research is to find a formal way to design complex software systems more rationally, with a reduction of the human factor. We use multivariant synthesis as a main methodological approach. The paper is novel because, in contrast to general approaches aimed at increasing the importance of the human factor and organizing teamwork, it offers tools to rationalize architecture under proposed quality metrics based on an entropy approach.

Keywords *- complex software systems; multivariable synthesis; entropy; integration.*

I. MAIN TEXT

In the development process of Complex Software Systems (CSS), one of the most critical issues is managing the development process itself, achieving the required quality, compensating for various risks, and preventing failures. According to different modern assessments, only one-fifth of development projects in IT could be successfully ended without any difficulties. In relatively small projects, it is easy to see a failure to meet deadlines or the possibility of failure. If the large-scale information system (IS) is developed with the project designed for months, then from the very beginning, it is hard to assess risks adequately. There are a variety of process management methodologies to mitigate these risks. At the same time, methodologies are sets of recommendations on phases of the design process, development and implementation of systems, resources involved in specific phases, input documents for each phase, phase results, and others. Classification of existing methods of development of components and subsystems of CSS can be presented, as shown in Figure 1.

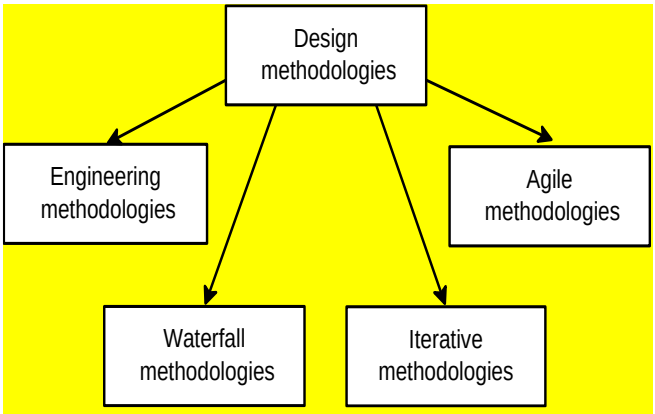

Figure 1. Classification of development methodologies

Engineering methodologies consider the development of software components in the form of a traditional production process for scientific and technical products. An example of such methods is the process recommended by GOST 34.601-90 "Automated systems. Stages of creation." This standard applies to automated systems used in various areas of activity (management, research, design, and others), including their combinations.

These methodologies regulate in detail the process of software component production, paying specific attention to detailed design specifications, detailed design, and subsequent development.

At the same time, the world experience [1, 4, 9, 17, and 18] of software component development shows that one of the properties of such methodologies is a significant chronic underestimation of time, material, and technical resources required for the successful implementation of one or another project.

Improvement of methodologies and management tools in IS component development has resulted in methodologies based on the so-called waterfall model. A feature of this model is the division of the development project into several vast phases, in which a set of processes required to achieve the goal of each phase is formed [3, 11, and 15].

The waterfall design model is usually used with structural software design methodologies, such as Gain-Sarson notation, Barker's method, and others [17] Compared to engineering methodologies, methods based on the waterfall model make it possible to manage resources more freely and to make the software development process more predictable. At the same time, the key to successful development is the presence of experienced analysts with substantial knowledge both in the subject area and in IS design. In the absence of such specialists, projects using the waterfall model are usually characterized by high overspending of material and labor resources at the final stages of the project due to miscalculations at requirements analysis and design [1, 14].

A more progressive approach under initial uncertainty of user requirements is the iterative or spiral development model. The iterative approach implies that each phase of the development process consists of several iterations, which aim at consistent identification and analysis of problems faced, building effective solutions, and, as a consequence, reducing the risk of potential errors in the project. At the same time, the software development task sequence covers several phases, passing through peaks and activity decays. Each project iteration cycle begins with planning for what needs to be accomplished. The result of the execution must be meaningful. The cycle ends with an evaluation of what was done and whether the goals were met. The Rational Unified Process (RUP) is a striking representative of methodologies that use an iterative approach to software development.

Methodologies based on the iterative approach to software development can significantly increase the efficiency of development teams and increase the probability of success of each specific project. However, the following objective difficulties prevent the widespread use of the iterative approach [2, 3, 4, 12].

Due to the development of modern software development technologies, which contribute to the reduction of labor hours spent on implementing particular required functionality, recently there has been a noticeable trend of evolution of development methodologies towards informal, person-oriented methods of organizing work. In particular, a whole family of methodologies focused on principles of the Agile Manifesto has recently appeared [12, 13].

Agile-oriented development methodologies can include XP (Extremal Programming), Scrum, and others. Agile-oriented methodologies can seriously increase the software quality indicators from both technical and user points of view and accelerate the development of software components. The main factor providing the advantage of Agile-oriented methodologies over others is the sound use of the human aspect. Therefore, it is possible to conclude that the main disadvantages of this methodology family will be high requirements to professional qualities of specialists, constant uncertainty present in the evaluation of development time, quality, and size of the software product.

Analysis of all these methods allows us to conclude that each of the presented types cannot fully provide an advantage in developing a wide range of projects. That is confirmed by the history of developing many existing projects and the need to introduce a whole range of mechanisms to ensure their flexibility and adaptability to specific needs into many methodologies (even such voluminous ones as RUP). In general, it makes sense to talk about building and using a particular methodology for each project (or a small set of similar projects), depending on the specifics of the subject area, the user requirements for the project, and the composition of the project team.

Among the features of the CSS component development process that determine its specific nature, we can also highlight the following:

- The developed product demonstrates a high degree of novelty. The process of IS components development almost always aims at solving some unique challenges, achieving some unique properties of the component. Replication of components, in contrast to mass production of the objects of the material world, does not bear practically any costs.
- Information technology develops rapidly. The high rate of industry development makes it impossible to effectively plan the development process of corporate systems in the medium term by traditional means, not to mention the long-term planning.
- There is a greater degree of uncertainty in the goals of IS development. The process of building corporate IS in practice is associated with a constant clarification of the functional requirements for its components, caused by both the problem of understanding between the developers and users of the system and users' ideas about the possibility of using IS components in their business processes.
- There are many methodologies, tools, and solutions in IS development for each specific component [1, 6, 13, 14]. That leads to the fact that no specialist can form a good picture of the most effective solution in the mind when solving problems. In practice, developers try to find an intuitive balance between applying already mastered technologies in solving the tasks and the search and approbation of new ones. These features seriously complicate the use of methods and tools to study the software development process by analogy with manufacturing material products.

Adequate modeling of software development situations requires careful analysis and development of unique models capable of accurately describing the methodologies used today in the production of software systems. Multivariable synthesis of design solutions is proposed as

one of the approaches to form a customizable software development process for the needs of a particular project.

II. MATERIALS AND METHODS

Let us consider formalizing the CSS development task in a general way. The structure of each set of requirements for the Si component of CSS can be represented by the equation [5]:

$$S_i = \big(1 + A(t)\big) * S_{i\,tech} + (1 + B(t) + C) * S_{i\,user} + S_{D(t)} \quad (1)$$

where $S_{i,\ tech}$ – technologies used in the work of the component (including technologies of interaction with other components of the system);

$S_{i,\ user}$ – user requirements for the component;

$A(t)$, $B(t)$ – factors that characterize the change in requirements for a component during its lifetime;

C – ingredient of agreement of requirements to the component between different users of CSS component;

$S_{D(t)}$ – a set of requirements defining the process of combining different functionality in a component (condition of existence of multipliers $A(t)$, $B(t)$).

Thus, the task of CSS component development can be generally represented as achieving, in a limited time, a set of requirements S_x, as close as possible to some ideal set of requirements S_{ideal}. In the general case, it is impossible to achieve the set of requirements S_{ideal} itself in the process of development for the following reasons:

- CSS component development time is limited. The nature of the limited time for component development is since not every set of requirements for a component S_{ideal} can be fundamentally achievable within the time allotted for project development. Thus, the set of requirements formulated for the component under development S_{target} should be initially realistic, i.e., the probability of its achievement in a definite time interval must be different from zero. Based on these considerations, it makes sense to consider not the set of requirements S_{ideal} itself initially. However, the closest to it S_{target} from the set of settled requirements $S_x$that is realistically achievable during the period under consideration.
- Typically, a set of requirements for a CSS component under development is formed not by a single user but by a whole group (or even several groups). Each group member may have requirements poorly aligned with those of the rest of the group. Forming the set S_{ideal} implies complete and consistent unification of all requirements, which is hard enough in practical terms. Therefore, in practice, the set of requirements S_{target} is formed not as a result of the complete unification of the requirements of various project participants but as a certain compromise set of the averaged requirements of all project participants S_{targ_1}, S_{targ_2},…,S_{targ_n}.
- The requirements for a CSS component both from users and from the interacting software change over time (which is reflected in formula (1) by factors $A(t)$, $B(t)$). An attempt to compensate for changes within a software development project leads to an increase in the labor intensity of component development. Due to the unpredictable influence of factors $A(t)$, $B(t)$, many promising software development methodologies currently use the strategy of maximum reduction of development time along with methodologies of early detection of changes in requirements and quick compensating reaction (usually due to human factors). It does not consider the impact of the time factor on changes in requirements for a CSS component outside of the development and startup process.
- Analysis of the properties of the process of achieving user requirements usually does not consider the subjective nature of the process itself, i.e., the qualification of the developers working on the software component. Usually, the qualification of the project executors is evaluated on the principle of matching their skills to the intended actions to develop the component that satisfies the initially specified requirements S_{target}. However, the influence of the factor of requirements change over time, even at the stage of CSS component development, can significantly change the set of qualification requirements for its developers.

Based on the above reasons, the development of corporate is quite a heavy burden. It includes the need for constant consideration of time and human factors and trade-offs under uncertainty.

Among the current methodologies of CSS organization, designed to improve the efficiency of software component development, to use these components, and to minimize the damage from inefficient solutions, we can highlight the strategy of decomposition of tasks arising during the development of the CSS component into small enough, logically isolated parts. The ideology that embodies this strategy is the object-oriented approach (OOA), and technology in modern CSS, the most popular means of implementing this ideology is the Service-Oriented Architecture (SOA) paradigm [6]. The effect of using the SOA paradigm in the CSS design and development is composed of the following components:

- Reducing the size of components under development makes them faster and cheaper to develop. Reducing the development time reduces the impact of the time factor on changes in the set of requirements S_x. Thus, the component under commissioning much more accurately meets the initially stated requirements S_{target}. In this case, when a component stops meeting the current set of requirements S_x, it is much easier to replace it for economic reasons.
- Observing the nature of changes in requirements for software components shows that there are groups of interrelated requirements in sets S_x that change according to a similar law. As a rule, these groups are formed based on the logic of the tasks solved by the CSS components. Breaking down the tasks of the

system into logically distinct components makes it possible to combine groups of interrelated requirements within the development tasks of specific CSS components, excluding their implementation from other components. That minimizes the number of components that need refining based on changes to the common set of CSS user requirements.

- The application of existing OOA practices in CSS component development helps seriously reduce the influence of the factor of changing requirements to the functionality providing the integration of components *A*(*t*). It should be noted that the primary trend in the strategy of software decomposition into as small elements as possible is to increase the importance of factor *A*(*t*). So, OOA pays much attention to combating the integration factor of individual components for data and functionality encapsulation, extensive use of inter-component interfaces, and organization of multi-version component functions based on inheritance and polymorphism mechanisms.

Based on the SOA concept [6], the task of maximizing the economic effect of a single system service can be defined as

$$\int_{T_0}^{T_0+T} f\big(S(t); F(t)\big) \to max \quad (2)$$

Where *S(t)* – a set of business requirements for the service, *F(t)* – implemented functionality, *T* – service usage time, T_0 – service start time, *f* – function evaluating the compliance of the component functionality *F(t)* with current requirements *S(t)*, $f \in [0,1]$. As the simplest, roughest version of the function *f*, we can use an expression in the form

$$f = \frac{D(S(t),F(t))}{|S(t)|} \quad (3)$$

where *D* – the power of the symmetric difference of the sets *S(t)* and *F(t)* at time *t*, |*S(t)*| – the power of set *S(t)*. The practice shows that the dependence of the distance between the sets *S(t)* and *F(t)* (i.e., the degree of compliance of the component functionality with the business requirements) in case of the intensive (purposeful) development process has S-shaped character. That is because, at the development cycle beginning, resources are spent not so much on the implementation of business functions but the construction of the software architecture. Approximately in the middle of the development cycle, the most optimal productivity is achieved, which decreases at the end of the development cycle due to the complication of both the process of implementation of individual business functions (the most complicated, complex business processes remain for implementation) and the process of integration of developed functions into the existing system. Accordingly, function *f* variants, more consistent with statistical data, should be sought among the families of S-functions of different curvatures.

A schematic diagram of the function f illustrating the ideal life cycle of such a service is shown in Figure 2.

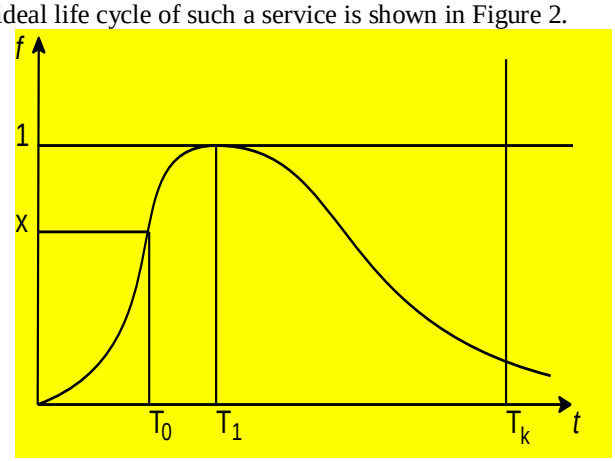

Figure 2. Lifecycle of service in the traditional development approach

Phases of software design and development are usually planned within the time interval [0, T_0], which ends after the release of the first version and the start of the pilot implementation. When using a waterfall model to organize the workflow by time moment T0, the planned phases of service development and design are finished. In this case, the proportion of the x functionality of the developed software *F(t)* to the set of business requirements *S(t)* belongs to the range [0, 1]. Sometimes if x is less than some threshold value, the decision is made to close the project and exclude its results from the enterprise automation process. Typically, after the actual closure of an unsuccessful project, a new project is opened to solve the same automation, where the work is done, taking into account the experience and developments of the first project. However, in the framework of the SOA approach, it is also possible not to start a new project within the same task. However, the developments of the completed project are used for related services in terms of functionality. In this case, the area of functionality of these services, respectively, is expanded.

Once the implementation decision is made, refining the service, testing, and embedding it into the operating business processes of the enterprise begins. That corresponds to the testing and implementation phases. Sometimes, suppose there is a significant divergence between the service F(t) functionality and the structure of business requirements *S(t)*. In that case, the process can return to the development phase and sometimes to the system design phase (the so-called "jumping salmon" model). Simultaneously with refining, the use of the developed product in the work of the enterprise begins, so at the stage of approaching the service functionality to the desired result [T_0, T_1], we can already talk about the effect of using the service. The process of service refining and implementation continues until some threshold value y (ideally *y*=1) of service functionality *F(t)* meets business requirements *S(t)*, after which the project is formally completed with the decision about industrial use of the developed software product. At this point, the project in terms of traditional methods can be considered complete.

III. RESULTS

Let us consider the process of building an optimal methodology for CSS projects based on the formalized apparatus of multivariable synthesis of design solutions. To do this, let us define the following set of artifact types that each project may contain:

A. Document

A document refers to an artifact that contains descriptive, advisory, or reference information that affects the decision-making process when using project results. A document in our definition should be understood not only paper or electronic documents but also other similar entities, such as RUP models or MSDN knowledge bases.

B. Component

A component refers to an artifact created by the project team to solve any project tasks. The artifact can be a software application or subsystem, developed network architecture, configuration of platform software, and others.

C. Third-party subsystem

A third-party subsystem refers to an artifact, using a black box metaphor with defined inputs, outputs, states, and behavior. They may be previously developed libraries, third-party software components, network and computing equipment, licensed and certified methodologies (for business processes), and others [5].

The project as a whole is described by a set of artifacts $b_1 \ldots b_m$, which are selected from a specific set B_0, consisting of the possible in the project artifacts of all the above types. To form each artifact in the project (in the case of third-party subsystems – for adaptation), one or more methods $a_1,\ldots, a_n$ are used from the common set of approaches A_0, formed based on the applicability analysis of various methodologies to the generation of artifacts. It is assumed that only one method a_i with the probability of P_{ij} can take part in the generation of each artifact b_j. In case if there is an initially intersecting set of methods (for example, in the development of CSS components, there is often such a set of paradigms as the OOA, a spiral development methodology, and a source code design standard), then the intersection A'_0 is formed from the set A_0, containing all possible options for a combination of methods:

$$A'_0 = A_{01} \times A_{02} \times \ldots \times A_{0N} \quad (4)$$

where for every element of the new set $a'_i=\{a_{i1}, a_{i2},\ldots,a_{iN}\}$, which is a combination of *1...N* elements of the initial seta_i, the probability of its existence is $P(a'_i) \neq 0$. The resulting set will consist of a plurality of combinations of methods simultaneously applied to the generation of each artifact.

The task of building the optimal methodology for project maintenance is thus reduced to finding the most optimal option for integrating the set of methods for generating artifacts A_n and a set of artifacts of the project B_m. At this, the form of interaction of the presented integration levels is described using probability vectors of sharing different variants of these levels:

$P(B_m/A_n)$ – conditional probability of selecting the B_m-th version of the set of design artifacts when using the A_n-th variant of the synthesized project maintenance methodology;

$P(A_n/B_m)$ – the conditional probability of using the A_n-th version of the methodology to generate the B_m-th variant of the set of design artifacts.

$$H(A) = -\sum_{n=1}^{N} P_n^A \lg P_n^A \quad (5)$$

$$H(B) = -\sum_{m=1}^{M} P_m^B \lg P_m^B \quad (6)$$

The variety of variants used is measured by the entropy of combinations, taking into account the mutual influence of the integrated levels:

$$H(S) = H(AB) = H(B) + H_B(A) \quad (7)$$

where $H_B(A)$is the conditional entropy of the variety of variants for using approaches to generate the necessary set of artifacts *B*. In the case of the inverse problem, when the set of artifacts depends on the set of used techniques, the following relation takes place

$$H(S) = H(AB) = H(A) + H_A(B) \quad (8)$$

Accordingly, the task of building an optimal methodology for each particular project can be formulated as one of the tasks of multivariable synthesis, similar to the cases considered in [5]. For our case, the solution is to choose one element from a modified set of approaches A' and a set of artifact sets *B* according to some requirements $F^*_i (i = 1, I)$. The number of variants for selection is

$$L = \prod_{j=1}^{J} A_j * \prod_{k=1}^{K} B_k \quad (9)$$

and apriori entropy is

$$H(S_2) = \sum_{j=1}^{J} \lg A_j + \sum_{k=1}^{K} \lg B_k \quad (10)$$

Having assessed the dimensions and performed the transformations, we have:

$$H(S_2) = \sum_{j=1}^{J} \lg A_j + \sum_{k=1}^{K} \lg B_k \leq$$
$$H(\mu_2) \leq \sum_{m=1}^{M} \lg N_m = \sum_{m=1}^{M_B} \lg N_M^B + \sum_{m=M_B+1}^{M} \lg N_m^C \quad (11)$$

where N^R_m is the number of simple experiments corresponding to elements of the set $A(m = 1, M^B)$; N^C_m is the number of simple experiments corresponding to elements of the set $B(m = M_B+1, M)$. Similar to problem S_2 in [5], let us introduce a set of booleans defining the boundary conditions of the multivariate optimization model. Let us supplement the set of constraints with logical relations, which allow us to reject knowingly incorrect variants of methodologies to generate artifacts (4). The procedure of optimal choice of variants is made in two stages:

- Sets of variants A^*_j, B^*_k, satisfying the above conditions are selected, and the values are calculated

$$v_i = \varphi_i(A_j^*, B_k^*) \quad (12)$$

- Vector values u^* are selected, characterizing the optimal parameters of the sets, providing minimization of the sum of squares of inconsistencies of the optimization criteria in the problem of structural synthesis of the integrated system, by the condition:

$$\Phi_j = (v_i(u_k) - y_i^*) \quad (13)$$

Mathematical description of relations between indicators of the system and initial elements is made based on approximation of functional dependences obtained from statistical data. The varieties of optimization models used in solving this problem may depend on:

- The degree of uncertainty in specifying design requirements;
- A set of possible approaches and methodologies;
- The specifics of the subject area and typical solutions;
- The composition of the basic set of artifacts.

IV. DISCUSSION

The proposed methodology for designing CSS consisting of multiple components makes it possible to formalize the process of selecting system architecture and its components based on the assessment of conditional probabilities of using one or another system component in combination with other selected components, calculation of multivariate integration entropy and its minimization as a target function of multicriteria optimization. This approach allows us at the stage of schematic design of the future CSS to determine the set of core components, which helps to ensure the specified requirements for the system, timing, and development quality. Determination of conditional probabilities of sharing the system components can be carried out from peer reviews or collecting statistics on the common use of these components in other projects based on retrospective information. Process formalization of selecting the system architecture, its components, and the use of mathematical relationships to assess the most promising option makes it possible to choose not based on the vision of the chief designer but using the quantitative characteristics of the variants under consideration.

The proposed methodology for designing complex software systems consisting of many components makes it possible to formalize the process of selecting system architecture and its components based on the evaluation of conditional probabilities of using one or another system component in combination with other selected components, calculation of multivariate integration entropy and its minimization as a target function of multicriteria optimization. In the practical application technique, it was found that it helps to make more accurate time estimates of the project developed since joint conditional probabilities of the use of components to some extent characterize the complexity of integration of these components in the system. Also, on a certain set of projects, there are tendencies to clarify the boundaries of different technology stacks and, even in some cases, the possibility of predicting for emerging technologies in which stack they will be most popular.

Unlike others used in the development of complex software systems, it may seem that the proposed methodology does not consider the human factor. However, the determination of conditional probabilities of joint use of system components is carried out from peer inspections of specialists or by analyzing statistics of mutual use of these components in other projects based on retrospective information. Both methods depend on the experience, qualification, and thinking style of the community developers in question. A significant human factor also manifests itself in the methodology set up in the way systems are divided into levels and components. That, in turn, leads to the fact that guaranteed reproduction of the same results of the adjusted methodology is possible only within a sufficiently homogeneous community of developers. The information content transfer from one group of developers to another should, at least, be verified.

In general, the approach makes it possible at the stage of the conceptual design of future complex software systems to determine a set of basic components, which helps to ensure the specified requirements for the system, timing, and quality of development. Formalization of the selection process of system architecture, its components, the use of mathematical relations to evaluate the most promising option enables to make design decisions in a balanced manner, using the quantitative characteristics of the options under consideration.

V. CONCLUSION

The modern software industry offers many different methodologies and approaches to the organization of the software development process. At the same time, due to differences in the subject area, software scope, technologies used, and ready-made subsystems, the most rational variant is to build own process based on already existing ones by borrowing different parts.

At the same time, the focus of overhead is gradually shifting from developing individual components to integrating selected components. In general, this process is so complex that most projects developed today either do not go beyond a single language or use ready-made integration structures (such as MVC within the HTML/JS/ServerPL technology stack). In this case, the issue of integration arises only when there is a need to combine large subsystems into a single whole. Usually, such tasks are associated with significant discrepancies in the subject area understanding, resulting in large overhead costs for various format converters, duplication of data with routine synchronization, and others). If initially to perceive the CSS development project as an integration of components, most of which are already developed, it is possible to get a fairly large benefit in cost and time at the stage of development. Another thing is that the integrated components must be compatible with each other.

Availabilityofadditionaltoolsthat make it possible to carry out process design based on the assumed rational software architecture, in some cases, provides a significant

economic effect. Thus, using a multivariable synthesis of design solutions in the task of component design of CSS is a relevant practice.

VI. DECLARATIONS

A. Author Contributions

Conceptualization, 1st and 2nd authors; methodology, 1st author; software, 1st and 2nd authors; validation, 1st, and 2nd authors; formal analysis, 1st and 2nd authors; investigation, 1st, and 2nd authors; resources, 1st and 2nd authors; data curation, 1st, and 2nd authors; writing—original draft preparation, 1st, and 2nd authors; writing—review and editing, 1st and 2nd authors; visualization, 1st, and 2nd authors; supervision, 1st author; project administration, 1st author; funding acquisition, 1st, and 2nd authors. All authors have read and agreed to the published version of the manuscript.

B. Data Availability Statement

Data is contained within the article.

C. Funding

The publication was made at the expense of the authors' personal funds.

D. Acknowledgments

The studies were conducted at the Center for Applied Research, "Design and development of information systems" (CAR DDIS), and the results were tested in commercial projects developed by the center.

E. Conflicts of Interest

The methods of component design, software life cycle, and rationalization of the CSS construction process described in the article were developed for testing and commercial use as part of the work of CAR DDIS. The use of statistics on commercial projects under development is limited according to the NDA used within CAR DDIS.

REFERENCES

[1] Cantor, Murray. Software leadership. A Guide to successful software development (2002).
[2] Booch, Grady, James Rumbaugh, and Ivar Jacobson. The Unified Modeling Language. User's guide (2000).
[3] Kuznetsov, Sergey Dmitrievich. Design and development of corporate information systems (1998).
[4] Jacobson, Ivar, Grady Booch, and James Rumbaugh. Unified software development process (2002).
[5] Ryndin,AlexandrAlexeevich. Multivariant integration: theory and applications in CAD: Monograph (2018).
[6] Service-Oriented Architecture (SOA). URL Integration. url: http://www.urlintegration.com/?page_id=752 (Accessed: 03.05.2016).
[7] Jacobson, I., and B.Meyer. Methods Need Theory. Dr. Dobb's (2009).
[8] Jacobson, I. and I.Spence.Why We Need a Theory for Software Engineering. Dr. Dobb's (2009).
[9] Gray, J., and B.Rumpe. Agile model-based system development. Software and Systems Modeling 17(4) (2018): 1053–1054. doi:10.1007/s10270-018-0694-1
[10] Gu, Q., and P. Lago. Guiding the selection of service-oriented software engineering methodologies. Service-Oriented Computing and Applications 5(4)(2011): 203–223. doi:10.1007/s11761-011-0080-0
[11] Harlin, U., and M. Berglund. Designing for sustainable work during industrial startups—the case of a high-growth entrepreneurial firm. Small Business Economics 57(2)(2021): 807–819. doi:10.1007/s11187-020-00383-3
[12] Hohl, P., J.Klünder, A. van Bennekum, R.Lockard, J. Gifford, J.Münch, M.Stupperich, and K. Schneider. Back to the future: origins and directions of the Agile Manifesto – views of the originators. Journal of Software Engineering Research and Development 6(1)(2018). doi:10.1186/s40411-018-0059-z
[13] Jacobson, Ivar, and Roly Stimson. The Essence of Software Engineering (2017) 37-58. doi:10.1007/978-3-319-73897-0_3
[14] Karhapää, P., Behutiye, W., Rodríguez, P., Oivo, M., Costal, D., Franch, X., Aaramaa, S., Choraś, M., Partanen, J., andA.Abherve. Strategies to manage quality requirements in agile software development: a multiple case study. Empirical Software Engineering 26 (2021): 28. doi:10.1007/s10664-020-09903-x
[15] Kettunen, P., and M.Laanti. Future software organizations – agile goals and roles. European Journal of Futures Research 5(1)(2017) 1–15. doi:10.1007/s40309-017-0123-7
[16] Klotins, E., M.Unterkalmsteiner, andT.Gorschek. Software Engineering in Start-up companies: an Exploratory Study of 88 experience reports. Empirical Software Engineering 24(1) (2016). doi:10.1007/s10664-018-9620-y
[17] Rabiser, D., H.Prähofer, P.Grünbacher, M.Petruzelka, K. Eder, F. Angerer, M.Kromoser, and A. Grimmer. Multi-purpose, multi-level feature modeling of large-scale industrial software systems. Software and Systems Modeling 17(3) (2018): 913–938. doi:10.1007/s10270-016-0564-7
[18] Stevenson, J., and M. Wood. Recognising object-oriented software design quality: a practitioner-based questionnaire survey. Software Quality Journal 26(2) (2018) 321–365. doi:10.1007/s11219-017-9364-8
[19] Delplanque, J., Etien, A., Anquetil, N., Auverlot, O.: Relational database schema evolution: an industrial case study. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME) (2018). https://doi.org/10.1109/ICSME.2018.00073.
[20] Engelenburg, S. van, Janssen, M., &Klievink, B. (2019). Design of a software architecture supporting business-to-government information sharing to improve public safety and security: Combining business rules, Events, and blockchain technology. Journal of Intelligent Information Systems, 52(3), 595–618. https://doi.org/10.1007/s10844-017-0478-z
[21] Haakman, M., Cruz, L., Huijgens, H., & van Deursen, A. (2021). AI lifecycle models need to be revised: An exploratory study in Fintech. Empirical Software Engineering, 26(5) 1–29. https://doi.org/10.1007/s10664-021-09993-1
[22] Greifenberg, T., Hillemacher, S., &Hölldobler, K. (2020). Applied Artifact-Based Analysis for Architecture Consistency Checking. Ernst Denert Award for Software Engineering 2019, 61–85. https://doi.org/10.1007/978-3-030-58617-1_5
[23] Hacks, S., Lichter, H.: Qualitative comparison of enterprise architecture model maintenance processes. In: 40 Years EMISA 2019 (2020)
[24] Kude, T.: Agile Software Development Teams during and after COVID-19. http://knowledge.essec.edu/en/innovation/agile-software-development-during-after-COVID19.html (2020).Accessed 5 Mar 2021
[25] Salentin, J., Hacks, S.: Towards a catalog of enterprise architecture smells. In: Gronau, N., Heine, M., Poustcchi, K., Krasnova, H.(eds.), WI2020 Community Tracks, GITO Verlag, pp. 276–290(2020)
[26] Salameh, A., Bass, J.M. An architecture governance approach for Agile development by tailoring the Spotify model. AI &Soc (2021). https://doi.org/10.1007/s00146-021-01240-x
[27] Raj, V., Sadam, R. Evaluation of SOA-Based Web Services and Microservices Architecture Using Complexity Metrics. SN COMPUT. SCI. **2,** 374 (2021). https://doi.org/10.1007/s42979-021-00767-6
[28] Kalalali Roseline Asimini-Hart, Bennet Okoni, Nuka Nwiabu, Mechanism For Detection of Software Design Defects, SSRG International Journal of Computer Science and Engineering 7.3 (2020): 12-21
[29] Jigar K Patel, Critical Success Factors for Implementation of Enterprise Resource Planning Software, SSRG International Journal of Computer Science and Engineering 8.2 (2021): 1-5.
[30] Mitesh Athwani, A Novel Approach to Version XML Data Warehouse, SSRG International Journal of Computer Science and Engineering 8(9) (2021) 5-11.