The Software Emergent Properties and them Reflection in the Non-Functional Requirements and Quality Models

Tetiana Hovorushchenko

Abstract - **In this paper the analysis of the known approaches to detection of emergent properties of software system in the context of involvement for this of the software quality models was done. The attempt of evaluation of share of the non-functional requirements in the software quality models was done.**

Keywords - **software, software system, software quality, non-functional requirements, emergent properties.**

I. INTRODUCTION

The software quality is essential factor for its successful implementation and exploitation. The researchers of The Standish Group International consider the most significant cause of the low quality of the large software projects. It is the increasing of the number of components (subsystems) and interfaces between them [1]. The result is the fact, that the software incidents and accidents today include the new type of accident, that caused by the components interaction [2]: each component has no defects (meets the requirements to it), but incorrect interaction between components leads to the problems. The improving of the quality and reliability of only separate components of software systems cannot prevent such accidents because their cause is not a failure of individual components.

The decline of the software quality can caused by the neglect of some software peculiarities in the process of software quality evaluation and assurance. The research of modern methodologies of software quality evaluation [3-6] has shown that nowadays they weakly consider the systemic aspect of modern software. In particular the insufficient attention is paid to emergent properties of software (that are an integral feature of the system).

The term "emergent properties" can be found in the number of works that devoted to software, but various authors differently interpreted this term. The analysis of the known definitions of "emergent property" in [7] showed, that emergent properties are properties that cannot be localized in the one component, but are the result of the components interaction.

Henceforth *the software emergent properties* are the randomly appear properties, that are displayed in the process of the software functioning during the subsystems interaction through interfaces, and if there are the specific data and the external influences.

Tetiana Hovorushchenko - Khmelnitsky National University, Institutska Str., 11, Khmelnitsky, 29016, UKRAINE,
E-mail: tat_yana@ukr.net

Henceforth *the emergent behavior* is the behavior of the software system, which is not typical to any separate subsystem, and emerges only during the interaction of subsystems; the developers didn't predict this behavior.

The emergent properties are not considered in the current approaches to the software development and the evaluation of its quality. So, these properties lead to permanent integration problems that explain high percentage of non-quality (failed & challenged) software projects [8]. The reasons of the ignoring of the emergent properties have not technological, but methodological nature. Emergent properties should be predicted at the early stages of the software life cycle for the further successful overcoming of the difficulties of software systems integration. But the evaluation and prediction of emergent properties at the early stages of the life cycle is the challenge during the software design, because the emergent properties must be assessed with respect to a thorough knowledge of the system and environment [9].

In addition, the software quality may be low by reason of the insufficient attention is paid to the subject domain information at the different stages of the software life cycle. New information can come at the different life cycle stages – as at the stages of requirements formulation and design, and at the stages of implementation and exploitation, - but it is often neglected. This neglect of subject domain information at the all stages of software life cycle is one of the critical factors during the software development.

During the software project, we often cannot estimate the share of the informational indeterminacy of the project. Identification of the information, that appears in the process of interaction of "subsystems-interfaces-data-external influences", is especially difficult. Identification of future properties of developed software, which will display this information (emergent properties and emergent behavior), is even more difficult task.

There is the situation, characterized by premature design decisions and their documentation, prior to understanding the design and its emergent properties and their impact on the functional characteristics of the product. This is area, referred to as the "knowledge gap," that is the result of the practice and the root cause of many engineering failures [10].

The fact of partial non-consideration of the subject domain information at the different stages of the software life cycle testifies that the size of knowledge gap is not constant for software project – during the

lifecycle it can increase and decrease, since new information appears and it should be taken into account.

Knowledge gap contains the software emergent properties. The subject domain information may be valuable for prediction and detection of the software systems emergent properties. Such information will reduce the size of the knowledge gap, and detected emergent property ceases to be emergent property. The reducing of the number of software emergent properties will increase software quality.

Thus, the purpose of this work is the analysis of the known approaches of the software emergent properties detection by the use of the software quality models. Because some of the software emergent properties are reflected in the non-functional requirements, it should also analyze the ability of consideration and evaluation of the non-functional requirements in the software quality models.

II. Approaches to Detection of Emergent Properties of Software Systems

One approach to detection of software emergent properties is the chronological approach as part of Software Process Improvement (SPI). This approach is based on the concept that emergent properties are different from the new properties, because always arise from the prehistory and context of the system. Chronological approach assumes the power increase of the software system and study the behavior of the system over time for analysis of the emergent nature of processes [11].

Another approach to detection of the software emergent behavior is the comparison of software systems with natural groups with emergent behavior - for example, a group of migratory birds or ant colonies [12-16]. The authors of [16] propose the genetic algorithms as a tool for modeling of emergent behavior of system, the authors [12] – the laws of Social Network Analysis (SNA), the authors [14] – agent-based modeling, that focuses on how local interactions among agents serve to create larger and perhaps global structures and patterns of behavior. The modeling efforts for emergent behavior are conceptual and only at the beginning stage for this new endeavor to integrate emergent behavior into architecture models [14].

Paper [17] proposes to use for demonstration of the emergent behavior not only more specialized techniques such as avoiding order n-squared computations and using adaptive optimization, but also approaches and techniques from biological and social systems, physical sciences. Emergent behavior in the form of influence, indirect effects, cascades, and epidemics among the autonomous constituents permeates systems of systems. Emergent behavior cannot be understood from the statistic.

The next approach to prediction of emergent properties is the use of multi-agent formal methods [15]: WSCCS, for that there are no effective tools to aid calculation and interpretation of the emergent behavior of more than two agents and no visualization capabilities exist to aid in the study of the emergent behavior; X-Machines, that has few predictive qualities for emergent behavior of multiple agents. Current abilities of methods are not robust enough for these purposes and will need to be enhanced by greater use of probability, Markov Chains and/or Chaos Theory.

Another approach to detection of emergent properties is UX-aware model for software requirements [18], that shows the relation of functional requirements FR and quality requirements QR to user experience UX. This model emphasizes that requirements can be categorized into layers where higher layers depend on requirements below. Different layers ask for different focus: individual features, emergent properties and end user's perception.

The next approach to detection of the software emergent properties is the use of the aspect-oriented programming technology [19]. It possible that multiple aspects woven into a primary abstraction class can interact in ways that are difficult to understand and result in emergent behaviors that are unexpected and beyond the composite specification of the woven artifacts. There are four alternatives that must be considered [19], including: the fault is an emergent property that results from some interaction between the aspect and the primary abstraction; the fault is an emergent property of a particular combination of aspects woven into the primary abstraction. New tools are needed that take into the account the effects of weaving and that can identify potential emergent properties.

Objective-based approach to detection of emergent properties of software component-based systems is semantic validation of emergent properties in component-based simulation models [20]. Emergent properties validation focuses on showing that the unexpected behavior is valid (or invalid) for a given set of conditions. In component-based complex systems, emergence validation approaches are classified in three key categories, namely, grammar-based, variable-based and event-based. Authors of [20] simulate the complex system and, at each simulation step, analyze the system state. Then authors compare the simulated system state with a calculated system state using reconstructability analysis. If there is an unacceptable deviation in the observed parameters, the authors highlight this state as a possible emergence state and add it to an emergent set.

The next approach is the use of service-oriented architectures (SOAs) [17, 21]. Author of [21] compares SOAs with Declarative approaches. One might speculate that there is a critical level of behavioral complexity below which it is feasible to program declaratively, but above which the attempt to do so becomes, in effect, an

increasingly chaotic process of "programming by emergent behavior;" that is, an attempt to reach the desired results by manipulating declarative rules, without a predictable connection between rules and results.

Another approach to detection of emergent behavior is based on the information model on SysML language [22]. As an invariant approach [21] is recommended to use the game theory to detection of emergent behavior, but details of these approaches are not considered.

Let's consider emergent software engineering technologies [23]. The emergent ones include the technologies aiming at automatic composition of software functionalities (on-line or off-line) achieved on account of multi-agent collaboration, the use of webservices, or automatic reasoning. These technologies provide for flexibility and plug-and-play composition of software. The main implementation challenge of component-based engineering is achieving execution properties such as determinism and real-time guarantees when using middleware [23]. It is clear that such predictability (determinism) cannot be achieved until the emergent properties are not detected, because, in fact, they are sources of non-determinism [21].

The paper [21] argues that if one cannot use induction, one might still be able to work from observations of the system' s behavior and reason in reverse (deductively) to infer what actually happened, once something does go wrong. These systems exhibit emergent behavior, and given the deductive inference, there might be some hope of directly removing the causes of their emergent misbehaviors. Jeffrey C. Mogul in [21] proposed the research agenda to deal with emergent misbehavior in complex software systems, that consists: 1) Creating a taxonomy of emergent misbehavior (thrashing, unwanted synchronization, unwanted oscillation or periodicity, deadlock, livelock, phase change, chaotic behavior); 2) Creating a taxonomy of typical causes (unexpected resource sharing, massive scale, decentralized control, unexpected inputs or loads); 3) Developing detection and diagnosis techniques (Fourier analysis; the system, called Pip); 4) Develop prediction techniques (leaves open the possibility of prediction techniques that operate at the whole-system level, to predict that a system could be prone to some unspecified form of emergent misbehavior and might be possible to predict the onset of serious emergent misbehavior from advance symptoms); 5) Develop amelioration techniques (systematic overprovisioning, admission control, introspection, and closed control loops for adaptation; system predictability can be improved by either by preventing unpredicted component behavior from propagating throughout the system, or by protecting components against unexpected inputs); 6) Develop testing techniques (the conditions that lead to emergent misbehavior are not always knowable or anticipated during testing).

Despite a plethora of definitions and methods, a practical approach to identify and validate emergent properties in newly composed simulation models remains a challenge [20]. There has been active research in defining emergence, all of them explain the nature of emergence and emergent properties, but are not formalized and are not suitable for modeling and prediction of emergent properties. In addition, all approaches are theoretical. There is no statistics about for what projects the approach was used and what effect it has given (as these projects were ended). Therefore formally detection and validation of emergent properties and automation of this process remain a key challenge.

III. SOFTWARE QUALITY MODELS

According to [24] the software quality model is "the set of characteristics, and the relationships between them that provides the basis for specifying quality requirements and evaluation".

Since 2000 the construction of software started to depend on generated or manufactured components and gave rise to new challenges for assessing quality. Consequently the models are classified in Basic Models which were developed until 2000, and those based on components called Tailored Quality Models – Fig.1. In the present age, aspects of communications play an important factor in the quality of software [3].

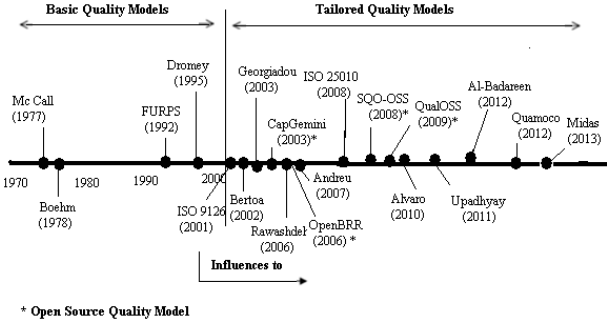

Fig.1. Quality Models

The Basic Models (Mc Call, Boehm, FURPS, Dromey, ISO 9126, ISO 25010) are hierarchical in structure; they can be adjusted to any type of software product and are oriented to the evaluation and improvement [3].

The main characteristic of Tailored Quality Models (Bertoa model, GEQUAMO Model, Alvaro Model, Rawashdesh Model and others) is that they are specific to a particular domain of application and the importance of features may be variable in relation to a general model. They arise from the need of organizations and the software industry for specific quality models capable of doing specialized evaluation on individual components. According to them, the success of the software product strongly depends on the quality of the components [3].

Authors of [25] classify the models according to user's characteristics. They distinguish three categories of models that correspond to: 1) the level of general public use or specific domain, 2) organizational level that focus on satisfying the interests of a specific organization, and 3) the level project that applies to a specific project to ensure quality. There are two strategies for software quality modeling: 1) fixedmodel approaches; 2) define-your-ownmodel approaches [25].

Ayala [26] establishes a process to select software components. The study concludes with varying results. One of them was discovering the use of informal procedures to find, evaluate and choose components, and hence there exists the need for methods to do components selection and support tools to help in it.

The major contribution of the McCall model was to considerer relationships between quality characteristics and metrics. The main drawback of it is the low accuracy in the measurement of quality and non-consideration of the functionality [3]. Boehm establishes large-scale characteristics that constitute an improvement over the Mc Call model because adds factors at different levels [3]. The Dromey model is based on the perspective of product quality. It states that for a good quality product, all the elements that constitute it, should be so. The quality evaluation for each product is different and a more dynamic evaluation is established [3]. The FURPS model categorizes the characteristics as functional and non-functional [3]. The ISO 9126 model has two main parts consisting of: 1) the attributes of internal (the system properties that can be evaluated without executing) and external (the system properties that can be assessed by observing during its execution) quality; 2) the quality in use attributes [3]. Model ISO 25010 is subdivided into 8 sub key features and characteristics. One of its main objectives is to guide in the development of software products with the specification and evaluation of quality requirements [3].

The Bertoa model is based on the ISO 9126. It defines a set of quality attributes for the effective evaluation of the software. It discriminates those features that make sense for individual components [3]. The model GEQUAMO (Generic, Multilayered and Customizable Model) by E.Georgiadou consists of the gradual breakdown into sub layers of features and characteristics and is intended to encapsulate the various user requirements in a dynamic and flexible way. In this form the user (end user, developer, and manager) can build their own model reflecting the emphasis (weight) for each attribute and / or requirement [3]. Alvaro model considers a framework for the certification of software components in order to establish the elements of quality components [3]. Rawashdeh Model has as main objective the needs of different types of users. It has been influenced by the ISO 9126 and Dromey [3]. The models are either product oriented (GECUAMO), or for particular domains (Bertoa) or adapted from the point of view of a user (Rawashdeh) [3].

Models for assessing the quality of Free Software products adapt models like ISO-9126, adding some particular aspects of Free Software. The ideal model that captures all aspects of quality in a free software product has not been defined yet [3]. CapGemini Open Source Maturity Model is based on the maturity of the product and is set according to maturity indicators [3]. OpenBRR Model was influenced by the CapGemini and ISO 9126. In this context identifies categories that are important for evaluating open software. It has seven categories and thereby accelerates the evaluation process, ensuring better choices with a small set [3]. SQO-OSS is a hierarchical model that evaluates the source code and the community process allowing automatic calculation of metrics. It differs from others in the following aspects: focus to the automation; is the core of a continuous quality monitoring system and allows automatic metrics collection; it doesn't evaluate functionality; it focuses on source code; considers only the community factors that can be automatically measured [3]. The QualOSS model states that quality is highly depending on the context in which it is used an the purposes that a company pursues with it [3].

Table 1 shows the main characteristics and subcharacteristics of the Basic, Tailored quality models and models for assessing the quality of Free software.

TABLE 1

CHARACTERISTICS AND SUBCHARACTERISTICS OF SOFTWARE QUALITY MODELS

№	*Software quality model*	*Characteristics and subcharacteristics*
1	*2*	*3*
1	McCall	Correctness: traceability, completeness, consistency. Reliability: accuracy, error tolerance. Efficiency: execution efficiency, storage efficiency. Integrity: access control. Usability: operability, training, communicativeness. Maintainability: simplicity, conciseness, self-descriptiveness. Testability: simplicity, instrumentation, self-descriptiveness, modularity. Flexibility: self-descriptiveness, expandability, generality. Portability: self-descriptiveness, software system independence, machine independence. Reuseability: machine independence, self-descriptiveness. Interoperability: modularity, communication commonality, data commonality

1	*2*	*3*
2	Boehm	Portability: device independence, self-contentedness. Reliability: self-contentedness, integrity, accuracy. Efficiency: accountability, accessibility. Human engineering: accessibility, communicativeness. Testability: structured-ness, self-descriptiveness, accountability, accessibility, communicativeness. Understandability: legibility, conciseness, structured-ness, self-descriptiveness. Modifiability: structured-ness, augment-ability
3	Dromey	Functionality. Reliability. Maintainability. Efficiency. Reuseability. Portability
4	FURPS	Functionality: joint of characteristics, capacities, security. Usability: human factors, aesthetic, documentation of the user, material of training. Reliability: frequency and severity of failures, recovery to failures, time among failures. Performance: velocity, efficiency, availability, time of answers, time of recovery, utilization of resources. Supportability: testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, instability, localizability
5	ISO 9126	Functionality: suitability, accuracy, interoperability, security, functionality compliance. Reliability: maturity, fault tolerance, recoverability, reliability compliance. Usability: understandability, learnability, operability, attractiveness, usability compliance. Efficiency: time behavior, resource utilization, efficiency compliance. Maintainability: analyzability, changeability, stability, testability, maintainability compliance. Portability: adaptability, installability, co-existence, replaceability, portability compliance. Productivity. Safety. Satisfaction
6	ISO 25010	Functional Suitability: functional appropriateness, functional correctness, functional completeness. Reliability: maturity, availability, fault tolerance, recoverability. Performance efficiency: time-behavior, resource utilization, capacity. Compatibility: co-existence, interoperability. Usability: appropriateness recognisability, learnability, operability, user error protection, user interface aesthetics, accessability. Security: confidentiality, integrity, non-repudiation, accountability, authencity. Maintainability: modularity, reuseability, analyzability, modifiability, testability. Portability: adaptability, installability, replaceability
7	Bertoa	Functionality: accuracy, suitability, interoperability, compliance, security. Reliability: maturity, suitability. Usability: learnability, understandability, operability. Efficiency: time behavior, resource behavior. Maintainability: changeability, testability. Portability: replaceability
8	GEQUAMO	Functionality. Usability: learnability, understandability, consistency
9	Alvaro	Functionality: accuracy, security, suitability, interoperability, compliance, self-contained. Reliability: recoverability, fault tolerance, maturity. Usability: configurability, understandability, learnability, operability. Efficiency: time behavior, resource behavior, scalability. Maintainability: stability, changeability, testability. Portability: deployability, replaceability, adaptability, reuseability
10	Rawashdeh	Functionality: accuracy, security, suitability, interoperability, compliance, compatibility. Reliability: recoverability, maturity. Usability: understandability, learnability, operability, complexity. Efficiency: time behavior, resource behavior. Maintainability: changeability, testability. Manageability
11	CapGemini Open Source Maturity Model	Integration: modularity, collaboration. Usability. Performance. Reliability. Security. Platform independence. Vendor independence. Proven technology. Support. Interfacing. Reporting. Administration. Advise. Training. Staffing. Implementation
12	OpenBRR Model	Functionality: compliance. Operational: transferability, security, usability. Support: service, training, consulting service. Documentation. Software Technology: portability, integration, modular and flexible. Community and adoption. Development process
13	SQO-OSS Model	Maintainability: analyzability, changeability, stability, testability. Reliability: maturity, effectiveness. Security. Community Quality
14	QualOSS Model	Robustness and Elvovability. Maintainability. Security. Availability. Repeatability. Interactivity. Adequacy. Capability of requirements and change management. Capability of release management. Capability of support and community management

The research of software quality models has shown, that nowadays they (especially Tailored Quality Models) weakly consider the systemic aspect of modern software. In particular the insufficient attention is paid to emergent properties of software (that are an integral feature of the system). The software quality models (especially Tailored quality models) are aimed at the evaluation and improvement of the quality of the separate components. But the systems analysis shows that the functions of the system is not the simple sum of the functions of system components, and the presence of the emergent (integrative) properties are one of the most important features of the system [27, 28]. So, the use of systems analysis is the prospective tool for development of the new software quality model with the ability of detection and evaluation of the software emergent properties.

IV. The Share of Non-Functional Requirements in the Software Quality Models

The non-functional requirements include the degree of internal indeterminancy (degree of subjectivity) [29] and are formulated at the system level [29-31] therefore they reflect some of the emergent properties of software systems. The list of non-functional requirements according to [32] are: quality, complexity, performance, maintainability, safety, reliability, security, interoperability, dependability.

Then the set of the software non-functional requirements is: $SEP = \{sep_1, \ldots, sep_9\}$, where sep_1 - quality, sep_2 - complexity, sep_3 - performance, sep_4 - maintainability, sep_5 - safety, sep_6 - reliability, sep_7 - security, sep_8 - interoperability, sep_9 - dependability.

The set of the software quality model is: $SQM = \{QM_{McCall}, QM_{Boehm}, QM_{Dromey}, QM_{FURPS}, QM_{9126}, QM_{25010}, QM_{Bertoa}, QM_{GEQUAMO}, QM_{Alvaro}, QM_{Rawashdeh}, QM_{CapGemini}, QM_{OpenBRR}, QM_{SQO-OSS,} QM_{QualOSS}\}$

where QM_{McCall} - McCall model, QM_{Boehm} - Boehm, QM_{Dromey} - Dromey, QM_{FURPS} - FURPS, QM_{9126} - ISO 9126, QM_{25010} - ISO 25010, QM_{Bertoa} - Bertoa, $QM_{GEQUAMO}$ - GEQUAMO, QM_{Alvaro} - Alvaro, $QM_{Rawashdeh}$ - Rawashdeh, $QM_{CapGemini}$ - CapGemini Open Source Maturity Model, $QM_{OpenBRR}$ - OpenBRR, $QM_{SQO-OSS}$ - SQO-OSS, $QM_{QualOSS}$ - QualOSS. Each software quality model is the set of characteristics and subcharacteristics that listed in Table1.

The software quality model should provide the ability of prediction of the largest possible number of the non-functional requirements with purpose of the reduce of their negative impact of the emergent properties. Each quality model considers not all but only some of the non-functional requirements. The following rules are developed with purpose of the evaluation of the quality models regarding the consideration of non-functional requirements:

$sc_{QM_{McCall}} = sc_{QM_{McCall}} + 1 \ \text{if}\ (sep_i \in QM_{McCall},\ i = 1..9,\ QM_{McCall} \subset SQM)$;

$sc_{QM_{Boehm}} = sc_{QM_{Boehm}} + 1 \ \text{if}\ (sep_i \in QM_{Boehm},\ i = 1..9,\ QM_{Boehm} \subset SQM)$;

…

$sc_{QM_{QualOSS}} = sc_{QM_{QualOSS}} + 1 \ \text{if}\ (sep_i \in QM_{QualOSS},\ i = 1..9,\ QM_{QualOSS} \subset SQM)$.

According to these rules, we will obtain the number of the non-functional requirements that can be considered and evaluated by the software quality model – Table 2:

TABLE 2

Number of the Non-Functional Requirements that Are Considered and Evaluated by the Software Quality Models

Non-functional requirements	Software Quality Models (according to Table 1)													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Quality	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Complexity										x				
Performance				x		x					x			
Maintainability	x		x	x	x	x	x		x	x			x	x
Safety					x									
Reliability	x	x	x	x	x	x	x		x	x	x		x	
Security				x	x	x	x		x	x	x	x	x	x
Interoperability	x				x	x	x		x	x				
Dependability														
Total score	4	2	3	5	6	6	5	1	5	6	4	2	4	3

So: $sc_{QM_{McCall}} = 4$, $sc_{QM_{Boehm}} = 2$, $sc_{QM_{Dromey}} = 3$, $sc_{QM_{FURPS}} = 5$, $sc_{QM_{9126}} = 6$, $sc_{QM_{25010}} = 6$, $sc_{QM_{Bertoa}} = 5$, $sc_{QM_{GEQUAMO}} = 1$, $sc_{QM_{Alvaro}} = 5$, $sc_{QM_{Rawashdeh}} = 6$, $sc_{QM_{CapGemini}} = 4$, $sc_{QM_{OpenBRR}} = 2$, $sc_{QM_{SQO-OSS}} = 4$, $sc_{QM_{QualOSS}} = 3$.

Thus, Table 2 shows that all models ignore the important non-functional requirement "dependability".There are three software quality models that can consider and evaluate the largest number of non-functional requirements. These models are: ISO 9126, ISO 25010, Rawashdeh model. Rawashdeh model is Tailored quality model that aims to detection and improvement of the quality of the individual components and weak considers the systemic aspect of software. ISO 9126 model is somewhat outdated model, ISO 25010 model is an updated version of ISO 9126 model, so in future we will use the basic ISO 25010 model. But this model considers and evaluates only 6 non-functional requirements of 9 possible.

Non-functional requirements reflect some of the software emergent properties, so we should maximize the ability of detection of non-functional requirements with the purpose of reducing the negative impact of emergent properties during the software exploitation.

In addition, more attention should be paid to the consideration of subject domain information in the software quality models (for example, in the selected ISO 25010 model). Nowadays the subject domain information already affects the finished product, that can increases the size of the knowledge gap during the life cycle and can cause the software accidents and disasters. The reducing of the knowledge gap size with the purpose of the safe exploitation of the software can achieve by the increase of the share of the considered subject domain information in the software quality models.

V. CONCLUSION

In this paper the analysis of the known approaches to detection of emergent properties of software system and the analysis of the known software quality models were done. The results of this analysis testify that the significant share of current software quality problems is caused by the insufficient attention to the software emergent properties and to the consequences of their display.

There has been active research in the explaining the nature of emergence of the emergent properties, but these properties are insufficiently formalized for their modeling and prediction.

The analysis of software quality models has shown, that they (especially Tailored Quality Models) are aimed at the evaluation and improvement of the quality of the separate components, and weakly consider the systemic aspect of modern software, and the insufficient attention is paid to emergent properties of software (that are an integral and essential feature of the system).

The conducted analysis of the ability of consideration and evaluation of the software non-functional requirements (that reflect some of the emergent properties) by the known software quality models shows, that: 1) all models don't consider the non-functional requirement «dependability»; 2) there are three software quality models that consider and evaluate the largest number of non-functional requirements: ISO 9126, ISO 25010, Rawashdeh model; 3) these three models consider and evaluate only 6 non-functional requirements of 9 possible.

Further research will focus on: 1) the improvement of the software quality models in terms of the consideration and evaluation of the non-functional requirements, and in terms of the consideration of the subject domain information; 2) the identification of the share of emergent properties that are reflected in the non-functional requirements; 3) the increase of the share of the emergent properties that can be reflected in the non-functional requirements by means of the software quality models.

REFERENCES

[1] CHAOS Manifesto: Think big, act small – The Standish Group International: CHAOS Knowledge Center, 2013 – 52 p.

[2] T.Ishimatsu. Hazard analysis of complex spacecraft using systems-theoretic process analysis / T.Ishimatsu, N.G. Leveson, J.P.Thomas, C.H.Fleming, M.Katahira, Yu.Miyamoto, R.Ujiie, H.Nakao, N.Hoshino // *Journal of Spacecraft and Rockets* - Vol. 51, No. 2 (2014), pp. 509-522

[3] J.P.Miguel. A review of software quality models for the evaluation of software products / J.P.Miguel, D.Mauricio, G.Rodríguez // *International Journal of Software Engineering & Applications (IJSEA)*, 2014 - Vol.5, No.6, pp.31-54

[4] O.Gordieiev. Evolution of Software Quality Models: Green and Reliability Issues / O.Gordieiev, V. Kharchenko, M.Fusani // *Proceedings of the 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer* – CEUR-WS, vol.1356, pp.432-445 // http://ceur-ws.org/Vol-1356/

[5] O.Pomorova. The modern problems of software quality evaluation / O.Pomorova, T.Hovorushchenko // *Radioeletronic and computer systems* – Kharkiv: NAU "KhAI", 2013 – № 5, pp.319-327 (in Ukrainian)

[6] T.Hovorushchenko The analysis of the field of software quality evaluation // *Transactions of National university "Lviv polytechnic" "Computer*

systems and networks" – Lviv: Publ. house of NULP, 2013 – №773, pp.41-48 (in Ukrainian)

[7] O.Pomorova, T.Hovorushchenko. The Way to Detection of Software Emergent Properties // Proceedings of the 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2015)

[8] D.Niebuhr. Towards reliable self-integrative IT systems / D.Niebuhr, C. Peper, A. Rausch // *Proceedings of the Tenth International Workshop on Component-Oriented Programming* - July 2005

[9] Christopher W. Johnson. What are Emergent Properties and How Do They Affect the Engineering of Complex Systems? // *Complexity in Design and Engineering* - Volume 91, Issue 12, 2006, pp. 1475-1481

[10] Patterson Jr F.G. Life cycles for system acquisition // *Encyclopedia of Life Support Systems, Systems Engineering and Management for Sustainable Development*. – 2004. – pp. 82-110

[11] I. Allison. Software Process Improvement as Emergent Change: a Structurational Analysis / I. Allison, Y. Merali // *Journal Information and Software Technology* – Vol. 49 Issue 6, June, 2007 – pp. 668-681

[12] S.Kaisler. Complex Adaptive Systems:Emergence and Self-Organization / S.Kaisler, G.Madey // *Tutorial Presented at HICSS-42 Big Island* - January 5, 2009

[13] Gerald E. Marsh. The demystification of emergent behavior // *SAO/NASA ADS*, 2009

[14] John C. Hsu. Emergent Behavior of Systems-of-Systems / John C. Hsu, M.Butterfield // *Proceedings of Mini-Conference INCOSE-2009*

[15] C.Rouff. Properties of a formal method for prediction of emergent behaviours in swarm-based systems / C.Rouff, M.Hinchey, W.Truszkowski, A.Vanderbilt, J.Rash // *NASA Technical Documents*, 2004

[16] Essay Emergence: The Connected Lives of Ants, Brains, Cities, and Software Emergence by Steven Johnson // *Paperback*, 2002

[17] David A. Fisher. An Emergent Perspective on Interoperation in Systems of Systems // *TECHNICAL REPORT CMU/SEI-2006-TR-003*, 2006 – 67 p.

[18] P.Kashfi. Models for Integrating UX into Software Engineering Practice: an Industrial Validation / P.Kashfi, R.Feldt, A.Nilsson, R.Berntsson Svensson // *Software Engineering*, 2014

[19] Roger T. Alexander. Challenges of Aspect-oriented Technology / Roger T. Alexander, James M. Bieman // *Proccedings Workshop on Software Quality of ICSE 2002* – pp.1-3

[20] C.Szabo. Semantic Validation of Emergent Properties in Component-Based Simulation Models / C.Szabo, Y.M.Teo // *Ontology, Epistemology, & Teleology for Model. & Simulation* - Springer-Verlag Berlin Heidelberg, 2013, pp. 319–333

[21] J.C.Mogul. Emergent (mis)behavior vs. complex software systems //*ACM SIGOPS Operating Systems Review*, 2006 – Vol. 40, №4, pp. 293-304

[22] R.Guillerm. Safety evaluation and management of complex systems: A system engineering approach / R.Guillerm, H.Demmou, N.Sadou // *Concurrent Engineering: Research and Applications, SAGE Publications (UK and US)*, 2012, 20 (2), pp.149-159

[23] V. Vyatkin, Software Engineering in Industrial Automation: State of the Art Review // *IEEE Transactions on Industrial Informatics*, 2013 – Vol. 9, № 3, pp. 1234-1249

[24] ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models / ISO/IEC, 2011

[25] M. Klas. Adapting Software Quality Models: Practical Challenges, Approach, and First Empirical Results / M. Klas, C.Lampasona, J.Munch // *37th EUROMICRO Conference on Software Engineering and Advanced Applications* - 978-0-7695-4488-5/2011, IEEE, pp. 341-348

[26] C.Ayala Selection of third party software in Off-The-Shelf-based software development — An interview study with industrial practitioners / C.Ayala, Hauge, Øyvind, R.Conradi, X.Franch, Li Jingyue // *The Journal of Systems and Software*, 2010, pp 24-36

[27] N.Wiener. Cybernetics: Or Control and Communication in the Animal and the Machine. 2-nd ed. - Massachusetts: MIT Press, 1961 – 328 p.

[28] M.Mesarovic. General Systems Theory: Mathematical Foundations / M..Mesarovic Y. Takahara - Moscow: Mir, 1978 - 344 p. (in Russian)

[29] R.Guillerm. Information model for model driven safety requirements management of complex systems / R.Guillerm, H.Demmou, N.Sadou // *Complex Systems Design & Management*. – Springer Berlin Heidelberg, 2010 – pp. 99-111

[30] M.Larsson. Predicting Quality Attributes in Component-based Software Systems - Department of Computer Science and Engineering, Mälardalen University, Västerås, Sweden – 218 p.

[31] N.J.Pizzi. Mapping Software Metrics to Module Complexity: A Pattern Classification Approach // *Journal of Software Engineering and Applications*, 2011 – Vol. 4, pp. 426-432

[32] P.Bourque. Guide to the software engineering body of knowledge (SWEBOK). Version 3.0 / P.Bourque, R.E.Fairley - A project of the IEEE Computer Society, 2014 – 335 p.