The Way to Detection of Software Emergent Properties

Tetiana Hovorushchenko,

Doctor, Senior Researcher, Associate Professor, Lecturer of System Programming Department, Khmelnitsky National University

Oksana Pomorova,

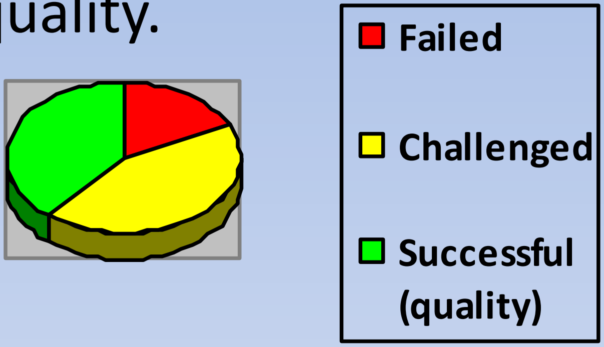
Full Doctor, Professor, Head of System Programming Department, Khmelnitsky National University

Plan of Report

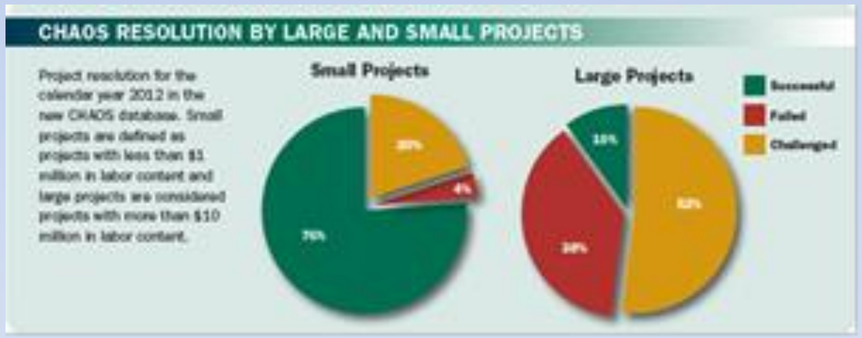
1. Introduction: Actuality of Research; Research Purpose
2. Emergent properties and consequences of their display for the software. Approaches to detection of emergent properties of software systems
3. Software quality models
4. The share of non-functional requirements in the software quality models
5. Impact of the subject domain information on the knowledge gap size
6. Conclusions
7. Questions & Discussion

1. Introduction: Actuality of Research; Research Purpose

The software quality is essential factor for its successful implementation and exploitation. The modern software is not ideal in terms of its quality.



According to the modern definition of quality (as the extent of compliance the present characteristics and requirements [SWEBOK’2014]), only 39% software projects are quality.


CHAOS RESOLUTION BY LARGE AND SMALL PROJECTS

Project resolution for the calendar year 2012 in the new CHAOS database. Small projects are defined as projects with less than $1 million in labor content and large projects are considered projects with more than $10 million in labor content.

According CHAOS Manifesto report, 76% small projects are quality, but only 10% large projects are quality.

The most significant cause of the low quality of the large software projects - is the **increasing of the number of components (subsystems) and interfaces between them** [CHAOS Manifesto report, The Standish Group International].

The next causes of the low software quality:

1) uncontrolled complexity;
2) weakly consideration of the systemic aspect of modern software by the software quality models and methodologies: **the insufficient attention to software emergent properties (integral feature of the system)**;
3) the **insufficient attention to the subject domain information** at the different stages of the software life cycle.

The **research purposes** are:

1) the detection of imperfections of the software quality evaluation (by the insufficient attention to the software emergent properties detection, evaluation and prediction; by the insufficient attention to the subject domain information during the software life cycle);
2) the analysis of the known approaches of the software emergent properties detection by the use of the software quality models;
3) the analysis of the ability of consideration and evaluation of the non-functional requirements in the software quality models

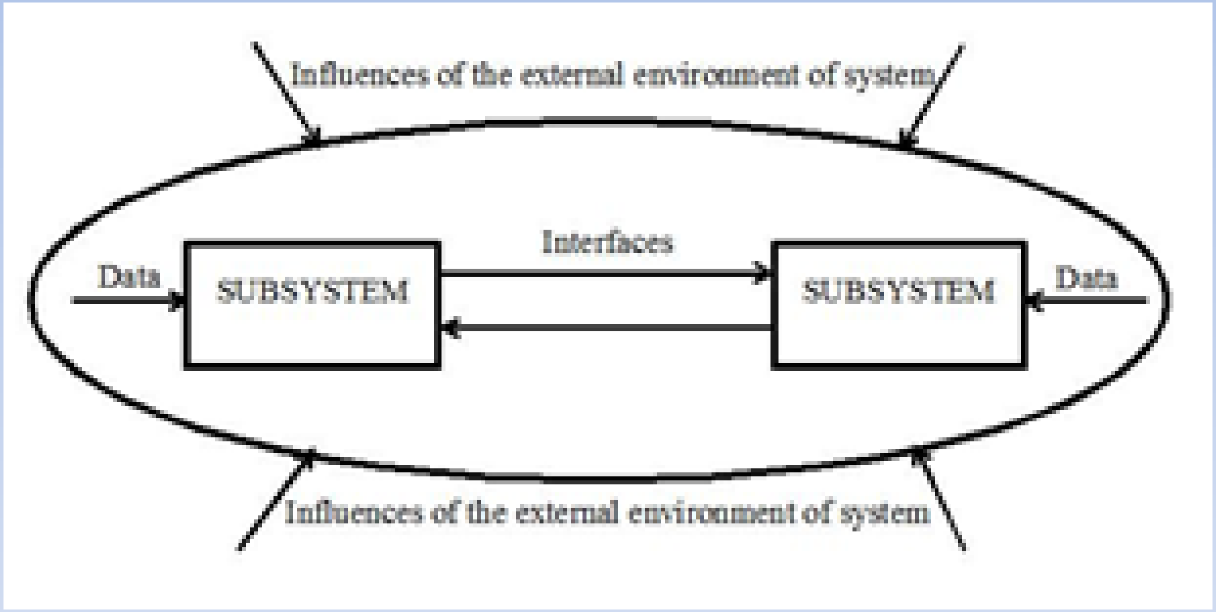
2. Emergent properties and consequences of their display for the software

Definition of the terms "emergent property", "emergent behavior"	Authors
An emergent property is any characteristic of a system that cannot be localized to a single independently acting constituent or to a small constant number of constituents. Emergent properties arise from the cumulative effects of the local actions and neighbor interactions of many autonomous entities.	David A. Fisher
Complex systems often exhibit properties that are not easily predictable by analyzing the behavior of their individual, interacting components. These properties, called emergent properties, are increasingly becoming important as software systems grow in complexity, coupling, and geographic distribution	C.Szabo, Y.M.Teo
The emergent properties are the divergence between intended and realized design. These properties cannot be known a priori.	I.Alison, Y.Merali
Quality attributes of software will often require putting a number of functions together (i.e. emergent properties)	P.Kashfi and other
Emergent properties are characteristic of complex systems. Systems of sufficient complexity with typically have properties that can't be explained by breaking the system down into its elements (emergent properties)	S.Kaisler, G.Madey

Definition of the terms "emergent property", "emergent behavior"	Author
The emergent properties of the collective whole do not in any transparent way derive from the underlying rules governing the interaction of the system's components	G.Marsh
Properties of the whole that are the result of interaction of the parts, or that are the result of addressing the collection of parts as a whole, are emergent properties. Emergent properties derive from the combination of constituent parts of systems	F.Patter-son
Emergent properties of software are requirements that cannot be addressed by a single component but that depend on how all the software components interoperate. Emergent properties are crucially dependent on the system architecture	SWEBOK' 2014
Emergent properties can be thought of as unexpected behaviors that stem from interaction between the components of an application and their environment. There is considerable disagreement about the nature of emergent properties: any unexpected properties exhibited by a complex system; or an application exhibits behaviors that cannot be identified through functional decomposition.	C.W.Joh-nson
Complex systems often behave in unexpected ways that are not easily predictable from the behavior of components – emergent behavior	Jeffrey C. Mogul

Definition of the terms "emergent property", "emergent behavior"	Author
Certain behaviors of a system of systems arise from the interactions among the individual systems and are not embodied in any of the individual systems – emergent behavior	A.Dorofe eand other
Emergent behavior is that which cannot be predicted through analysis at any level simpler than that of the system as a whole. Emergent behavior is what's left after everything else has been explained	G.Dyson
Emergent behaviors are characteristics that arise from the cumulative actions and interactions of the constituents of a system-of-systems. Emergent behavior is inversely proportional to the degree of bondage between systems (the more tightly the component systems are coupled the less likely that the global emergent behavior will prevail; emergent behaviors do not arise in closed hierarchically structured systems), is non-linear and self-organized	J.C.Hsu, M.Butter -field
The set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components is emergent behavior	C.Rouff and other

We will assume that **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:


Influences of the external environment of system
Interfaces
Data
SUBSYSTEM
SUBSYSTEM
Data
Influences of the external environment of system

We will assume that **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 software incidents and accidents today include the new type of accident, that caused by the components interaction: **each component has no defects, but incorrect interaction between components leads to the problems:**

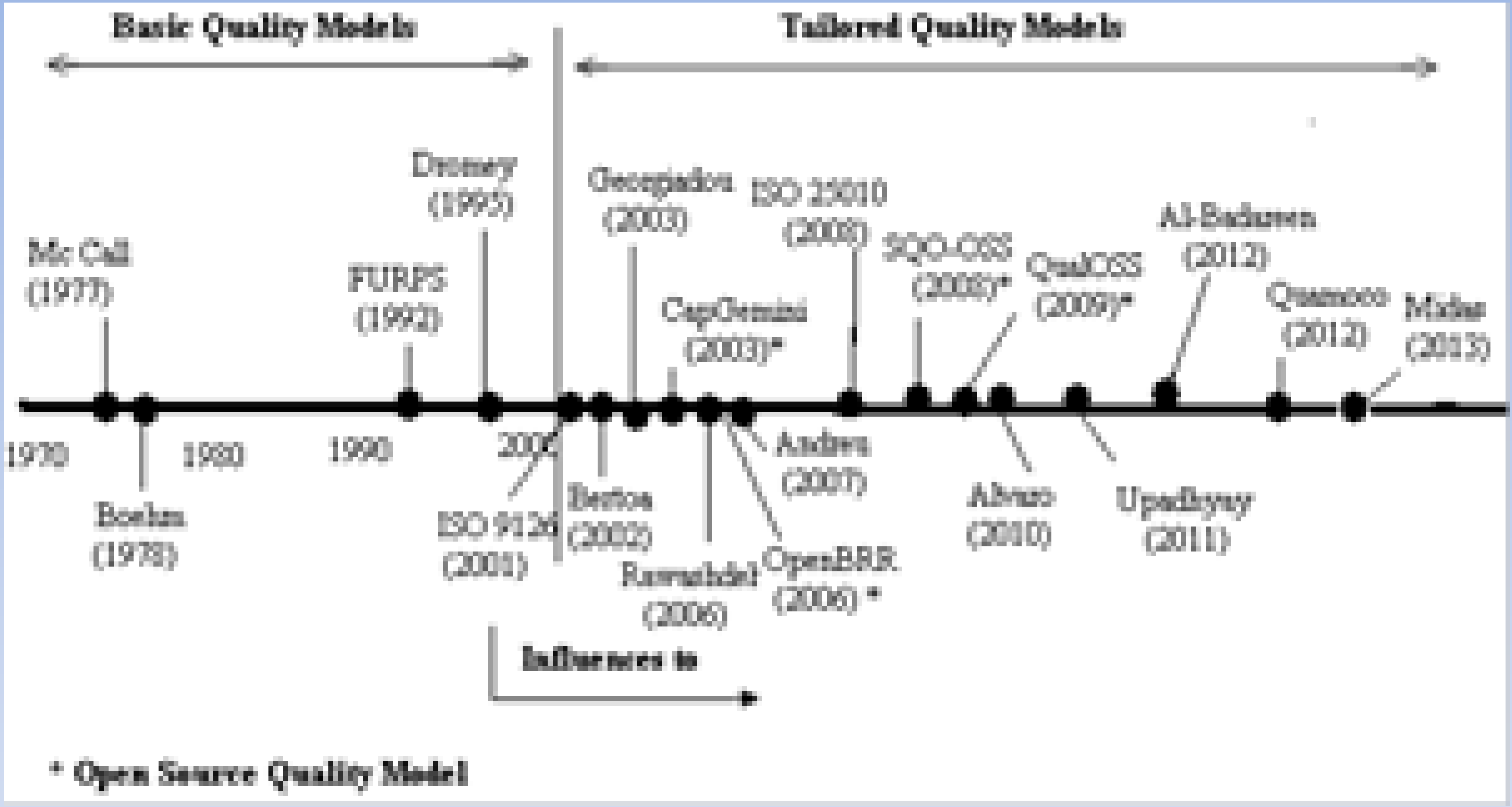
Event	Cause	Consequences
Explosion of rocket Ariane 5 in 1996	Mismatch of requirements to ensure reliability and maximum allow-able load	The cost of equipment and development - 7.5 $ billion, "lost profits" - 2 $ billion
Accidents of Mars Climate Orbiter and Mars Polar Lander in 1999	Defects in the project through the use of different measurement units	327.6 $ million – apparatus, 91.7 $ million - launch
Violation of flight of launch vehicle of Titan IV in 1999	Error in software of management system of motor	The loss of the satellite Milstar
The crash of plane «Superjet 100»	Incompatibility and inconsistency of software	The death of 48 people

Event	Cause	Consequences
"Death" sessions of radiation therapy with Therac-25 in 1985-1987 [27]	Incompleteness of SRS; defects in the development and formulation of the project; incorrect assessment and prediction of risks	6 patients received a lethal radiation dose
Failure in mobile system of missile defense "Patriot" in 1991 [27, 28]	Rounding error that was not critical at the level of a single component, but intensified during its integration into the system	28 American soldiers were killed and about 100 people were injured
Falling into the Pacific Ocean of three satellites in 2010 [27, 29]	Error in software system integration	The impossibility of completing the GLONASS

Today there are many **approaches to detection of emergent properties of software systems**:

1) the chronological approach as part of Software Process Improvement (SPI);
2) the comparison of software systems with natural groups with emergent behavior - for example, a group of migratory birds or ant colonies;
3) approaches and techniques from biological and social systems, physical sciences;
4) multi-agent formal methods: WSCCS, X-Machines;
5) UX-aware model for software requirements;
6) the use of the aspect-oriented programming technology;
7) semantic validation of emergent properties in component-based simulation models;
8) the use of service-oriented architectures (SOAs);
9) on the basis of the information model on SysML language;
10) on the basis of the use of the game theory;
11) the research agenda to deal with emergent misbehavior in complex software systems;
12) <u>the use of system analysis</u>.

3. Software quality models



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 the software emergent properties**.

Software quality model	Characteristics and subcharacteristics
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
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
Dromey	Functionality. Reliability. Maintainability. Efficiency. Reuseability. Portability
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
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
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

Software quality model	Characteristics and subcharacteristics
Bertoa	Functionality: accuracy, suitability, interoperability, compliance, security. Reliability: maturity, suitability. Usability: learnability, understandability, operability. Efficiency: time behavior, resource behavior. Maintainability: changeability, testability. Portability: replaceability
GEQUAMO	Functionality. Usability: learnability, understandability, consistency
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
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
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
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
SQO-OSS Model	Maintainability: analyzability, changeability, stability, testability. Reliability: maturity, effectiveness. Security. Community Quality
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

4. The share of non-functional requirements in the software quality models

The non-functional requirements include the degree of internal indeterminacy (degree of subjectivity) and are formulated at the system level therefore they reflect some of the emergent properties of software systems.

The non-functional requirements [SWEBOK’2014]:

- quality
- complexity
- performance
- maintainability
- safety
- reliability
- security
- interoperability
- dependability

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 on Slide 10)**													
	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

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 (only 6 of 9 possible): ISO 9126, ISO 25010, Rawashdeh model.

5. Impact of the subject domain information on the knowledge gap size

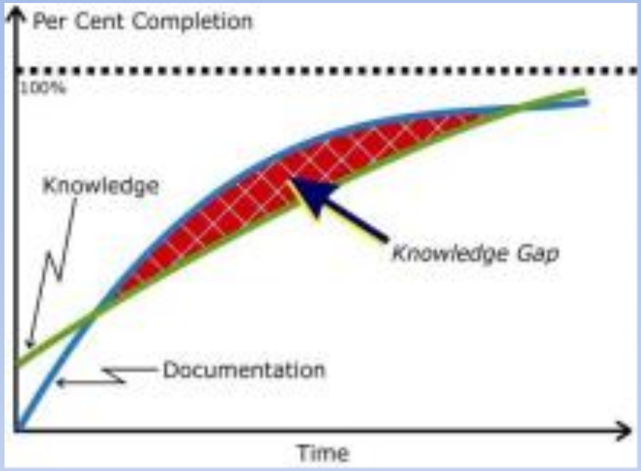


Figure 1. The knowledge gap

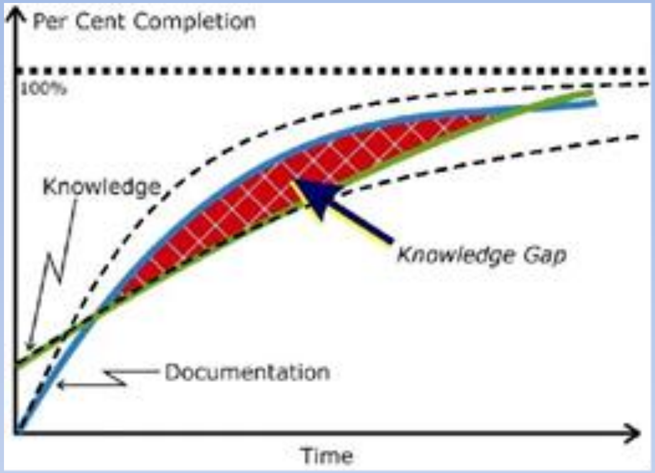


Figure 3. The real size of the knowledge gap with non-consideration of subject domain information in the software quality models (NOW)

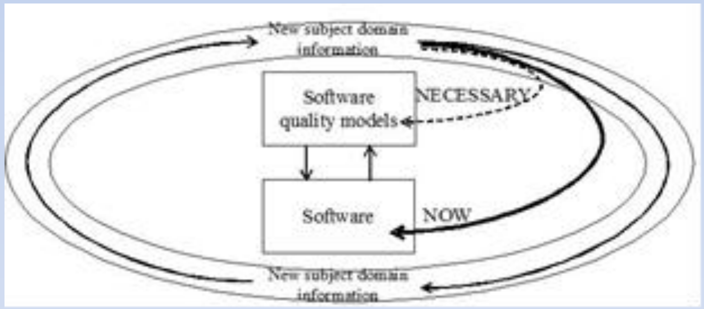
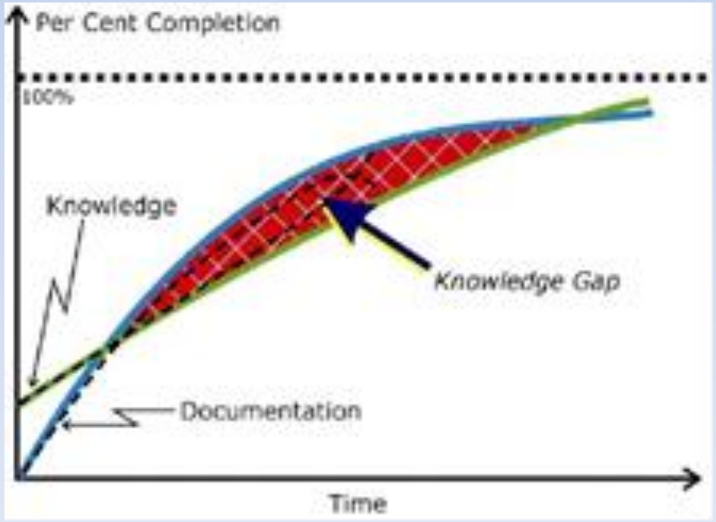

New subject domain information
Software quality models
NECESSARY
Software
NOW
New subject domain information

Figure 2. The detection of the emergent properties by consideration of the subject domain information in the software quality models

Figure 4. Knowledge gap with the consideration of the subject domain information in the software quality models (NECESSARY)



6. Conclusions

- The analysis of the known approaches to detection of emergent properties of software system and the analysis of the known software quality models testify that **the significant share of current software quality problems is caused by the insufficient attention to the software emergent properties**, to the consequences of their display and to the subject domain information that appears during software project life cycle.
- There has been active research in the explaining the nature of emergence of the emergent properties, but **the software emergent properties are insufficiently formalized for their modeling and prediction**.
- **The effective methods and approaches for software emergent properties detection in principle were not developed**. In general we need the fundamentally new approaches related to the full consideration of the subject domain information in the software quality models.
- The software quality models: don't consider the non-functional requirement «dependability»; consider and evaluate maximum 6 of 9 possible non-functional requirements (only 3 models).

- **The software quality models** (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 the software emergent properties**.
- **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.
- During software projects development there is gap of knowledge about the developed software characteristics. This **knowledge gap emerges by reason of the partial consideration of the subject domain information in the software quality models**. **The size of knowledge gap is not constant for software project.** The probability of disappearance of the knowledge gap during the software project life cycle is low, and the appearance of new subject domain information leads to increasing of the knowledge gap size.
- **For safe software functioning the knowledge gap size is desirable to reduce with consideration of maximum subject domain information in the software quality models**.

Our 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;
4) the development of methodology of software quality evaluation based on the development of methods and technologies of detection, evaluation and prediction of the software emergent properties on the basis of system analysis.

7. Questions & discussion

Thank you for attention! Questions, please!

Tetiana Hovorushchenko:

tat_yana@ukr.net

+38-095-11-22-544

skype: tat_yana_13

Oksana Pomorova:

o.pomorova@gmail.com

+38-067-384-17-50

skype: oksana.pomorova