PSEUDO-CODE SIMULATION OF DESIGNER ACTIVITY IN CONCEPTUAL DESIGNING OF SOFTWARE INTENSIVE SYSTEMS

Petr Sosnin
Computer Department
Ulyanovsk State Technical University
Ulyanovsk, 432027, Russia
E-mail: sosnin@ulstu.ru

KEYWORDS

Conceptual designing, pseudo-code, simulation.

ABSTRACT

The paper presents an experiential approach to a real time activity of designers at a conceptual stage of their work. The offered approach is based on a pseudo-code simulation of such activity. The simulation are being implemented by the designer who investigate own actions aimed at the conceptual solution of the project task.

INTRODUCTION

An important feature of designing the Software Intensive Systems (SIS) is a collaborative activity of designers in conditions of a high complexity. Negative influence of human factors in such a kind of the activity is an essential reason of an extremely low degree of a success (about 35%) in designing the SIS (El Emam and Koru 2008).

In general sense the complexity (or simplicity) of SIS reflects the degree of a difficulty for designers in their interactions with definite models of SIS (or its components) in solving the definite tasks.

The system or its any component is complex, if the designer (interacting with the system) does not have sufficient resources for the achievement of the necessary level of understanding or achieving the other planned aims.

Often enough various interpretations of Kolmogorov measure (Li and Vitanui 2008) are applied for estimations of a degree of the system complexity. This measure is connected with the minimal length of program P providing the construction of system S from its initial description D. Distinctions in interpretations are caused usually by features of system S and formal descriptions to be used for objects P and D.

In accordance with their destinations the programs of P-type it is more preferable to build as programmable systems (P-systems) of designers' activity with using the certain metaprogramming M.

Explicit and/or implicit means for managing the designers' activity are used in any technology of designing the SIS. An actuality of such managing demonstrates that the complexity of P-program is no less than the complexity of SIS in its any used state. Moreover, any M-program providing the construction of P-program should be built on the base of the same initial description D as the system S.It can be presented by the following chain $D \rightarrow M \rightarrow P \rightarrow S$.

Named relations between D, M, P can be used by designers for disuniting the process of designing on stages $[D(t_0) \rightarrow M_1 \rightarrow P_1 \rightarrow S(t_1)]$, $[D(t_1) \rightarrow M_1 \rightarrow P_1 \rightarrow \rightarrow S(t_2)]$,…, $D(t_i) \rightarrow M_1 \rightarrow P_1 \rightarrow S(t_{i+1})]$, …., $[D(t_{n-1}) \rightarrow \rightarrow M_1 \rightarrow P_1 \rightarrow S(t_n)]$ where a set $\{S(t_i)\}$ collects the states of creating SIS.

Division of designing on stages is a base of any modern technology providing the creation of SIS. In technologies such manner is used in different forms for different aims. This manner helps to decrease the complexity of interactions with SIS in its any state $S(t_i)$. But till now the viewpoint of programming on the designer activity is not being supported instrumentally in early stages of designing.

It is necessary to note that a creation of instrumental means for the explicit work with artifacts of M- and P-types essentially depends on their understanding. This paper presents the empirical approach to such artifacts which are like experimental setups helping to find the solutions of project tasks and opening the possibility for their confirmatory reuse. The offered approach is based on the use of a specialized toolkit WIQA (Working In Questions and Answers) providing the pseudo-code simulation of the designer activity (Sosnin 2012).

PRELIMINARY STATEMENTS

The offered approach focuses on the designer activity at the conceptual stage of designing the SIS when any project task should be conceptually solved and registered in the form which is suitable for the reuse in the computerized medium (because collaborative designing in life cycle of SIS).

We consider that any task solution should give an opportunity to any member of designers' team for repeating the activity of the designer who has solved this task in the conceptual form.

One of possible such forms is a protocol of designer actions registered in a natural language L^A in its algorithmic usage. Creating of protocols and their using in repeatable solutions of tasks can be implemented similarly creations and executions of programs. In this case a function of the language L^A can be fulfilled by the executable pseudo-code language L^P helping to create pseudo-code programs for corresponding protocols.

Any protocol in any its form and in the program form also is a model of the designer activity. The value of models of such a type is being defined by activity units registering in protocols. As minimum the registered

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

units should provide understanding of the conceptual solution and checking its conformity to requirements.
In the offered approach an achievement of indicated goals is being provided by experiments of the designer researching a certain task situation. The designer creates the necessary experimental setup the use of which demonstrates the conceptual solution of the project task. The experimental setup is created as a pseudo-code program (model) of the designer activity for its researching. Any model of such type consists of program operators each of which registers the corresponding action of the designer.
Thus experiment steps as program operators are being registered by the designer in the solution protocol for the corresponding project task. Hence any protocol is a plan (program, model) which should be accessible for interested designers in a rational and suitable form. It should open for designers an opportunity for step by step performing the programmed plans in order to understand and check their conceptual solutions for corresponding tasks. In such actions any designer will actively interact with personal and collective experience and also with models of useful experience units (or shortly with the accessible experience). Any operator of indicated programs registers a certain "trace" of designer interactions with the accessible experience.
Certainly, it is possible to create many other models of the designer activity. Even in the offered approach the described model of the program type is a part of integrated model of a precedent type underlining the behavioral nature of the designer activity. In accordance with Cambridge dictionary "precedents are actions or decisions that have already happened in the past and which can be referred to and justified as an example that can be followed when the similar situation arises" (http: //dictionary.cambridge.org/dictionary/british/precedent). Thus any action corresponding to the certain program operator can be also interpreted as the precedent.
Models of precedents are used in the offered approach as structural units of the accessible experience in any its forms. Moreover any task solved by designers is being included to the SIS project and/or to processes of designing as the model of the corresponding precedent. Therefore interactions with the accessible experience in the context of the use of precedents occupy an essential place in our version of programming the designer activity.
Interactions of a human with personal experience reflect their own existence in the dialog forms (question-answer forms) in consciousness. A question is a natural phenomenon which appears when a human should use or wants to use the experience. If the phenomenon of such type is revealed by the human the revealed question can be described in the natural language. Any of such description is no more than a linguistic (symbolic) model of the question. Interactions with the symbolic model of the question can renew the phenomenon of this question in brain structures of designers for its reuse.
The adequacy of any model of any question should be carefully tested because such a model is used for creating the necessary answer. The definite question and corresponding answer are bound as "cause and effect". They supplement each other. The answer form depends on the use of the answer.
Questions have different types. A very important class of questions is tasks. Any statement of any task is its symbolic model also. There are several useful forms of the answer for the tasks, for example, "method of task solving", "solution idea", "conceptual solution" and "programmed solution".
Thus questions and answer are used not only for interactions with the experience. They can be used, for example, for the access to the appropriate unit of experience or for modeling the unit of experience or for reasoning in different cases. Any human has the rich experience of using the questions and answers and their models.
It is necessary note that presented statements and their described understanding have been used in a programmable simulation of the designer activity described below.

RELATED WORKS

Registering the human activity in program forms has been offered and specified constructively for Human Model Processor (MH-processor) in the paper (Karray et al. 2008). The EPIC version of MH-processor is oriented on programs written in the specialized command language Keystroke Level Model (KLM). A set of basic KLM actions includes the following operators: K – key press and release (keyboard), P – point the mouse to an object on screen, B – button press or release (mouse), H – hand from keyboard to mouse or vice versa and others commands. Operators of KLM-language and their values help to estimate temporal characteristics of human interactions for alternative schemes of interfaces. KLM-programs are far from the sense of used reasoning and therefore they do not reflect interactions with the accessible experience.
Explicit programmable forms of the designer activity are not used in modern technologies of SIS designing. For example in technologies based on Rational Unified Process (Borges et al. 2012) the conformity to requirements and understandability are being reached with the help of "block and line" diagrams expressed in the Unified Modeling Language (UML). The content of diagrams built by designers is being clarified by necessary textual descriptions. But UML is not the language of the executable type and therefore diagrams are not suitable for experimenting with them as with programs of P-type.
For collaborative solving the tasks in coordination the RUP suggests the means of normative workflows the relations between which are being regulated by a set of rules. For any task of the definite normative workflow the RUP has its interactive diagrammatic model with a set of components the use of which can help in solving the task. Forms of programming are not used also in all of these means. The similar state of affairs with conceptual designing exists in other known technologies supporting the development of SIS.
In the offered approach its scientific point of view correlates with two faces of the software engineering described in (Cares et al. 2006) where functional

paradigms and scientific paradigms are discussed. In the context of this paper the approach means are oriented on scientific paradigms used by software engineers.
An empirical line of the approach inherits understanding the place and role empirical methods in software engineering generally presented in (Sjoberg 2007). It is necessary to mark numerous papers of V. Basili (especially papers (Basili et al. 2001) and (Basili 2012).
The important group of related works concern means of Question-Answering, for example, papers (Webber and Webb 2010) and (Xu and Rajlich 2005). In this group the nearest work presents experience-based methodology "BORE" (Henninger 2003) where question-answering is applied also but for the other aims and this methodology does not support programming of the intense designer activity.

QUESTION-ANSWER INTERACTIONS WITH ACCESIBLE EXPERIENCE

In the suggested approach the designers interact with all tasks being solved in the WIQA environment in accordance with the scheme presented in figure 1.

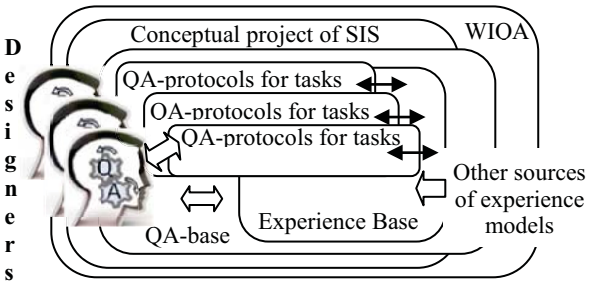

Figure 1: Experiential interactions with tasks

Solving any appointed task, the designer registers the used question-answer reasoning (QA-reasoning) in a specialized protocol (QA-protocol) so that this QA-protocol can be used as the task model (QA-model).

Models of such a type can be used by designers for experimenting in the real time with tasks being solved. Units of the experiential behavior extracted from solution processes are being modeled on the base of QA-models of tasks. Created behavioral models are being loaded in the question-answer database (QA-base) and Experience Base of WIQA. After that they can be used by designers as units of the accessible experience. Experience models from the other sources can be uploaded in the Experience Base also.
If designers of SIS use the toolkit WIQA they have the opportunity for conceptual modeling the tasks of different types. In this case the current state of tasks being solved collaboratively is being registered in QA-base of the toolkit and this state is visually accessible in forms of a tree of tasks and QA-models for corresponding tasks. The named opportunity is presented figuratively in figure 2 where QA-base is interpreted as a specialized QA-memory the cells of which are visualized by inquiries of designers. First of all, the cells are used for storing the registered units of QA-reasoning.
Any cell has the following basic features:

1. Cell is specified by a set of normative attributes reflecting, for example, the textual description of the stored interactive object, its type and unique name, the name of its creator, the time of last modification and the others characteristics.

2. Designer has the possibility to appoint to the cell a number of additional attributes if it will be useful for the work with the object stored in the cell.

Having chosen necessary attributes the designer can adjust the cell for storing any question or any answer in a form of an interactive object which is accessible by inquiries as designers so programs. Thus any question and its answer are stored in QA- memory as a pair of related interactive objects named below as QA-unit.

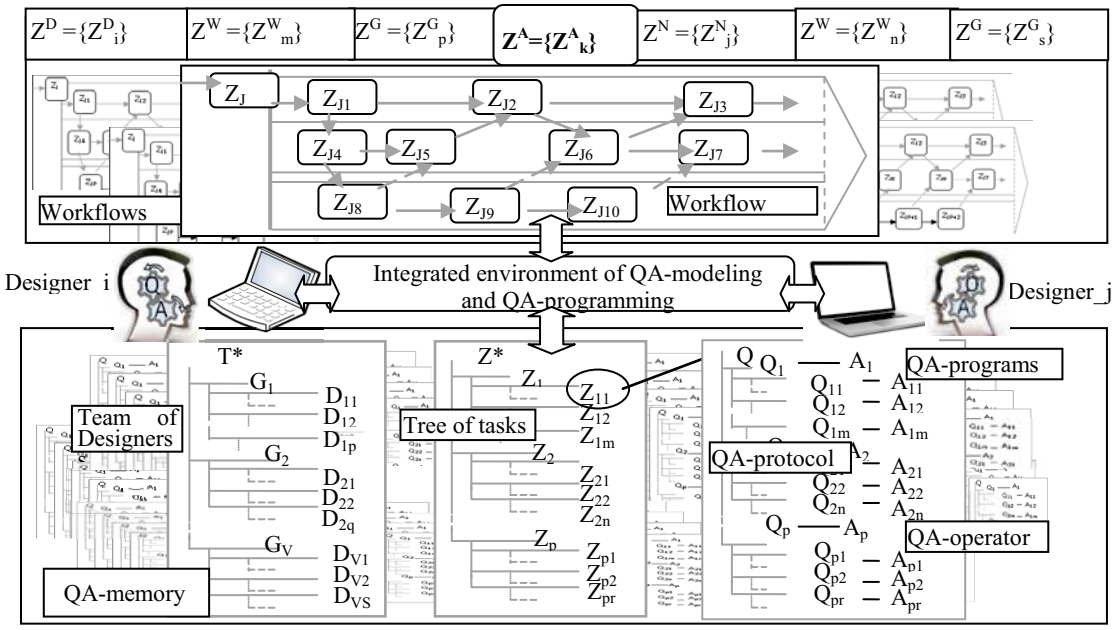

Figure 2: Sinking of tasks in WIQA-environment

QA-units are stored in QA-memory as data the abstract type of which will be named as QA-data. The use of this type helps to emulate other data types, including descriptions of operators. First of all it is necessary for the use of QA-memory in pseudo-code programming. Thus cells of QA-memory destined for storing QA-units can be adjusted for storing the units of the other nature, for example, units used in solving the tasks.

In figure 2 the scheme of QA-memory demonstrates the store of presentations for "Team of designers", "Tree of tasks" and a pseudo-code program with its operators and data. The program, its operators and used data are designated as QA-program, QA-operators and QA-data to underline that they inherit the features of QA-memory cells.

The responsibility for tasks being solved is being distributed among designers in accordance with the competence of each of them. The team competence should be sufficient for the real time work with following sets of tasks: subject tasks $Z^S = \{Z^S_i\}$ of SIS subject area; normative tasks $Z^N = \{Z^N_j\}$ of technology used by designers; adaptation tasks $Z^A = \{Z^A_j\}$ providing an adjustment of tasks $\{Z^N_j\}$ for solving the tasks $\{Z^S_i\}$; workflow tasks $\{Z^W_m\}$ providing the works with tasks of Z^S-type in workflows $\{W_m\}$ in SIS; workflow tasks $\{Z^W_n\}$ providing the works with tasks of Z^N-type in corresponding workflows $\{W_n\}$ in the used technology; workflow tasks $\{Z^G_p\}$ and $\{Z^G_r\}$ any of which corresponds to the definite group of workflows in SIS or in technology.

The indicated diversity of tasks emphasizes that designers should be very qualified specialists in the technology domain but that is not sufficient for successful designing. Normative tasks are invariant to the SIS domain and therefore designers should gain certain experience needed for solving the definite tasks of the SIS subject area. The most part of the additional experience is being acquired by designers in experiential learning when tasks of Z^S-type are being solved. Solving of any task Z^S_i is similar to its expanding into a series on the base of normative tasks.

Objects uploaded to QA-memory are bound in hierarchical structures. In their real time work the designers interact with such objects. They process them with the help of appropriate operations helping to find and test the solution of tasks.

One way for conceptual solving any task indicated types is based on creating its QA-model as a system of questions and answers which have accompanied the solution process. The generalized scheme of such models is presented in figure 3.

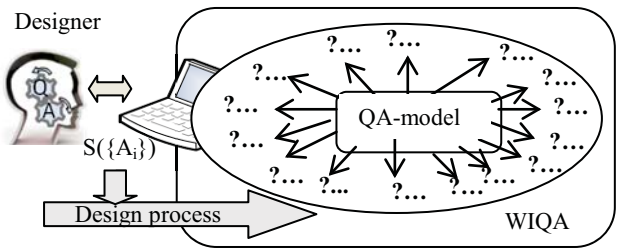

Figure 3: Interactions with QA-model of Task

Question-answer models, as well as any other models, are created for an extraction of answers to the questions enclosed in the model. Moreover, the model is a very important form of the representation of questions, answers on which are being generated during visual interactions of designers with this model.

The description of any behavioral unit composed of designer interactions with QA-model in accordance with the definite scenario can fulfill the role of a model of such designer activity. In order to distinguish this type of models from other types of models they can be named "QA-models of the designer activity". Any such scenario as a specific program reflects designer interactions (actions) aimed at understanding the corresponding task and its solution.

The other way of coding the designer activity is bound with its programming in the context of the scientific research of the task. All tasks indicated above are being uploaded to QA-memory with the rich system of operations with interactive objects of Z-, Q- and A-types. Designers have the possibility to program the interactions with necessary objects. Such programs are similar to the plans of the experimental activity in conceptual designing of SIS. Operators of programs are placed in Q-objects. Corresponding A-objects are used for registering the facts or features of executed operations.

Thus, experimenting with units of the own behavior the designer has a flexible means for specifying the QA-programs, QA-operators and QA-data used in simulating of such behavior's units. Experimenting is being fulfilled in forms of QA-modeling aimed at solving tasks in conceptual designing.

PSEUDO-CODE LANGUAGE L^{WIQA}

QA-reasoning can be used by designers when they create different conceptual models of tasks, for example, in formulating the task statement or in cognitive analyzing of the formulated statement or in (pseudo-code) programming the solution plan of the task. The toolkit WIQA supports the creative work of designers with all indicated conceptual modes and conceptual models of the other types.

So the specialized pseudo-code language L^{WIQA} has been developed for the use of QA-reasoning in programming the solution plans. This language is oriented on its use in experiential interactions of designers with the accessible experience when they create programs of the own activity and investigate them. Step by step L^{WIQA} has been evolved till the state included following components:

1. **Traditional types of data** such as scalars, lists, records, sets, stacks, queues and the other data types.

2. **Data model** of the relational type describing the structure of database.

3. **Basic operators** including traditional pseudo-code operators, for example, Appoint, Input, Output, If-

Then-Else, GOTO, Call, Interrupt, Finish and the others operators.

4. **SQL-operators** in their simplified subset including Create Database, Create Table, Drop Table, Select, Delete From, Insert Into, Update.

5. **Operators for managing the workflows** oriented on collaborative designing (Seize, Interrupt, Wait, Cancel and Queue).

6. **Operators for visualization** developed for the creation of dynamic view of cards presenting QA-units in the direct access of the designer to objects of QA-memory.

The important type of basic operators is an explicit or implicit command aimed at the execution by the designer the definite action. Explicit commands are being written as imperative sentences in the natural language in its algorithmic usage. When designer interactions with descriptions of questions or answers are used as causes for designer actions then such descriptions can be interpreted as implicit commands written in L^{WIQA}. For example, questions are a very important class of implicit commands.
In general case QA-program can include data and operators from different enumerated subsets. But traditional meaning of such data and operators is only the one side of their content. The other side is bound with attributes of QA-units in which data and operators are uploaded. As told above QA-data and QA-operators inherit the attributes of corresponding cells of QA-memory. They inherit not only attributes of QA-units but their understanding as "questions" and "answers" also.

SIMULATING THE DESIGNER'S BEHAVIOR

The principal feature of the offered approach is an experimental investigation by the designer the programmed own behavior which has led to the conceptual solution of the appointed task. Any solution of such a type should demonstrate that its reuse meets necessary requirements when any designer of the team will act in accordance with QA-program of the investigated behavior.
In order to achieve it the designer should work similarly to the scientist who prepare and conduct experiments with behavior units of M- or P-types. In the discussed case the designer will experiment in the environment of the toolkit WIQA. In this environment to prove achieving the aims of any experiment the designer has possibilities of experimenting with any QA-operator of investigated QA-program and/or with any group of such QA-operators or with QA-program as a whole. Describing the experiment for the reuse the designer should register it in an understandable form for other members of the team.
To begin the definite experiment the initial text of QA-program should be built. In general case such work includes the following steps:

1. Formulation of the initial statement of the task.

2. Cognitive analysis of the initial statement with the use of QA-reasoning and its registering in QA-memory

3. Logical description of "cause-effect relation" reflected in the task.

4. Diagrammatic presentation of the analysis results (if it is necessary or useful).

5. Creation of the initial version of QA-program.

Indicated steps are being fulfilled by the designer with the use of the accessible experience including the personal experience and useful units from Experience Base of WIQA.
Only after that the designer can conduct the experiment, interacting with QA-program in the context of the accessible experience. The specificity of interactions can be clarified on examples of QA-operators of any QA-program or its fragment, for example, the fragment of QA-program coding the well-known method of SWOT-analysis (Strengths, Weaknesses, Opportunities, and Threats):

```
Q 2.5 PROCEDURE &SWOT main&
   Q 2.5.1 &t_str& :=
QA_GetQAText(&history_branch_qaid&)
   Q 2.5.2 SETHISTORYENTRIES(&t_str&)
   Q 2.5.3 CALL &ShowHistory&
   Q 2.5.4 IF &LastHistoryFormResult& == -1 THEN
RETURN
   Q 2.5.5 IF &LastHistoryFormResult& == 0 THEN
&current_action_qaid& :=
QA_CreateNode(&current_project&,
&history_branch_qaid&, 3, "") ELSE
&current_action_qaid& := &LastHistoryFormResult&
   Q 2.5.6 &t_str& :=
QA_GetQAText(&current_action_qaid&)
   Q 2.5.7 SWOT_DESERIALIZE(&t_str&)
   Q 2.5.8 &t_int& := SWOT_SHOWMAINFORM()
   Q 2.5.9 &t_str& := SWOT_GETSERIALIZED()
   Q 2.5.10 IF &t_int& == 0 THEN
QA_UpdateNode(&current_project&,
&current_action_qaid&, &t_str&)
   Q 2.5.11 IF &t_int& != 0 RETURN
   ……..
Q 2.5.14 FINISH
```

This source code demonstrates a habitual syntax but features of the code are being opened in interactions of the designer with it. Conditions and means of experimenting are shown in figure 4, where one of operators (with address name *Q2.5.2*) is shown in the context of previous and subsequent operators. Any QA-program is being executed by the designer step by step any of which is aimed at the corresponding QA-operator. In this work the designer uses the plug-in "Interpreter" embedded to the toolkit.

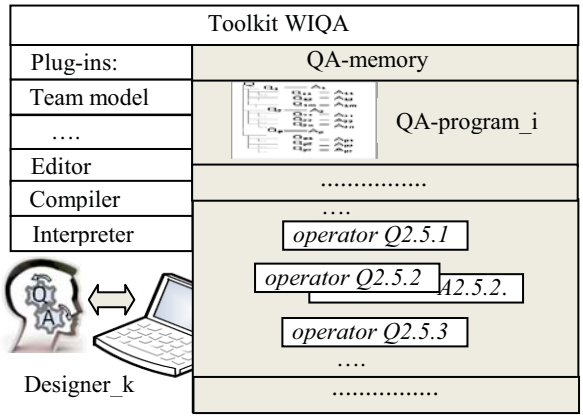

Figure 4: Experimenting of designer with QA-program

Interpreting the current operator (for example, *Q2.5.2*), the designer can fulfill any actions till its activation (for example, to test existing circumstances) and after its execution (for example, to estimate the results), using any means of the toolkit WIQA. When the designer decides to start the work with QA-operator this work can include different interactive actions with it as with corresponding QA-units or with their elements. The designer can analyze values of their attributes and makes useful decisions

Moreover, the designer can appoint the necessary attributes for any QA-operator and for any unit of QA-data in any time. In accordance with appointments the designer can include changing in the source code of QA-program being executed (investigated). Such work can be fulfilled as in QA-memory so with the help of the plug-in "Editor".

The current QA-program or its fragment can be executed or step by step by the designer or automatically as a whole with the help of the plug-in "Compiler". Therefore all aforesaid about the work with QA-operator can be used for any their group and for any QA-program as a whole. That is why the execution of QA-operator by the designer is similarly experimenting. Thus the designer has a flexible possibility for the experimental research of any task being solved conceptually. This is the principal feature which distinguishes pseudo-code QA-programs from programs written in pseudo-code languages of different types including the class of Domain Specific Languages (Karsai G. et al).

The specificity of the described kind of the designer activity is the work controlled by QA-program executed by the designer interacting with the accessible experience. To underline this specificity the specialized role named "intellectual processor" (I-processor) was defined and supported in the use of WIQA (Sosnin 2012). This role is additional for other kinds of roles used in conceptual designing (Borges et al. 2012).

As told above any fulfilled experiment should be presented by the designer in the understandable and reusable form. In the offered version of experimenting the function of such form is being fulfilled by the typical integrated model of the precedent shown in figure 5.

The scheme, fulfilling the function of framework F(P) for models of precedents, allows integrating the very useful information accompanying the experiment process in its actions indicated above.

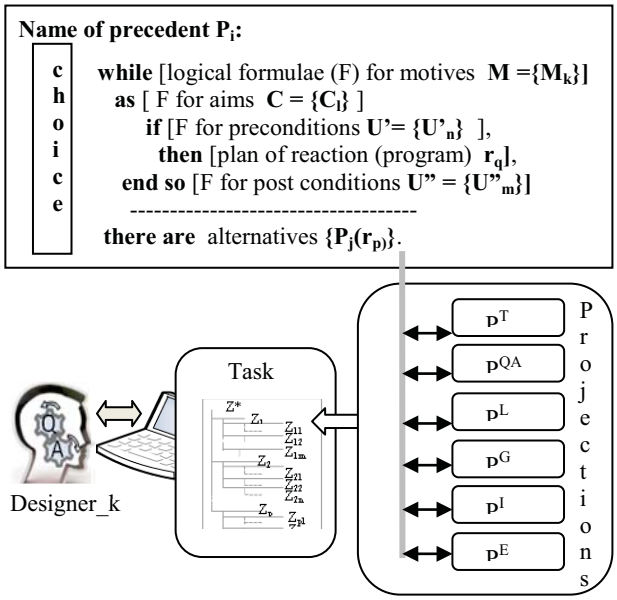

Figure 5: Interactions of designer with precedent

The central place in this model is occupied the logical scheme of the precedent. The scheme explicitly formulates "cause-effect regularity" of the simulated behavior of the designer Framework F(P) includes following components: textual model P^T of the solved task, its model P^{QA} in the form of registered QA-reasoning, logical formulae P^L of modeled regularity, graphical (diagram) representation P^G of precedent, pseudo-code model P^I in QA-program form and executable code P^E. Any component or any their group can be interpreted as projections of F(P), the use of which allow to build the precedent model in accordance with the precedent specificity. But in any case the precedent model should be understandable for its users.

All built models of precedents are divided in two classes one of which includes models embedded in Experience base of WIQA used by the team not only in a current project. The second class includes models only for the current project.

INTELLECTUAL PROCESSOR

Any designer playing the role of intellectual processor uses question-answer reasoning and its models for the access to the experience.

The essence of I-processor is being defined by features enumerated below.

1. First of all I-processor is a role being played by any designer solving the appointed tasks in conceptual designing of the SIS. In life cycle of any SIS its conceptual stage is an area of intensive modeling the tasks being solved.

2. I-processors are intended for experimenting with tasks the solving of which are problematically without the explicit real time access to the personal

experience and/or collective experience and/or models of useful experience.

3. In its activity any I-processor interprets tasks as precedents and interacts with experience units as with models of precedents.

4. The real time work of any I-processor is being accompanied by QA-reasoning and its models being registered in QA-memory of the toolkit WIQA.

5. QA-reasoning is used by I-processor for creating QA-models of tasks in different forms including their versions as QA-programs.

6. The use of QA-reasoning in interactions of I-processor with QA-programs is being implemented in the pseudo-code language L^{WIQA}. Knowing and effective using of this language by I-processor is a very important its feature.

7. In general case the activity of I-processor is similar to the experimental activity of the designer who creates QA-model of the task for experimenting with the prototype of its solution. In such experimenting with tasks I-processor can use as means of QA-modeling so and means of QA-programming.

As told above the use of QA-programs by I-processor for experimenting with tasks is the very essential its feature. That was a cause to create the library of the specialized QA-programs providing some versions of such experimenting. This library includes a number of QA-techniques for cognitive tasks analysis, decision-making and typical procedures of estimations.
It is necessary to note that our materialization of I-processor implements by means of WIQA but it is our solution. In principle requirements to I-processor can be implemented on the base of instrumental means distinct from the toolkit WIQA.

COLLABORATIVE ACTIVITY OF DESIGNERS

Conceptual designing of SIS is the collaborative activity of designers working in coordination. Typical units of such activity are workflows any of which can be simulated with the empirical approach also. In this case experiments will be fulfilled collaboratively by the group of designers playing the roles of I-processors. For QA-programming of workflows the specialized subset of operators has been included to the language L^{WIQA}.
A number of plug-ins is embedded to the toolkit WIQA for supporting the collaborative activity of I-processors. For example, plug-in "Organizational structure" supports the real time appointing of tasks to members of the designer team. The copy of the team model $(T^*,\{G_v\}, \{D_{vs}\})$ can be uploaded in QA-memory in the form presented in figure 2. This plug-in is used not only to appoint tasks. For example, the toolkit supports relations among I-processors through solving of a number of communication tasks in their collaborative experimenting with project tasks. And as told above the toolkit provides the real time management of workflows the tasks of which are being executed by groups of I-processors.

Means of "Organizational structure" support the real time appointments of tasks embedded in workflows to their executors (designers) from the certain group G_v of the team T*. The group manager uses the subsystem "Controlling of assignments" for binding any assignment with planned time of its fulfillment by the designer who is responsible for the assigned task. Subsystem "Kanban" (Wang 2010) automatically reflects the steps of the workflows execution with the help of visualizing the current state of queues of tasks. It helps to control by the process of designing.

Pseudo-code programming of workflows is based on using the library of workflow patterns (Van der Aalst and Hofstede 2004). Coding the units of this library illustrates the example of the workflow pattern "Simple Merge". This pattern is described by the statement: *The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch).*

For three tasks Z_1, Z_2 and Z_3 this pattern has the following view:

```
Q.3.5 PROCEDURE &Simple_Merge&// index name
are inherited also but from library of pattern
Q 3.5.1SET &out&, 4; &ins[0]&, 1; &ins[1]&, 2;
&outgroup[0]&, 1; &outgroup[1]&, 2; &cnt&, 0
Q 3.5.2 LABEL &L1&
Q 3.5.3 SEIZE &outs[&cnt&]&,
&outgroup[&cnt&]&
Q 3.5.4 INC &cnt&
Q 3.5.5 TEST L, &cnt&, &ins&.length &L1&
Q 3.5.6 LABEL &L2_1&
Q 3.5.7 SET &cnt&, 0;
Q 3.5.8 LABEL &L2&
Q 3.5.9 TEST E, &ins[&cnt&]&,
&ins[&cnt&]&.state, DONE &L3&
Q 3.5.10 INC &cnt&
Q 3.5.11 TEST L, &cnt&, &ins&.length &L2_1&
Q 3.5.12 TRANSFER &L2&
Q 3.5.13 LABEL &L3&
Q 3.5.14 QUEUE &out&, TRUE
Q 3.5.15 SET &ins[&cnt&]&.state, WAITING
Q 3.5.16 TRANSFER &L2_1&
Q 3.5.17 ENDPROC &Edit_Assignment&
```

Execution results of similar programs are being placed in Kanban queues which are visualized in WIQA. Any designers interacts with the own queue with the help of "Subsystem of interruption".
The pattern source code also as previous QA-program is presented only for the demonstration of syntax and therefore without explanations. But used specimens of QA-programs are not so difficult for their understanding.

It is necessary to note that programming of workflows leads to QA-programs of M-type presenting the activity of designers on the level of their collaboration. Programs of this type simulate parallel work of designers in workflows. They simulate also pseudo-parallel work of any designer with several appointed tasks.

CONCLUSION

The approach described in this paper has led to the system of means simplifying the complexity arising in the use of the experiential behavior in conceptual designing the SIS. Simplifying is being achieved due to programming of behavior units in the form supposing its reuse in any time by any designer working in the team.

In the centre of the suggested approach is the additional new role named "intellectual processor" the responsibility of which is focused on constructive interactions of the designer with the accessible experience. The activity of I-processor is being supported by the toolkit WIQA providing the opportunity for QA-modeling and QA-programming of the experiential behavior of designers. The language L^{WIQA} embedded to the toolkit allows creating pseudo-code programs of QA-type for tasks corresponding to M- and P-programs to be used in programmed workflows.

There is a possibility for the separate execution of any operator of QA-program by the designer playing the role of I-processor. Before and after execution of the operator the designer can check or investigate its preconditions and post-conditions. In such actions the designer can interact with the accessible experience. Thus there is an opportunity for experimenting with any programmed task of any workflow being executed in conceptual designing of SIS.

Pseudo-code programs of behavior units have very useful features. Debugged QA-programs are the source of resources of the M- and P-types which are not only simplifying the complexity in their reuse. For example, the language L^{WIQA} was used in creating the system of the living project documentation. Suggested means are used in the professional activity of one project organization developing the family of SIS.

The offered approach and its instrumental means can be used not only for conceptual designing. They can fulfill the function of a shell for creating a number of other applications. For example, they have been used in the creation of "Multi-agent system for simulation of surrounding the sea vessel".

REFERENCES

Basili V. R., Lindvall M. and P. Costa. 2001. "Implementing the experience factory concepts as a set of experience bases." In *Proceedings of the 20011 International Conference on Software Engineering & Knowledge Engineering*, 102-109.

Basili V. R. 2013. "Learning through Application, SEMAT position." http://semat.org/wp-content/uploads/2012/03/

Borges P., Machado R.J. and P. Ribeiro. 2012. Mapping RUP Roles to Small Software Development Teams. In *Proceedings of the* 2012 *International Conference on Software and System Process*, 190-199.

Cares C., Franch X. and Mayol E. 2006. "Perspectives about paradigms in software engineering." In *Proceedings of the 2006 2nd International workshop on Philosophical Foundations on Information Systems Engineering* (PHISE'06), 737-744.

Cho Y. 2010. "The state of the art of action learning research," *Advances in Developing Human Resources,* 2 No.12, 163-180.

El Emam K. and A.G. Koru. 2008. "A Replicated Survey of IT Software Project Failures." *IEEE Software* 25 No. 5, 84-90.

Henninger S. 2003. "Tool Support for Experience-based Software Development Methodologies." *Advances in Computers* 59, 29-82.

Karray F., Alemzadeh M., Saleh J. A. and M. N. Arab. 2008. "Human-Computer Interaction: Overview on State of the Art", *Smart sensing and intelligent systems* 1 No. 1, 138-159.

Karsai G, Krahn H., Pinkernell C., Rumpe B., Schindler M. and S. Völkel. 2009. "Design Guidelines for Domain Specific Language," I*n Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling*, 7-13.

Li M. and P. Vitanui 2008. "An Introduction to Kolmogorov Complexity and Its Applications." *Series: Text in Computer Science*, 3rd ed., Springer.

Sjoberg D.I. K., Dyba T. and M. Jorgensen. 2007. "The Future of Empirical Methods in Software Engineering Research." In *Proceedings of the* 2007 *workshop Future of Software Engineering,* 358-378.

Sosnin P. 2012. "Experiential Human-Computer Interaction in Collaborative Designing of Software Intensive Systems," *In Proceedings of the 11th International conference on Software Methodology and Techniques*, 180-197

Van der Aalst W.M.P. and A.H.M. Hofstede. 2004. "Workflow Patterns Put Into Context." *Software and Systems Modeling* 11 No.3, 319-323.

Wang J. X. 2010. "Kanban: Align Manufacturing Flow with Demand Pull." Chapter in the book: Lean Manufacturing Business Bottom-Line Based, CRC Press, 185-204.

Webber, B. and N. Webb. 2010. "Question Answering." In *Clark, Fox and Lappin (eds.): Handbook of Computational Linguistics and Natural Language Processing*. Blackwells.

Xu, S. and V. Rajlich. 2005. "Dialog-Based Protocol: An Empirical Research Method for Cognitive Activity in Software Engineering." *In Proceedings of the 2005 ACM/IEEE International Symposium on Empirical Software Engineering*, 397-406.

PETR SOSNIN was born in Ulyanovsk in the USSR, on July 12, 1945. He graduated from the Ulyanovsk Polytechnic Institute (1968).

His employment experience included the Ulyanovsk Polytechnic Institute and Ulyanovsk State Technical University. His special field of interests includes AI applications for computer aided design. P. Sosnin defended doctor degree in Moscow Aviation Institute (1994). He is an author of eight books and more two hundred articles.