Precedent-Oriented Experimenting in Designing of Software Intensive Systems

Petr Sosnin
Ulyanovsk State Technical University, Ulyanovsk, Russia
Contact(s). sosnin@ulstu.ru

Abstract— The paper presents a precedent-oriented approach to experimenting with programmable units of developer's activity in conceptual designing of Software Intensive Systems (SIS). The reuse of any such a unit is being implemented as a typical work of a designer in accordance with the definite technique which is previously programmed. The offered approach is coordinated with simplifying the complexity on the base of interactions of designers with the accessible experience the kernel of which consists of models of assets included into Experience Base. The simplifying is being achieved by the use of the specialized pseudo-code language in programming of assets for their reuse by designers.

Keywords/Index Terms— conceptual designing, pseudo-code language, programming, precedent-oriented approach, software intensive systems.

1. INTRODUCTION

The successful creation of any SIS in an essential measure depends on what means are used by designers for operated mastering by the system complexity. In general sense the complexity (or simplicity) of SIS reflects the degree of 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. In most general case the complexity or simplicity is a function of three variables – time, accuracy (variety) and volume of information. The complex system is being produced more difficultly, than the simple system.

Often enough various interpretations of Kolmogorov measure (Li & 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 first of all by features of system S, and also what contents are connected with objects P and D and how these contents are being specified.

In creating of SIS the objects of the P-type are being built in step by step into the process of designing with using the certain "method of programming of M". Reality of such a work demonstrates that the complexity of "P-object" no less than the complexity of SIS in its any used state. Moreover, M-program providing the construction of P-object is being 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 $[D(t_0) \rightarrow M_1 \rightarrow P_1 \rightarrow S(t_1)]$, $[D(t_1) \rightarrow M_2 \rightarrow P_2 \rightarrow S(t_2)]$,…, $[D(t_i) \rightarrow M_{i+1} \rightarrow P_{i+1} \rightarrow S(t_{i+1})]$,…., $[D(t_{n-1}) \rightarrow M^n \rightarrow P_n \rightarrow S(t_n)]$ where a set $\{S(t_i)\}$ collects the states of SIS being created.

This division of designing on stages is a base of any modern technology providing the creation of SIS. In technologies such a 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(ti). But till now the viewpoint of programming is not being supported instrumentally at early stages of designing.

This paper presents the experiential approach to supporting the collaborative designing at its early stages when the conceptual project of SIS is being created. The essence of the approach is defined by the explicit work of designers with Mi- and Pi-programs in the creation of which the specialized toolkit WIQA is being used (Sosnin, 2012a). This toolkit can be interpreted as Experience Factory (Basili et al., 2001) with a library of assets programmed in a specialized pseudo-code language oriented on question-answer reasoning. The library of assets is organized as Experience Base (Basili et al., 2006) including the base of precedents' models (Base of Precedents).

2. ESSENCE OF PRECEDENT-ORIENTED APPROACH

The extremely low degree of a success in designing of software intensive systems are an important reason (El Emam & Koru, 2008) for searching the new approaches to the collaborative work in this subject area. The analysis of the successfulness problem indicates that in a search of new ideas and their

technological materializations the special attention should be given to the human factor when the intensive human-computer activity is being fulfilled in conditions of the complexity.

Let us notice that designing (especially conceptual designing) is impossible without researches by designers of numerous and practically unpredictable situations of a task type. In such situations the designer should behave as a researcher who uses the appropriate type of the research in which the personal and collective experience is applied creatively in the real time.

In designing of the definite research the designer would rather work as a scientist who wants to fulfill the definite experiment the results of which will be applied in the creation of the current project and can be useful for the future reuse. Therefore, the search of improved forms of designer interactions with own and collective experience is a promising way for increasing the degree of the success in designing of SIS.

As told above about complexity of SIS it can be operated if designers will build and execute M- and P-programs of their activity. Such programs should present the plans of experiments which should help to solve the project tasks in forms suitable for the future reuse. At the conceptual stages of designing the language for M- and P-programming should be as near as possible to the natural language in its algorithmic usage. In deep our opinion such a language should have a pseudo-code type.

The previous reasoning was used for the specification and materialization of the experiential approach to its applying in the designing of the family of SIS. In this work the standard "Framework for Software Product Line Practice (Version 5.0)" [available at http://www.sei.cmu.edu/ product lines/ tools/framework] has been taken into account. Such an orientation of the approach leads to the question about modeling the assets, first of all, as units of previous experience which can be reused in designing as in the current project so in developments of new "members" of the SIS family.

The precedent-oriented approach to the behavior of designers has following features:

1. The definite set of designer actions are being presented as a program (M- or P-types) the execution of which leads to the creation of the corresponding part of the conceptual project.
2. Programs of actions are built by designers with the help of the pseudo-code language L^{WIQA} (embedded to the toolkit WIQA) which is similar to the naturally professional language in its algorithmic usage. The language L^{WIQA} was specified in details in our paper (Sosnin, 2012a).
3. Pseudo-code programming of designer actions is oriented on their reuse as typical units each of which can be qualified as a precedent. 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). Models of precedents are used in the approach as structural units of the experience in any its forms.
4. Pseudo-code programs not only are being created by designers, but also are being executed by them.
5. In all actions with M- and P-programs the designers fulfill the role of "intellectual processor" in the frame of which they actively interact with personal and collective experience and also with experience models registered in Experience Base.
6. Role of "intellectual processor" (I-processor) is being supported by specialized workflows "Interactions with Experience" (Sosnin, 2012b) opening the possibility for scientifically experimental activity of designers.
7. Activity of I-processor is being implemented by means of question-answer reasoning (QA-reasoning) and its models in order to coordinate the natural forms of the access to the experience (as a natural phenomenon) and the access forms to models of experience.
8. Means of QA-reasoning can be applied by designers as for analyzing so for pseudo-code programming of any project task of any type.

Let us clarify some details of named features. All of them are bound by the behavioral point of view on creating of programs describing the work of designers. Moreover, this point of view is based on the use of precedents as basic units of the behavior. Such units are built and reused as intellectually processed "conditioned reflexes".

Any precedent is appeared in the experiential interaction of a human with surrounding in definite conditions. In general case the acting human wants to achieve the definite aims in the frame of definite motives.

Any project precedent is also the result of intellectual processing of the definite unit of the designing activity. Hence, any project precedent appears as a result of the corresponding "experiment" executed by the designer or a group of designers. Similar "experiment" is being planned and being implemented in the definite conditions for achieving the definite aims of the motivated experimenter(s).

In general case the precedent can be described by the logical scheme presented in figure1. This logical model is a human-oriented scheme the human interaction with which is able to activate the internal

logical process on the level of the second signal system in human brains.

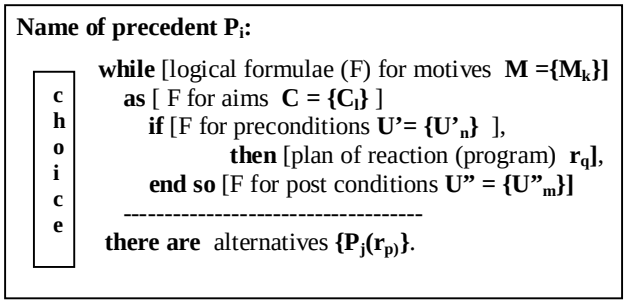

FIGURE 1. LOGICAL SCHEME OF THE PRECEDENT

To understand the developed precedent the designer should try to emulate the behavior embedded to it. Such emulation in the programmed form we bind with the creation and experiential execution of P-programs.

When the precedent is being assembled from other precedents and such a work should be managed it is need the other type of program supporting the flows of works of designer or the group of designers. This type of programs we bind with M-programs. Designers should create programs of this type for technological flows of works, scientific workflows and human workflows.

M-programs also as P-programs should be previously created and tested with using the means of emulation. In the offered approach such work are fulfilled by designers with using the specialized tools supporting the designer role named "intellectual processor" [2] 18Sosnin P..

All named features influence on simplifying of the complexity. As told above any human (designer) estimates the complexity of the definite interaction with artifacts on the base of accessible resources. The library of assets included to Experience Base is a useful source of resources opened for simplifying the complexity.

3. 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., 20080. 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 et al., 2007). It is necessary to mark numerous papers of V. Basili (especially papers (Basili et al., 2001) and (Basili et al., 2006).

The important group of related works concern means of Question-Answering, for example, papers (Webber & Webb, 2004) and (Xu & 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.

An important group of related works is connected with workflows. Executable languages for descriptions of workflows (for example BPEL or YAWL) have some restrictions which prevent in programming of workflow(Van der Aalst & Hofstede, 2004). Their pseudo-code programming has not any restrictions. The necessity of programming for scientific workflows is indicated in publication (Held & Blochinger, 2009) but without practical solutions and suggestion of using the pseudo-code means for such aims.

4. QUESTION-ANSWER MODELING OF DESIGNER ACTIVITY

A. Operating Space of Experimentation

Any experimental research is being implemented in an appropriate medium of experimenting. In described case the role of such a medium fulfills an operating space supported by the toolkit WIQA. The generalized scheme of experimentations in the indicated space is presented in figure2.

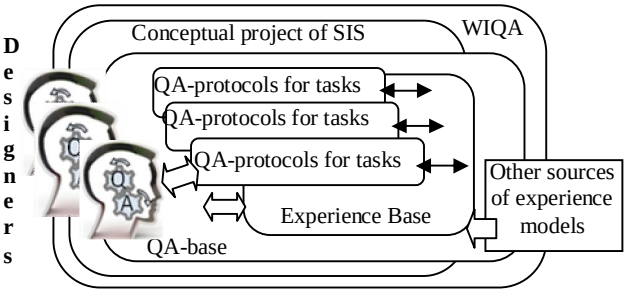

FIGURE 2. INTERACTIONS WITH TASKS

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). Typical units of QA-reasoning are questions (Q) and answers (A) of different types. Tasks are a very important type of questions. Below the tasks will be designated with the use of a symbol "Z".

Models of such a type can be used by designers for experimenting in the real time with all tasks being solved. Units of the experiential behavior extracted from solution processes are being modeled on the base of QA-models of tasks.

The scheme reflects also that the investigated behavior model can be uploaded as the model of the corresponding precedent in the question-answer database (QA-base) and in 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.

B. Question-Answer Memory

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 3 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.

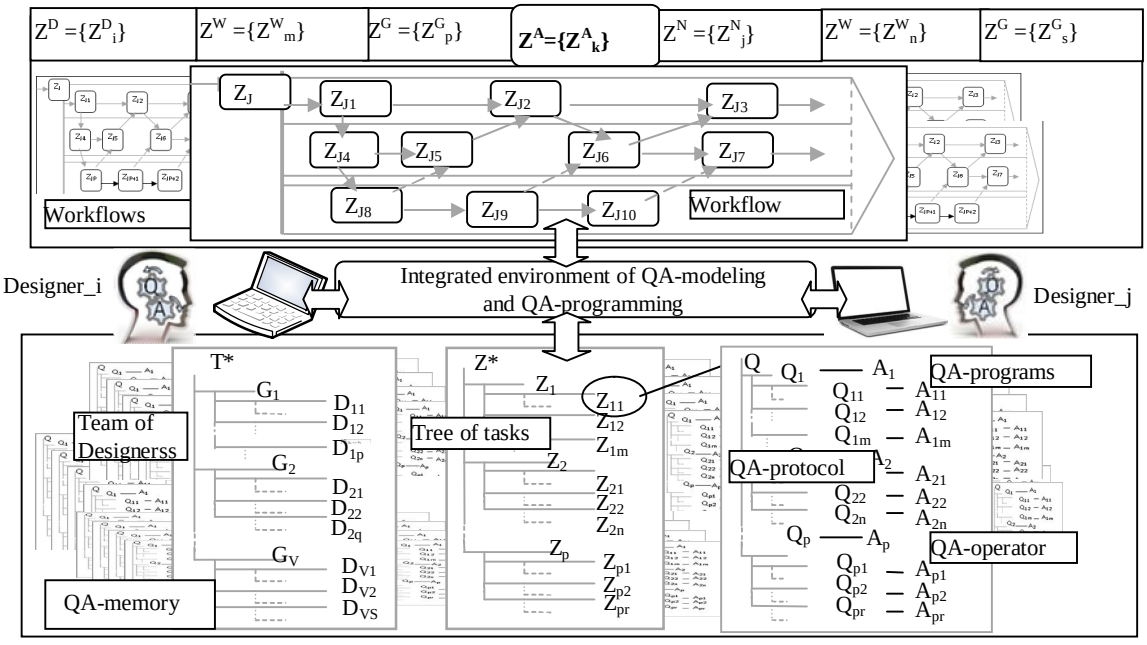

FIGURE 3. SINKING OF TASKS IN WIQA-ENVIRONMENT

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.

the work with the object stored in the cell.

Having chosen necessary attributes the designer can

2. Any cell has a unique address the function of which is fulfilled by the type name of the stored unit and its unique index appointed automatically at creating the unit. Empty cells are absent.

3. Designer has the possibility to appoint to the cell a number of additional attributes if it will be useful for 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.

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, for units used in solving the tasks.

In figure 3 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 the SIS subject area; normative tasks $Z^N = \{Z^N_j\}$ of the technology used by designers; adaptation tasks $Z^A = \{Z^A_k\}$ 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 ZS-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 the 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 ZS-type are being solved in conceptual designing. Solving of any task ZSi 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.

Objects in QA-memory are accessible to designers in accordance with given rights of an access. But in any case any QA-model is accessible to the group of designers who interact with it with different purposes which include checking this model. Thus any QA-model is a product of a collaborative reasoning and coordinated understanding.

C. *Question-Answer Modeling*

One way for conceptual solving any task of 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 a model is presented in figure 4.

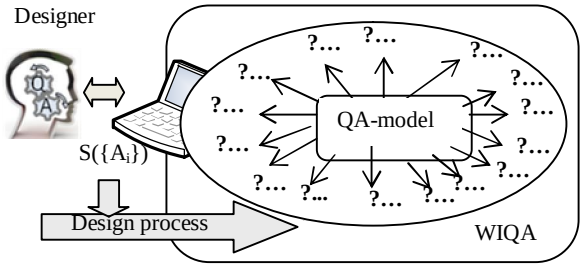

FIGURE 4. INTERACTIONS WITH QA-MODEL OF TASK

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 essence of QA-modeling is interactions of designers with artifacts included to QA-model in their current state. For such an interaction the developer can use the special set of QA-commands, their sequences and a set of WIQA plug-ins.

The main subset of positive effects of QA-modeling includes:

- controlling and testing the reasoning of the developer with the help of "collaborative reasoning" and "integrated understanding" included into the QA-models;
- correcting the understanding of designers with the help of comparing it with "integrated understanding";
- combining the models of the collective experience with an individual experience for increasing the intellectual potential of the designer on the definite workplace;
- including the individual experience of the developer in accordance with the request on the other workplaces in the corporate network.

As it is shown in this scheme any component of QA-model is a source of answers accessible for the designer as results of interactions with this model. At the same time the potential of QA-model is not limited by the questions planned at defining and creating the QA-model. Another source of useful effects of QA-modeling is an additional combinatorial "visual pressure" of questions and answers which is caused by

influence on brain processes in their contact with components of QA-model. In this case there is no difference who has created QA-model.

There are different forms for building answers with the help of QA-modeling, not only linguistic forms. But in any case the specificity of QA-modeling is defined by the inclusion of additional interacting with "question-answer objects" into dynamics of the integrated consciousness and understanding (into natural intellectual activity of designers).

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 a designer activity. In order to distinguish this type of models from other types of models used in our approach they can be named "QA-models of the designer activity". Any such a scenario as a specific program reflects designer interactions (actions) aimed at understanding the corresponding task and its solution. In the discussed case the scenario is a text which consists of instructions indicating the designer actions which should be executed in the reuse of behavioral unit in the WIQA-medium.

Similar scenarios can be created for acting the human not only in the WIQA-medium. Their content, form and appointment are demonstrated by the following technique:

//Reset of Outlook Express

O1. Quit all programs.

O2. Start On the menu Run, click.

O3. Open In the box regedit, type, and then OK the click.

O4. Move to and select the following key:

HKEY_CURRENT_USER/Software/Microsoft/Office /9.0/Outlook

O5. In the Name list, FirstRunDialog select.

O6. If you want to enable only the Welcome to Microsoft Outlook greeting, on the Edit menu Modify, click the type True in the Value Data box, and then OK the click.

O7. If you also want to re-create all sample welcome items, move to and select the following key:

HKEY_CURRENT_USER/Software/Microsoft/Office /9.0/Outlook/Setup

O8. In the Name list, select and delete the following keys: CreateWelcome First-Run

O9. In the Confirm Value Delete dialog box click Yes, for each entry.

O.10. On the Registry menu, click Exit.

O11. End.

This technique is chosen to emphasize the following:

1. There are many behavior units describing the human activity in different computerized mediums.

2. Descriptions of similar typical activities help in the reuse of these precedents.

3. Descriptions of techniques have forms of programs (N-programs) written in the natural language LN in its algorithmic usage.

4. Such N-programs consist of operators being fulfilled by the human interacting with the definite computerized system. In the example of N-program its operators are marked by the symbol "O" with the corresponding digital index.

Thus there are no obstacles for uploading the N-programs in QA-memory. This way is used for uploading the techniques supporting the designer activity in the WIQA-medium.

So 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 a 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 behavioral units. Experimenting is being fulfilled in forms of QA-modeling aimed at solving tasks in conceptual designing.

5. Simulating the Designer's Behavior

A. *Preparing of Experiments*

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.

As told above, 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 a 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.

B. Experimenting with QA-program

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 following 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& := &LastHistoryForm Result&
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 2.5.14 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 5, 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.

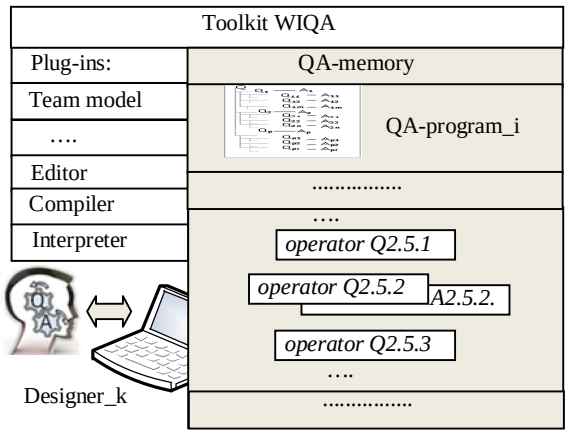

FIGURE 5. EXPERIMENTING WITH QA-PROGRAM

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 of the investigation), 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-ins "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 et al., 2009).

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 "intellectual processor" was constructively defined and effectively being supported in the use of WIQA (Sosnin, 2012). This role is additional for other kinds of roles applied in conceptual designing (Borges et al., 2012).

C. Describing of Experiments

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 a form is being fulfilled by the typical integrated model of the precedent shown in FIGURE 6.

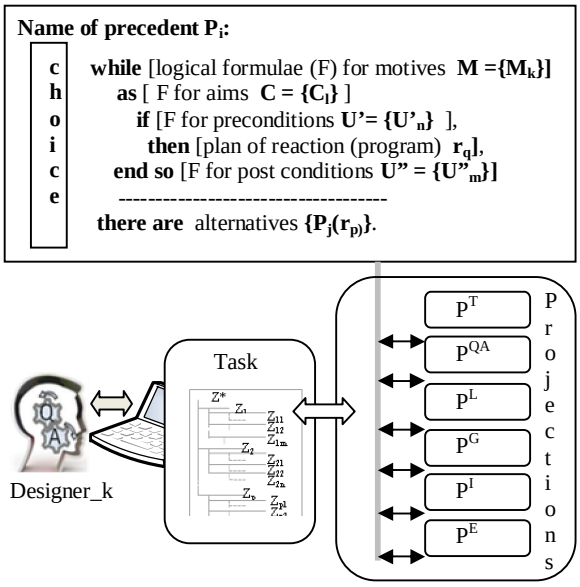

FIGURE 6. FRAMEWORK OF PRECEDENT MODEL

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.

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 PT of the solved task;
- its model PQA in the form of registered QA-reasoning;
- logical formulae PL of modeled regularity;
- graphical (diagram) representation PG of precedent;
- pseudo-code model PI in QA-program form;
- and executable code PE.

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.

Experience base of assets

In the experiential approach the presentation of assets is oriented on the behavior of designers in the asset reuse. Therefore the basic forms for presenting the assets in Experience Factory are QA-models and QA-programs of different types and also their compositions. The kernel of compositions consists of precedents' models combined in Base of Precedents.

Potential of L^{WIQA} is sufficient for QA-modeling and QA-programming the assets of following kinds: previous projects, valuable project solutions, prototypes, documents, interface samples, schemes of reports, standards, frameworks, guides, patterns, samples of different types, schemes of modeling, structure of the software, packages of the source code, tools, platforms, infrastructure and other valuable units.

Models of assets are registered in the catalog of Experience Base and allocated in its corresponding sections (as shown in figure 7). Only one part of assets is placed in Precedent Base. Interaction with accessible assets provides by their catalog implemented as specialized plug-ins of WIQA.

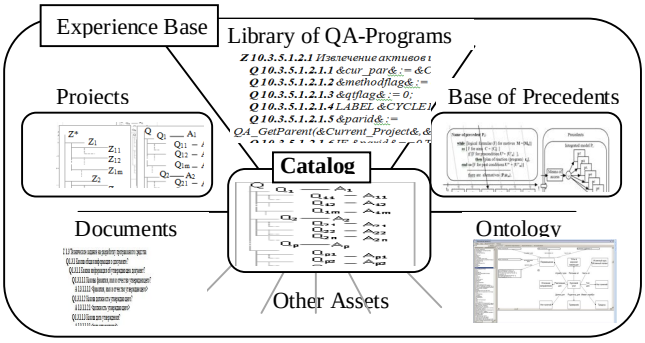

FIGURE 7. STRUCTURE OF EXPERIENCE BASE

The special subsection of Experience Base is assigned for workflow patterns. Any unit of this subsection has a corresponding pseudo-code description. QA-programs of workflow patterns include operators of the specialized subset of L^{WIQA}.

Conclusion

The offered approach is aimed at managing the decrease of the complexity in designing of SIS by the use of real time interactions of designers with the accessible experience. Moreover, such interactions are being programmed so that any created program describes the unit of the designer work in the process of designing. Such a possibility is being provided by means of pseudo-code programming of the designer activity in the language L^{WIQA}. The specificity of pseudo-code programs in this language is being defined by their uploading in the memory which is destined for coding question-answer reasoning. QA-programs are one of kinds of reasoning of this type. QA-programming can be used as for normative tasks so for tasks connected with scientific and human workflows.

In experimenting the investigated behavioral units are modeled as precedents. Such a form of a human activity is natural because intellectual processing of precedents lays in the base of the human experience. Experimenting the designers evolve the accessible experience by using real time interactions with its current state. This feature has found its normative specifications in the role "intellectual processor" playing of which by designers is being supported by the toolkit WIQA. In collaborative way-of-working this role can be used additionally to any other role of the technology applied in conceptual designing.

The toolkit opens the possibility for the separate execution of any operator by the designer playing the role of the intellectual processor. Before and after the execution of any operator of any QA-program the designer can check or investigate its preconditions and post-conditions. Moreover the investigated operator can be changed and evolved as syntactically so semantically, for example with the help of additional attributes.

The designer has the possibility to test any QA-program and improve it. Then this program (as the corresponding asset) can be included to the specialized library. Assets of the similar type are played the role of techniques any of which can be included to future processes of designing as in the current project so in the development of the next SIS . The complexity is being reduced because the library of programmed assets is the source of automated resources each of which can be included to the program of designer activity through calling the name of the asset. In WIQA-environment the assets are embedded to the repository (Experience Base) which is a kernel of Experience Factory. Such Experience Factory supports the real time interactions with the repository for designers from a number of groups each of which uses separated WIQA.

REFERENCES

Basili, V. R., Lindvall M. & Costa, P. (2001). Implementing the experience factory concepts as a set of experience bases. 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. &Ribeiro, P. (2012). Mapping RUP Roles to Small Software Development Teams. Proceedings of the 2012 International Conference on Software & System Process, 190-199.

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

El Emam, K. & Koru, A.G. (2008). A Replicated Survey of IT Software Project Failures. IEEE Software 25 No. 5, 84-90.

Framework for Software Product Line Practice, Version 5.0. http://www.sei.cmu.edu/ productlines/tools/framework.

Held, M. & Blochinger, W. (2009) Structured collaborative workflow design, Future Generation Computer Systems, vol.25 no.6, 638-653.

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

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

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

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

Sjoberg D.I. K., Dyba T. & Jorgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research. 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. Proceedings of the 11th International conference on Software Methodology and Techniques, 180-197

Sosnin, P.(2012) Pseudo-code Programming of Designer Activity in Development of Software Intensive Systems. Proceeding of the 25-th International conference on Industrial Engineering and other Applications of Applied Intelligent Systems (IEA/AIE 2012), Dalian, Chine, 457-466.

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

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

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