Scientifically Experimental Way-of-Working in Conceptual Designing of Software Intensive Systems

P. Sosnin*
* Ulyanovsk State Technical University, Computer Department, Ulyanovsk, Russia
psosnin@hotmail.com

Abstract—Inclusion of experimenting into way-of-working used by a team of designers facilitates increasing a degree of successfulness in designing of Software Intensive Systems (SIS). This paper presents a scientific approach to experiments the objects of which are units of designers’ behavior aimed at solving the project tasks in conceptual designing. The offered approach is based on specifying the behavior units as precedents and pseudo-code programming the plans of experiments. Reasoning used by designers in experiments is registered in question-answer forms. Experimenting is being supported by a specialized toolkit.

I. INTRODUCTION

Extremely low degree of a success in developments of SIS is caused basically by problems in a domain of software engineering. Last attempts of estimating these problems and even re-founding software engineering are bound with a SEMAT (Software Engineering Methods And Theory) initiative in documents of which a way of working used by a team of designers is marked as a very important essence [1]. There “way-of-working” as a notion is defined as “the tailored set of practices and tools used by a team to guide and support their work.”

Ways-of-working have a rich history in software engineering. They have been used in the past and are used nowadays for a system life cycle as a whole so and for separate cycle stages. The good example of collaborative way-of-working is Rational Unified Process (RUP) created by corporation IBM [2]. Thus many effective ways-of-working exist but the search of their new versions aimed at increasing the degree of success is very actual. It is necessary to note that new versions should be coordinated with essential features of software engineering.

There is a steady understanding, that development of the software has the empirical nature [3]. This feature is essential and it has a rich potential for an improvement of ways-of-working. One of promising directions of the improvement is connected with analogies between designing and a scientific research. One version of the analogy between ways-of-working and scientific experiments will be presented in this paper.

The suggested analogy has the following basic features:

- It captures only the conceptual stage of designing where mistakes are the most expensive.
- Objects of experimental investigations are units of designers’ behavior which are created and investigated by designers in the frame of their way-of-working.
- Investigated behavior units are interpreted as precedents, working with which the designers use question-answer reasoning and pseudo-code programming of the own activity.
- A very important result of any experiment is an understanding embedded to its description which is sufficient for repeatable understanding in experiment reuses.

Experiments are implemented with using a toolkit WIQA (Working In Questions and Answers) which is oriented on the real time support of the team work in collaborative conceptual designing of SISs [4].

II. PRELIMINARY BASES

A. Why Investigation of Behavior Unit

An important feature of designing the SIS is the work of designers in conditions of a high complexity. The system or its any component is estimated as “complicated” 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, various interpretations of the Kolmogorov measure [5] are applied for estimations of the degree of the system’s complexity. This measure is connected with the minimal length of program \( P \) that provides the construction of system \( S \) from its initial description \( D \). Distinctions in interpretations are usually caused by features of the system \( S \) and formal descriptions to be used for objects \( P \) and \( D \).

In accordance with their destinations, for the programs of \( P \)-type, it is more preferable to build them as programmable systems of designers’ actions while using the specific methods of metaprogramming \( M \). Explicit and/or implicit methods for managing the designers’ activity are used in the technology behind SIS design. The reality of such management demonstrates that the complexity of a \( P \)-program is no less than the complexity of SIS in any of its used states. Moreover, any \( M \)-program that provides the construction of a \( P \)-program should be built on the basis of the same initial description \( D \) of the system \( S \). It can be presented by the following chain: \( D \rightarrow M \rightarrow P \rightarrow S \).

Named relations between \( D \), \( M \), and \( P \) can be used by designers to divide the process of designing into stages
[D(t0)→M0→P0→S(t1)], [D(t1)→M1→P1→→S(t2)],….,
D(tn)→Mn→Pn→S(tn)],
where a set \{S(t0)\} reflects the states of the SIS that are
being created.

Dividing the designing process into stages is a typical
step in any modern technology that involves the creation
of SISs. In different technologies, such an approach is
used in different forms for different aims. This approach
helps to decrease the complexity of the interactions with
SIS in any of its states S(t). However, until now, the
viewpoint of programming on the designer activity has not
been supported instrumentally in the early design stages.

It is necessary to note that the creation of instrumental
means for an explicit work with M- and P-type artifacts
essentially depends on their understanding. In an approach
described in this paper, these artifacts are understood as
models of designers’ behavior that are created for
scientific experimentation on the corresponding behavior
units included in the way-of-working. Therefore, the
indicated instruments should have the potential for the
creation and use of the artifacts of M- and P-types in
experiments with behavioral units of designers’ activity.
Moreover, the means of supporting the experimental work
should help the designers create solutions for investigating
tasks and opening the possibility for their confirmatory
reuse in the designers’ team.

Thus any P-object (as a program of actions) defines the
way-of-working the use of which helps to the designer or
designers to build the corresponding artifact in definite
conditions. Let us notice that a very important kind of
such artifacts is a prototype version of future program
units. In this case the P-objects describe algorithmically
the prototypes in actions executed by designers. The
prototypes are being created not only for program units.
Such objects are used always in experimental aims.
Therefore, investigating algorithmic prototypes, the
designers investigate usually the units of the own
behavior.

B. Why Attention to Conceptual Designing

There were several reasons for the choice of conceptual
design as an activity domain for the constructive use of the
analogy between designing and scientific experimental
research. The first reason is connected with the high cost
of errors that mainly arise from incorrect understanding of
the cases that the designers are working on.

Any conceptual project is aimed at a description of the
system that is being created, to allow for its structure and
behavior to be coordinated with natural laws and
normative rules the system should satisfy. Hence,
designers should prove that the used models of “cause-
and-effect regularities” provide the required coordination.
This approach should be put into practice in the
conceptual design of a specific SIS as early as possible.
The aim of any scientific experiment is an existence of the
confirmation of a corresponding cause-and-effect
regularity. The similarity of the obligations of designers
and scientists was the second reason for this choice.

The third reason is the lack of methods that are
included, especially modern technologies for supporting
the experimental activity of designers at the conceptual
stage of design. It is necessary to note that, at the
conceptual stage of design, the investigated scheme of the
designer’s actions can be a simplified version of the task
solution, demonstrating only how regularities of the task
can be materialized and will be used.

Why Interpretation of Behavior Units as Precedents

In a general case, a system of the SIS-type involves
software that is combined with peopleware and other
different components. Any such system can be interpreted
as a naturally artificial world, the processes of which are
implemented in accordance with specific cause-and-effect
laws (regularities). Some of these regularities are laws of
nature, while others have a normative character.

In the offered approach, the work with regularities is
considered from the viewpoint of the designers’ behavior
in the solution processes of the project tasks. Moreover,
the approach is oriented toward the analogy between
projects completed by designers and scientists. In specific
circumstances, any designer plays the role of a scientist
who prepares and conducts experiments with P-type
behavior units. In such experiments, the designer works in
the naturally artificial world of SIS, which is developed in
the used technological medium. Any experiment is
connected with solving the corresponding task appointed
to the designer on the team.

When solving the appointed task, the designer must
focus on the causes-and-effects of this task, to prepare an
experiment that confirms that the indicated causes-and-
effects exist in the investigated world. In the experiments,
the designers should focus on “how to do” but not on
“What to build”, which correlates with the main principles
of SEMAT. In experimenting, the designers acquire units
of experience, which they should register in the form of
models that provide future reuses of the experiments. It is
necessary to notice that typical (reused) units of human
behavior are usually called “precedents.”

According to the 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).

Natural precedents are based on conditioned reflexes,
which are intellectually processed and are included as
models of the human experience, as its units. An
interpretation of behavior units as precedents prompts the
necessity of intellectual processing them, which is aimed
towards their planned reuse. In the offered approach, the
intellectual processing of precedents is based on the
logical scheme that is presented in Fig. 1. The applied
logical scheme allows integrating the natural and
normative regularities in the precedent model. Natural
regularities are reflected in pre- and post-conditions, while
normative regularities are specified in the motives and
aims components.

```
Name of precedent P;
while [logical formulae (F) for motives M = [M_a]]
as ( [F for aims C = [C_b]])
if [F for preconditions U = [U_c]]
then [plan of reaction (program) r]n
end so [F for post conditions U* = [U*_d]]
------------------------------
there are alternatives {P_f(r)},
```

Figure 1. Logical scheme of precedents
C. Why Question-Answering and Pseudo-Code Programming

Intellectual processing of natural precedents is fulfilled by means of a natural language that also supports the access to the units of experience. Such effects are caused by processes of consciousness of a dialog nature. Dialogical processes of consciousness provide a choice of experience units and their adjustments for corresponding situations.

Accordingly, a question is understood by the author as the natural phenomenon associated with a specific situation in which a human interacts with their own experience (models of precedents). In this case, the “question” is a symbolic (sign) model of the corresponding question. The associated understanding helps explain the necessity of fitting the “question” to the investigated situation. Implicit questions and answers exist in actuality, while “questions” and “answers” are presented as symbolic models.

Thus, question-answer reasoning reflects processes in the consciousnesses of designers when they interact with their own experience. Explicit question-answer reasoning helps in controlling interactions with experience. This type of reasoning is expedient for the creation and use of experience models that present the designer’s behavior.

In activity practice, the person usually represents their own behavior by means of a plan that is written in natural language based on its algorithmic usage. Repeatable work that is fulfilled by people is represented by techniques that are also written in natural language based on its algorithmic usage. This approach prompts the designer to use this technique in planning experiments, which was the reason behind choosing a pseudo-code language for programming the units of the designer’s ideas. Interacting with such programmed models of behavior, the designers will apply their experience in the use of natural language.

The logical scheme of the precedent specifies two parts of its model because of their separate pseudo-code programming. The first part must define the access to the precedent model, while the second part must describe the corresponding reaction that must be implemented by the designer.

It is necessary to recall the M- and P-types of programs that were chosen to describe the designers’ activity in the offered approach. Let us clarify the difference between these types of programs. P-programs are destined for simulating the units of the designer’s behavior in the solutions of the appointed tasks, while M-programs describe the collaborative work of the designers. Thus, M-programs are destined for controlling the activity of the designers, while they work in parallel and in coordination. This type of program is also suitable for controlling the pseudo-parallel work of the designer, with a number of appointed tasks. In general, the M-programs automate the implementation of the workflows.

Workflows in conceptual designing are formed according to the planned schemes; thus, they conform with the current situation. Workflows of the planned type can be programmed in advance, but workflows of the second type are to be programmed by working designers (not professional programmers) in real time. Hence, the language of pseudo-code programming, which will be used by designers for the creation of P-programs, can be used for the creation of M-programs also.

D. Place of Offered Means in the Creation of Ways-of-Working

In accordance with its definition, the way-of-working includes the tailored set of practices chosen for the SIS design. The presented approach concerns the used practices, and mechanisms of their tailoring applied to the processes of conceptual designing. The specificity of the approach will be clarified by means of roles that provide the constructive work of designers in technological environments.

Any modern technology that is used for designing the SIS includes modeling the work of designers with the help of roles. For example, the current version of Rational Unified Process (RUP) supports the activities of designers playing approximately 40 roles. RUP is a “heavy” technology; therefore, for small teams, the quantity and specifications of the used roles are decreased and simplified [2].

In any case, the role is a special version of a designer’s behavior that satisfies a certain set of rules. Role specifications depend on the corresponding tasks and tools that support their initial solution and reuse of solutions. Thus, for specifications of any role, the corresponding tasks and necessary tools should be defined by their essential features.

This approach recommends expanding the traditional set of roles by including into this set the specialized role called “intellectual processor” (I-processor). This role exists in addition to any other role that is played by any designer in the study for the investigated task.

Acting as an I-processor, the designer constructively uses the accessible experience for different useful purposes. Interactions with experience units are implemented explicitly and/or implicitly but on the basis of QA-reasoning. The main purpose of using the explicit forms is the programmable creation of the necessary precedents and experimentation with them. Implicit forms are suitable for (pseudo-code) programming of used practices oriented to execution of their programs by a designer playing the role of I-processor. Thus, the applied way-of-working can be adjusted to the use of I-processors in situations where it is planned or estimated as being useful.

Actions of I-processors are supported by the toolkit WIQA, which provides the collaborative execution of workflows “Interactions with Experience” by the group of designers in the client-server medium [6]. The basic features of an I-processor will be specified below.

We note that a set of practices embedded into the workflows “Interactions with Experience” can be combined with other practices of the technology used.

III. RELATED WORKS

The offered version of experimentation described is coordinated with basic principles of the SEMAT Kernel, which is described in [2]. The version is oriented towards ways-of-working that focus on the real-time activity of designers. “The process is what the team does. Such processes can be formed dynamically from appropriate practices in accordance with current situations. The informational cards and queue mechanisms are being used for managing of ways-of-working [7].”
It is necessary to note that this version applies units of task types for structuring the activity of the designers because solutions of tasks facilitate the enrichment of the accessible experience by scientific experimentation. By experimenting, the scientists solve specific tasks creatively.

In the offered approach, the scientific viewpoint correlates with two faces of the software engineering described in [8], where functional paradigms and scientific paradigms are discussed. In the context of this paper, the approach means are oriented towards scientific paradigms used by software engineers.

Therefore, the important group of related studies includes publications that present empirical viewpoints on software engineering. In this group, we note the following works [3] and [9], which present the domain of empirical software engineering; papers [10] and [11], which define the Goal-Question-Metrics method (GQM-method) and Experience factory, which includes the Experience Base. All of the indicated studies were taken into account in the offered version of scientifically experimental ways-of-working.

The next group of related publications concerns the use of question-answering in computerized mediums, for example, papers [12] and [13]. In this group the nearest work presents experience-based methodology “BORE” [13] where question-answering is applied also but for the other aims and this methodology does not support programming of the creative designer activity.

The idea of the designer model as an I-processor is derived from publications [14] in which the “Model Human Processor” (MH-processor) is described as an engineering model of human performance in solving different tasks in a real-time regime. It is necessary to emphasize that I-processor is similar to MH-processor and includes the cognitive processor. However, the existence and work of this “component” of I-processor are derived from another technique that is based on reasoning of the question-answer type.

IV. ENVIRONMENT OF EXPERIMENTATION

Any experimental research is being implemented in an appropriate environment of experimenting. In the described case the role of such an environment fulfills an operational space supported by the toolkit WIQA. The generalized scheme of experimentations in the indicated space is presented in Fig. 2. The scheme reflects that, experimenting, the designer D fulfills the definite work based on question-answering (QA-work) for any appointed task. The object of experimenting is the designer behavior used in solving of the appointed task.

For the used behavior the designer builds its model (model of behavioral unit) which is accessible in WIQA for other members of the working team (in the scheme for the designer D). The used question-answer reasoning (QA-reasoning) is being registered in question-answer database (QA-base). The registered QA-reasoning is being applied in the creation of the behavior model and in experimenting with it. The scheme also reflects that the investigated behavior model can be uploaded as the model of the corresponding precedent in the Experience Base of WIQA. Any model from the Experience Base is accessible for members of the team (in the scheme for the designer D).

In the operational space the central place is occupied by the question-answer memory (QA-memory) the cells of which store the registered units of QA-reasoning. Typical units of QA-reasoning are questions (Q) and answers (A) of different types. Tasks (Z) are a very important type of questions.

We understand the question as a natural phenomenon which is being revealed when a human should use or wants to use the experience. If the phenomenon of such a type is revealed then it can be described in the natural language. Textual descriptions of questions are no more than their symbolic models. The definite question and corresponding answer are bound as “cause and effect”. They supplement each other. The answer form stored in QA-memory depends on the use of the answer.

In accordance with described understanding, questions, answers and tasks are registered in cells of QA-memory as interactive objects bound by hierarchical relations (Fig. 3).

Below a pair of corresponding question and answer will be named as QA-unit. QA-units are stored in QA-memory as data the abstract type of which will be named below 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.

In Fig. 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.

Any cell has the following basic features:

- A cell is specified by a set of normative attributes that reflect, for example, the textual description of the stored interactive object, its type and unique name, the name of its creator, the time of the last modification and the other characteristics.
- Any cell has a unique address, which has a function that is fulfilled by the type name of the stored unit and its unique index, which is appointed automatically when creating the unit. Empty cells are absent.
- The designer has a chance to appoint a number of additional attributes to the cell if it would be useful for the work involving the object stored in the cell.
The kernel of QA-memory is a specialized data base (QA-base) of the relational type realized by means MS SQL Server. The use of QA-memory as a shell of QA-base is supported by a rich set of command applied directly and as actions included into plug-ins of WIQA. Interactions of designers with QA-memory are aimed at their interactions with the accessible experience [2].

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.

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

In Fig. 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, respectively, to underline that they inherit the features of QA-memory cells.

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

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

V. PSEUDO-CODE LANGUAGE L\textsuperscript{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 analysis 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.

The specialized pseudo-code language L\textsuperscript{WIQA} has been developed for the use of QA-reasoning in programming the solution plans. This language is oriented towards its use in experiential interactions of designers with accessible experience when they create programs of their own activity and investigate them. Step-by-step, L\textsuperscript{WIQA} has been evolved to a state with the following components:

- Traditional types of data, such as scalars, lists, records, sets, stacks, queues and the other data types.
- Data model of the relational type, describing the structure of the database.
- Basic operators, including traditional pseudo-code operators, for example, Appoint, Input, Output, If-Then-Else, GOTO, Call, Interrupt, Finish and the others operators.
- SQL-operators in their simplified subset, including Create Database, Create Table, Drop Table, Select, Delete From, Insert Into, and Update.
- Operators for managing the workflows oriented towards collaborative designing (Seize, Interrupt, Wait, Cancel and Queue).
- Operators for visualization developed for the creation of the 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 includes an explicit or implicit command aimed at the execution by the designer of the specific action. Explicit commands are 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\textsubscript{WIQA}. For example, textual forms of questions are a very important class of implicit commands.

In a 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. Language L\textsubscript{WIQA} as any other language does not separate from its use. The generalized scheme of designer interactions with any QA-program is presented in Fig. 4.

The scheme indicates that first of all any QA-program is a source of the answers to the planned questions. In this sense any QA-program is the model created for achieving the definite aims. But the content of this paper is aimed at the use of QA-programs for simulating the designer activity in solving the appointed tasks.

![Figure 4. Interactions of designer with QA-program](image)

The scheme indicates that first of all any QA-program is a source of the answers to the planned questions. In this sense any QA-program is the model created for achieving the definite aims. But the content of this paper is aimed at the use of QA-programs for simulating the designer activity in solving the appointed tasks.

Originally, QA-data had been suggested and developed for real-time work with such interactive objects as “tasks”, “questions” and “answers”, which were stored in the QA-database and used by designers in the corporate network. It is necessary to recall that “task” is a type of question and “decision of the task” is an answer to a question.

On the logical level, any QA-data can be interpreted as the specialized hierarchical model of data emulated by means of the relational model of the data. Two hierarchical trees of data, the units of which are connected as questions and answers, is one of the specificities of QA-data. The general version of QA-data includes the dynamic tasks tree of the units, which are united with a system of QA-models for corresponding tasks.

VI. DETAILS OF EXPERIMENTATION

A. Preparation of Experiments

The principal feature of the proposed approach is an experimental investigation by the designer into the programmed behavior, which has led to the conceptual solution of the appointed task. Any solution of such a type should demonstrate that its reuse meets the necessary requirements when any designer of the team will act in accordance with QA-program of the investigated behavior.

As described above, to achieve the goals, the designer should work in a way similar to a scientist who prepares and conducts experiments with the behavior units of the M- or P-types. In the discussed case, the designer will experiment in the environment of the toolkit WIQA. In this environment, to prove that the aim of an experiment has been achieved, the designer has the possibility of experimenting with any QA-operator of an investigated QA-program and/or with any group of such QA-operators or with the QA-program as a whole. Describing the experiment for reuse, the designer should register it in an understandable form for the other members of the team.

To begin a specific experiment, the initial text of the QA-program should be built. In the general case, such a project would include the following steps:

- Formulation of the initial statement of the task.
- Cognitive analysis of the initial statement with the use of QA-reasoning and registering it in QA-memory.
- Logical description of the “cause-effect relation” reflected in the task.
- Diagrammatic presentation of the analysis results (if it is necessary or useful).
- Creation of the initial version of the QA-program.

The indicated steps are fulfilled by the designer with the use of the accessible experience, including the personal experience and useful units from the Experience Base of WIQA.

B. Experimenting with the QA-program

Only afterward can the designer conduct the experiment, interacting with the 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& := &LastHistoryFormResult&
Q 2.5.6 &t_str& := QA_GetQAText(&current_action_qaid&)
Q 2.5.7 SWOT_DESERIALIZE(&t_str&)
………………..
Q 2.5.14 FINISH

This source code demonstrates an often-used syntax, but features of the code are opened in interactions of the designer with it. Conditions and methods of experimenting are shown in Fig. 5, where one of the operators (with address name Q2.5.2) is shown in the context of previous and subsequent operators. Any QA-program is executed by the designer step-by-step, in which each step is aimed at the corresponding QA-operator. In this study, the designer uses the plug-in “Interpreter” embedded into the toolkit.

Interpreting the current operator (for example, Q2.5.2), the designer can fulfill any actions until its activation (for example, to test existing circumstances) and after its execution (for example, to estimate the results of the investigation), using any means in the toolkit WIQA. When the designer decides to start the work with the 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 make useful decisions.

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

The current QA-program or its fragments can be executed or used step-by-step by the designer or automatically as a whole with the help of the plug-in “Compiler”. Therefore, all of the work described above with the QA-operator can be used for any of the groups and for any QA-program as a whole. For this reason, the execution of QA-operator by the designer is similarly experimentation.

Thus, the designer has a flexible possibility to perform experimental research on any task that is solved conceptually.

The specificity of the described type of designer activity is the work controlled by the QA-program and executed by the designer interacting with the accessible experience. To underline this specificity, the specialized role of “intellectual processor” was constructively defined and is effectively supported in the use of WIQA [4]. This role is added to the other types of roles that applied in the concept design [2].

C. Description of Experiments

As described above, any experiment that is conducted should be presented by the designer in an understandable and reusable form. In the offered version of experimentation, the function with such a form is fulfilled by the typical integrated model of the precedent, which is shown in Fig. 6.

The scheme, which satisfies the function of framework F(P) for models of precedents, allows integrating the very useful information that accompanies the experiment process in its actions, as indicated above.

The central position in this model is occupied by the logical scheme of the precedent. The scheme explicitly formulates the “cause-effect regularity” of the simulated behavior of the designer.

Name of precedent P_i:

while [logical formulae (F) for motives M = {M_k}] as [F for aims C = {C_l}] if [F for preconditions U' = {U'n}] then [plan of reaction (program) r_q], end so [F for post conditions U" = {U"m}],

there are alternatives P_r(p_i).
Framework $F(P)$ includes the 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 the modeled regularity;
- graphical (diagram) representation $P^g$ of the precedent;
- pseudo-code model $P$ in QA-program form;
- the executable code $P^e$.

Any component or any of their group can be interpreted as projections of $F(P)$, the use of which allow us to build the precedent model in accordance with the precedent specificity. However, in any case, the precedent model should be understandable to its users.

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

VII. CONCLUSIONS

The approach described in this paper suggests a system of methods that simplify the complexity of designers’ interactions with project tasks in the conceptual designing of an SIS. The simplification is caused by using, in the designers’ activity, useful analogies with the work of scientists when conducting experiments. Emulating scientists is accomplished by designers while investigating their own behavior in the processes of obtaining task solutions. Moreover, they simulate such behavior with the help of pseudo-code programs that describe the plans for experimentation. Thus, designers investigate the programmed plans of the experiments that they prepare, and they conduct and describe the results in understandable and checkable forms, for later reuse.

In the experiments, the investigated behavioral units are modeled as precedents. Such a form for a human activity is natural because the intellectual processing of precedents comprises the base of the human experience. In experimentation, the designers evolve accessible experience by using real-time interactions with the current state. This feature has found normative specifications in the role of “intellectual processor”, which is played by designers and is being supported by the toolkit WIQA. In the collaborative way-of-working, this aspect can be used in addition to any other aspect of the technology that is applied in the conceptual design process.

This 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 syntactically as well as semantically, for example, with the help of additional attributes.

Debugged QA-programs are the source of resources of the M- and P-types, which promotes the simplification of the complexity that is involved in their reuse. The possibility of experimenting is supported by the special library of QA-programs destined for cognitive task analysis, problem-solving and decision-making included in the named toolkit. Suggested means are used in one project organization in creating a family of SISs.

REFERENCES