Обучение пониманию программ в компьютерном классе

А. П. Гагарин

Введение. Понимание компьютерных программ считается ключевой способностью и даже искусством, необходимым для успешного выполнения программным инженером таких его профессиональных обязанностей, как анализ прототипов, принятие и обсуждение алгоритмических и структурных решений, составление детальной документации, разработка тестов, анализ результатов тестирования и рекламаций. Выполнение этих обязанностей требует от программного инженера особой компетенции – умения судить о поведении программы на основе анализа её исходных и, иногда, цифровых текстов, скриншотов, протоколов выполнения. Умение и возможность экспериментировать с программой, выполнять её на компьютере, могут дополнять, но не заменять эту компетенцию. Проявляясь на конкретных программах, она не равносильна сколь угодно глубокой осведомлённости в отношении одной программы, формируется опытом работы с множеством программ, но при этом маскируется другой компетенцией – компетенцией разработчика.

Цель статьи – уточнить сущность понимания программ как компетенции программиста, исследовать возможности целенаправленного и форсированного обучения пониманию программ, разработать методы и инструментальные средства оценки процессов обучения пониманию программ в высшей школе. Эта цель рассматривается как актуальная в связи с тем, что в современных университетских программах обучение программированию занимает значительное время на лекциях и в ходе лабораторных работ. Но этого времени постоянно не хватает в виду высокой интенсивности инноваций в области программной инженерии. В качестве ключевой проблемы при достижении поставленной цели решается проблема управления процессами развития понимания программ в компьютерном классе.

Материалы и методы. Исследование основывается на университетских учебных планах, конспектах лекций автора, протоколах экспериментальных и показательных запусков разработанных программ, Интернет-ресурсах, таких как сайт общества ACM, и другие. Использовались следующие исследовательские методы: терминологический анализ и выбор словаря, востребованного в ходе исследований и в тексте статьи, моделирование, включая теоретическое понятийное построение модели понимания программы, имитационное программное моделирование управления развитием понимания, построение UML-моделей, а также опытная реализация предложенного интерфейса управления пониманием программ.

Результаты исследования. Предложена модель понимания компьютерных программ как состояния готовности когнитивной базы знаний к порождению правильных ответов о возможном поведении программы. На этой основе выделен особый вид обучения пониманию программ – «учебный разбор», в котором динамика вопросов и ответов о программе во время разбора используется для оценки успешности обучения и, в конечном итоге, для управления темпом обучения. Предложены пользовательский интерфейс (индикаторы, «кнопки» и алгоритм пользования ими) для учебных компьютерных систем, а также стратегии, которые обеспечивают поддержку управления учебным разбором. Примерная реализация этого интерфейса встроена в компьютерную систему JL (Joint Lesson). Разработаны граф-модель и соответствующая программа имитационного моделирования, которые отражают существенную динамику управления учебным разбором и применимы для исследований и демонстрации общих эффектов учебного разбора, а также для планирования процессов обучения пониманию программ и управления полученными планами в ходе конкретных лекций.

Заключение. Результаты исследования обеспечивают формирование среды, методов и компьютерных инструментов, позволяющих благоприятно влиять на процессы понимания программ. В частности, они позволяют управлять уровнем понимания программ у конкретной аудитории, адаптировать стиль преподавания к этой аудитории, уточнять временные планы лекций по программной инженерии.

Ключевые слова: компьютерная программа, понимание программ, учебный разбор программ, учебная система, пользовательский интерфейс (UI), синхронизация учебного процесса.

A. P. GAGARIN

Monitoring and control of the program comprehension teaching in a computer class environment

Introduction. The comprehension of computer programs is claimed to be a decisive faculty and even skill of any successful exercising professional duties of program engineers, for example, prototype analysis, decision making and discussion about algorithms and structures, development of detailed documentation, test development and test analysis. Exercising of the duties demands from the engineer a special competence to judge about a program behavior basing on the consideration of its source and sometimes digital codes, screenshots, execution protocols. The skill and opportunity to make trial runs can supplement but not replace the competence. Manifesting itself on particular programs it is not a presentation of an extensive and deep familiarity with the programs, but results from the experience with a lot of programs and is obscured by another competence, the competence of the developer.

This paper is aimed to precise the interpretation of the notion “program comprehension”, to research the opportunities of a purposeful and enhanced teaching the program comprehension, to develop methods and tools for the assessment of the program comprehension teaching in the higher school. The aim is relevant because of current university curricula allot considerable lecture and laboratory time for teaching programming, including program comprehension. But this time is constantly not enough in view of a growing intensity of innovations in the area of the program engineering. The problem of a control over processes of the program comprehension fostering in a computer class is a key problem of achieving the aim having been set.

Materials and methods. The research is based on the university syllabi, the author’s lecture notes, protocols of experimental and sample runs of the developed programs, Internet resources, such as site of the ACM and others. The following research methods were used: terminological analysis and elaboration of a vocabulary demanded in the research and discourse within the paper; modelling and simulation, including a theoretical conceptual model representation of the program comprehension, a computer program simulation of the monitoring and control over the comprehension progress; an UML-presentation and a trial implementation of the proposed interface for the program comprehension monitoring and control.

Research results. Redefinition of conceptual cognitive models of program comprehension is proposed. The program comprehension is postulated to be a state of the cognitive knowledge base that represents an actual affordance of giving correct answers to questions about an eventual program behavior. This model highlights a special kind of teaching to be called “didactic parsing” that is targeted to generate at the audience a comprehension of programs being parsed. Dynamics of question and answers during the didactic parsing is used for assessing the education success and, finally, for a control of the teaching tempo. A user interface for learning computer systems (counters, controls and policies) are proposed to assist in monitoring and control of the didactic parsing. Sample implementation of the interface is built in JL (Joint Lesson) computer system. A graph model and a corresponding simulation program are developed that capture relevant dynamic features of control over the didactic parsing and are applied for the further investigation and demonstration of general didactic parsing effects as well as an aid for planning and control of program comprehension teaching during particular lectures.

Conclusion. The research results provide creation of an environment, methods and tools enabling to positively impact the processes of program comprehension attainment. In particular, they permit to control the program comprehension level at a particular audience, adjust style of teaching to be most applicable to it, refine time plans of lectures in the area of program engineering.

Keywords: computer program, program comprehension, didactic program parsing, learning system, user interface (UI), monitoring and control, synchronization of learning

For Reference:
Introduction

Within numerous initiatives originating from UNESCO Institute for Informational Technologies in Education two directions are outlined that belong to the scope of the paper. These are Teacher Professional Development with ICT-empowered Innovative Pedagogy as well as Learning for the Future: ICT Competency Development for Schools. Solving pedagogical problems on the cognitive level it is a promising approach, that permits the pedagogy to gain intended innovative future-oriented traits. This paper represents an attempt to apply this approach to one of the old didactic problems, namely, to the problem of program comprehension.

The program comprehension, that is the comprehension of computer programs is a topic being on the agenda of the IT community about last five decades. The concept “comprehension” was firstly applied to computer programs, apparently, 1977 by B. Schneiderman [1] and J. P. Boysen [2]. They defined this concept as “the recognition of overall function of the program, an understanding of intermediate level processes including program organization and comprehension of the function of each statement in a program” [1, p. 95]. Now a formal description of the concept “comprehension” as of a process was proposed by Y. Wang and D. Gafurov in [3].

But the phrase “program comprehension” if it is encountered in a periodical on computing will be usually explained as a process of reading or understanding some program or programs, e.g. in a special site “Code Comprehension: What is it?” [4] or by A. A. Shargabi [5, p. 263]. Such an explanation deriving the verb “comprehension” from the verb “understanding” cannot, of course, be taken for the strict definition, as far as the first meanings of both verbs “understand” and “comprehend” are the same. Both verbs are synonyms according to the Oxford as well as the Marriam Webster dictionaries. The phrase “program comprehension” is broadly used in computing as an umbrella logo, for example, in titles of ACM conferences: IEEE/ACM International Conference on Program Comprehension (ICPC), International Computing Education Research (ICER). In the issue the concept is further used in a restricted meaning that the author has tried to define more clearly.

Within particular researches the concept “program comprehension” is formalized by means of various conceptual models. Major phenomena of the program comprehension had been early recognized and the first conceptual models had been proposed based on the toolbox of the contemporary cognitive psychology. A corresponding review is presented by M. P. O’Brien in [6]. These models represent in their most generic form a trivial and quite natural structure, that includes: a programmer. a computer program to be comprehended in front of the programmer, and the so-called assimilation process. It is performed by the programmer generating a mental model of the program in a knowledge base contained inside the programmer’s cognition. The program could be accompanied by auxiliary explanatory materials jointly known as an “external representation”.

From the very beginning, studies of the program comprehension have been aimed at facilitating and qualitative improvement of any program development by fostering the comprehension processes as a critical constituent of software construction, validating and maintenance processes. The favorable strategies and context conditions for the comprehension flow had to be found and investigated, as well as the principles and tools for the comprehension assessment. An empirical approach to the investigations had been
chosen as the dominant one. That is, a task has to be executed as a trial work, that includes comprehension processes. The comprehension flow is assessed by duration of the task execution and the quality of the information outcome if any. The assessment is usually expressed in terms of the chosen mental model.

It is remarkable, that no measure of comprehension was apparently proposed. The majority of experiments directed to imply some enhancement of the comprehension under observation assessed it implicitly, taking into account the results of later actions, such as program construction, design, testing. The actions surely involve comprehension but cannot represent it. In fact, these were the experiments of enhancing the software engineering processes in their fullness. In this capacity they can be matched with works of J. Sweller et al. [7], where the Human cognitive architecture is based on the concept of the cognitive load.

The concept of the cognitive load was developed on the assumption that human memory manifests itself in two forms: as a long-term memory and as a working memory. Cognitive load is defined as a measure of mental efforts needed to transfer information from the long-term memory into the working memory. Alleviation of cognitive load fosters solution of tasks, that can have different complexity and content, going beyond programming and program comprehension, but dedicated for learning.

Numerous investigations have been carried out on the ground of the mentioned concepts, but a retrospective analysis made by J. Siegmund [8] gave evidence of important drawbacks to the practical applicability of the attained results. The findings and technological improvements did not justify the expectation of the software development, Novel system software and languages have brought new opportunity and new challenges, old problems of comprehension measurement grew scarper. Programming languages, configuration and modes of application programs provide a variety of parameters, that are hardly to take into account by a researcher setting up the experiments. Building a specialized artificial environment for a particular experiment may circumvent the challenges, but it leads to trade-off between control and generalizability of the experiments outcomes, making them poorly comparable.

A survey conducted 2014 by W. Maalej et al [9, pp. 33-34] over 1 447 developers from software companies about the state of the practice in program comprehension showed that “tools for comprehending programs and accessing and sharing knowledge about it are either unknown or rarely utilized”, “wherever possible, developers seem to prefer strategies that avoid comprehension”, “there is no one-size-fits-all approach to comprehending programs and exchanging knowledge about it. Most observed developers choose from a set of structured comprehension strategies depending on their current work context because of time and mental effort needed”.

Last years, focus of the researches in program comprehension has moved from the support of software developers to the development of program comprehension at novices, the persons who begin to learn programming, appearing mainly to be school children, students of high schools and universities. That is the simultaneous double contribution of C. Izu et al highlighting in [10] the Learning Trajectories passed by novice programmers and in [11] – the Learning Activities along the Trajectories. The other research branch is represented by the issue of A. van Leeuwen and J. Janssen [12] about teacher guidance during collaborative learning. The ideas of the previous period were ramified and enhanced.

In the exemplary contribution of A. Shargabi et al [13] special tasks are performed by the participants of an experiment that call implicitly some program comprehension processes at various levels of abstraction. It is postulated, the tasks impact the comprehension faculty of participants that is later assessed by tests.
The hallmarks of the experiment are: tasks, pre-tests, post-tests, program comprehension mental models (PC MM). Each task belongs to one of six types, that should be executed by the participants over a program module (program text) in order to accomplish the task. The PC MM expresses the ability of the experiment participants to operate on some of three levels of abstraction. This ability is assessed by pre-test before and post-tests after every task accomplished in the experiment.

The numeric score is calculated depending on the answers’ correctness. There were calculated four scores: for each of the three abstraction levels and the total score of the PC MM calculated as their mean. Semester four university students in Computer Science and Information Technology were recruited as participants of the experiment. They were divided into six groups, each containing about 30 members.

Comparison of average scores of groups and the means of comparisons of individual scores within groups and between comparable groups entailed conclusions that “the different tasks improve different abstraction levels and different information categories depending on the nature of the task” and “higher cognitive categories tasks can improve higher abstraction levels in one or more information categories”.

In spite of undeniable regularities discovered by the experiment permitting to use the outcomes by design of curricula according to evidence of C. Izu et al [11], the scores measured for particular groups show large scale variations reflecting influence of unveiled factors. For example, task “Remember” provides results shown in Table 1.

<table>
<thead>
<tr>
<th>Levels of abstraction</th>
<th>Pre-test</th>
<th>Post-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement Level (SL)</td>
<td>37.65</td>
<td>74.23</td>
</tr>
<tr>
<td>Block Level (BL)</td>
<td>46.52</td>
<td>57.84</td>
</tr>
<tr>
<td>Module Level (ML)</td>
<td>32.03</td>
<td>37.48</td>
</tr>
<tr>
<td>Total score as mean</td>
<td>38.73</td>
<td>56.52</td>
</tr>
</tbody>
</table>

The cognitive load study appeared to be closely connected with collaborative learning that contributes to enhance collaboration between learners by designing instructional interventions and supporting social and cognitive processes, respectively, during the collaborative learning, look L. Kester and F. Paas [14]. Collaborative learning theory was integrated with the theory of cognitive load, look P.A. Kirschner et al. [15].

Along with the comprehension of programs similar researches are performed regarding algorithms. It should be remarked that the researches avoid using token “comprehension” in favor of “understand”, “analyze” as in [16] of A. Kerren et al and in [17] of N.I. Pak et al. Attempts are observed merging the program comprehension with the comprehension of algorithms by P. Kather and J. Vahrenhold, [18] and “computational thinking” of A. Lamprou and A. Repenning [19].

Majority of experimental studies reviewed relies on IDE’s as tools, but specialized products for the experiments were also developed: for Automated Assessment of Teaching Programming by C.A. Higgins et al. in [20], Pedagogical Framework of N. Looker [21], software comprehension framework of Z. Porkoláb et al. [22], editor of M. Mészáros et al. [23].
Last years has shown up an interest in comprehension of large software systems as O. Levy and D. G. Feitelson in [24].

The analysis of program comprehension has revealed that old primary challenges of the program comprehension persist despite the considerable progress in strategies of experimental researches and development of corresponding tools. Considering more than 10 years long author’s experience of teaching software engineering in high school, including the experience of distant teaching during COVID-19 quarantine, the author decided instead of looking for general ways of better program comprehension teaching to elaborate a technology that could enable any teacher to invent and justify own patterns of teaching program comprehension adapting them to the audience, class of the software to be comprehended and any other valuable parameters. In this regard the objective of the study presented in this paper has become to propose principles, algorithms and tools for monitoring the processes of acquiring skills of program comprehension by the learners, especially in a classroom environment. The initial author’s ideas and approaches to their implementation were presented in [25]. Now, in the current issue they are enhanced by proposed algorithmic solutions and justified by simulation experiments.

Materials, methods and instruments

The research is based on the university syllabi, the author’s lecture notes, protocols of experimental and sample runs of the developed programs, scientific publications, including pedagogical and engineering periodicals, conference materials, Internet resources, such as site of the ACM and others.

The following research methods were used: terminological analysis and elaboration of the vocabulary demanded in the research and discussion in the paper; system analysis of the experience gained by the author, as well as by the scientific community in the area of software engineering and computing education, including a comparative analysis of the recent scientific and engineering results relating to the learning and teaching programming; modelling and simulation, including a theoretical conceptual model representation of the program comprehension, a computer program simulation of the monitoring and control over the comprehension progress, an UML-presentation and a trial implementation of the proposed interface for the program comprehension monitoring and control.

The trial implementation has involved following program products: UML editor StarUML®; IDE Microsoft Visual Studio® VS 2022; ASP .Net Core 3.8; RDMS MS SQL Server; DBMS MongoDB.

Results of the study

1. Conceptual cognitive model of the program comprehension

The model is shown in Figure 1 and, following a generic model of learning programming, includes a Learner, a Teacher and Subjects of comprehension as major objects. The optional Teacher may be only one. The audience may involve multiple Learners. The Subjects of comprehension may differ in content. All components of the Subjects of comprehension are optional except of a Program code.
The Learner wields a Mind shown in Figure 2, that contains a Cognitive Knowledge base and a Cognitive processor operating on it. The Cognitive Knowledge base has a Working memory and a Long-term memory, the latter is split into Declarative and Procedural long-term memories. Properties and relations of these objects correspond at large to the tenets of famous cognitive architectures reviewed by A. Fekete, and Z. Porkoláb in [26].
The comprehension is ensured by the Cognitive processor executing the assimilation process. Program being comprehended and other auxiliary components of the Subject of comprehension are mapped into program models that are provisionally located in the Working memory and stored in the Long-term memory. Apart from program models, the assimilation process produces strategies of assimilation, that fall in two categories: generic strategies and particular strategies associated with particular programs. The generic strategies represent a derivative of an experience accumulated during series of assimilations. The strategies represent a procedural knowledge stored in the correspondent part of the mind’s knowledge base.

The strategies are applied by the Cognitive processor to the programs currently being submitted for comprehension. In this regard, it is envisioned that the assimilation process of any program may have a repetitive character recurring continuously during a time period, or may proceed time by time. Independently of the mode of recurring each attempt enriches the program model. Program comprehension is not equal to the ability to restore the text (code) of the programs. It is sometimes possible but appears as seldom as never because it is quite in vain and, moreover, it is not needed at all. The outcome of a successful assimilation is about to accumulate a toolbox of patterns and techniques, empowering the programmer to answer the majority of questions about the program without any significant delay. Other words, the program models inside the human mind represent a resource of evaluation of predicates over programs that have been comprehended.

Conceptually, processing of the questions (predicates) consumes data (if any) from the Declarative Long-term memory and procedures from the Procedural Long-term memory stored by the processing of the program under consideration during its current or previous session of assimilation and related to the peculiarities of the program. though some data and procedures may be alien to the current program and may originate from the overall programmer’s experience. At the same time the cognitive processor stores data and procedures somehow emphasized during current session of assimilation in both long-term memories.

Indisputably, content and flow parameters of the questions depend on the goal of dealing with the program. But when the assimilation strategy primarily used involves rather long and complicated flow of subordinate questions it can get relatively independent from the initial goal. Assimilation processes being sufficiently independent are usually incited by questions of high degree generality, as example: whether the program is usable, or secure, or meets the requirements of the object-oriented programming.

Assimilation process as reasoning can be evolved loudly by voice, in a form of an internal speech, or, one can guess, subconsciously. It should flow smoothly else it hazards not to be completed successfully. Any contradiction in inferences or embarrassment by applying strategies will hold the assimilation till the obstacle will be overcome, otherwise the comprehension will not be attained.

As result, it is proposed a more precise meaning of the concept comprehension: it has to be a state of the cognitive knowledge base that represents an actual affordance of giving correct answers to questions about program under consideration (other words, being comprehended).

2. Didactic parsing
A special kind of teaching is taken into consideration, called “didactic parsing”, targeted to generate at the audience a comprehension of programs being parsed.
A group of students and a teacher (instructor) conduct a lecture or training session of learning programming according to the following tenets:

- The workplaces are equipped as shown in Figure 1.
- Subjects of comprehension are identical at all workplaces including the teacher’s one.
- The teacher delivers a speech demonstrating an assimilation process of the program laid before and targeting to gain comprehension of the program at the students.
- The teacher’s speech is composed of extracts from the program code and other subjects of comprehension, own comments, and directives to move across the presented subject of comprehension.
- Listeners (students) follow the speech trying to approve the teacher’s assertions about the program, especially about its behavior.
- A student may interrupt the teacher’s speech if she/he is not able to agree with some assertions or execute a directive. In this case the student puts questions or requests a delay to get her/him a time for contemplation.
- The teacher may put questions to students and wait for reply.
- The teacher replies the questions and requests of delay by additional explanations or pauses the discourse.

The algorithm of cooperation between the teacher and students is shown in Figure 3 as UML-model.

The comprehension subject is processed in its parts, one by one from the beginning to the end. It is a determining feature of the didactic parsing and its algorithm backbone. The comprehension subject is decomposed in parts rather arbitrarily. The size of a part may
vary from a couple dozen lines to a dozen pages. It is anyway desirable, that each part be semantically cohesive. These parts will be further called pages or “chunks”. Consecutive sequence of chunks processed as a whole composes a session.

3. User interface for didactic parsing automation

Levels of monitoring and control. A user interface for learning systems is proposed that enables monitoring and control of the didactic parsing. It includes hardware and software features as well as a methodology of monitoring and control the user should follow. The monitoring and control target the attainment of desired program comprehension by students under respect of their syllabus and are exercised at three levels:

- synchronization of participant’s attention;
- control over didactic parsing tempos;
- control of attained comprehension quality.

The interface may be implemented in any distributed computer system containing at least two terminals equipped with graphical display, a keyboard and conventional manipulators, as computer mouse or touchpad.

On each terminal, the interface includes a couple of windows (Main and Auxiliary), and controls, specified in Table 2.

Table 2

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teacher’s terminal</td>
<td></td>
</tr>
<tr>
<td>Left control</td>
<td>There are two Left controls: one for the Main and one for the Auxiliary windows at any terminal. Activation of the control replaces the content of corresponding window by the previous chunk of the lecture notes at the terminal. If the current chunk is the last one, the window is cleared.</td>
</tr>
<tr>
<td>Right control</td>
<td>There are two Right controls: one for the Main and one for the Auxiliary windows at any terminal. If the control is activated at the Main window of teacher’s terminal, then the content of all main windows in the system will be replaced by the next chunk or the windows are cleared and the previous content of the Main windows is moved to the Auxiliary windows. If the control belongs to a window other, than the Main window of the teacher’s terminal, the content replacement proceeds only on this window without other effects.</td>
</tr>
<tr>
<td>Dialog control</td>
<td>It opens and closes a span of time spent in the dialog state.</td>
</tr>
<tr>
<td>Tempo counter</td>
<td>It shows the value $V_{pars} = S_{parsed} / T_{pars}$, where $S_{parsed}$ is size of the lecture notes part being already parsed. $T_{pars}$ is the elapsed session time.</td>
</tr>
<tr>
<td>Delay counter</td>
<td>It shows how many students has activated the Delay control. The counter is cleared as soon as the teacher activates the Right control at the Main window of teacher’s terminal.</td>
</tr>
<tr>
<td>Time-in-delay counter</td>
<td>It is the total time $T_{delay}$ spent in the delay state elapsed from the beginning of the session including dialog emerged in delay state.</td>
</tr>
<tr>
<td>Time-in-dialog counter</td>
<td>It is the total time $T_{dialog}$ spent in the dialog state elapsed from the beginning of the session.</td>
</tr>
<tr>
<td>Time-in-parsing counter</td>
<td>It is the total time $T_{pars}$ spent in the parsing state elapsed from the beginning of the session.</td>
</tr>
<tr>
<td>Rel/Abs control</td>
<td>It toggles between showing the times in the states delay, dialog and parsing in absolute clock units or weighted by size of the program in parsing. The size may be expressed in bytes of the lecture notes presented as a module, or in lines of the source code.</td>
</tr>
</tbody>
</table>

Student’s terminal

| Delay control      | The Delay counter at the teacher’s terminal is incremented. Student requests delay. The repetitive activation of the control by the same student does not affects the Delay counter until it is cleared. |
4. Implementation
The specified interface has been partly implemented in the learning system JL (Joint Lesson) by I.L. Pyzh’janov [27]. It is a distributed client/server system. The client part is implemented in the WPF framework of Visual Studio 2022, and the server part is implemented in ASP.Net Core 3.8 technology. The system includes RDMS MS SQL Server and DMS MongoDB.

If an academic discipline is installed on the Joint Lesson, group as well as individual training may be held for the authorized student by the authorized teachers. The Joint Lesson stores the online manuals and lecture notes belonging to the teachers and the students. Automated testing is supported. Screens of the teacher’s and student’s terminals (overlayed) at the beginning of parsing are shown in Figure 4.

Left and Right controls are implemented as Buttons. Delay control on the student’s terminal is implemented as an item of a context menu.

5. Synchronization
Synchronization is specified as a feature of the proposed interface in boundaries of sessions. Usually, a session is associated with some lecture notes that include the Subject of comprehension in such an order, that all, if any, references are directed backwards from the end of the lecture notes to their beginning. A window matches approximately the size of a chunk and may contain no more than a chunk.

Roughly speaking, the synchronization is expressed in an instant simultaneous transition in all Main windows to the next lecture notes chunk that has not yet been processed within current session. Using the Right control at her/his terminal the teacher ensures parsing of one and the same chunk of the lecture notes by all students. The teacher dares this decisive action taking into account states of the Delay counter and of other indicators at her/his terminal.

Before finishing a chunk processing the teacher cautions student about coming chunk change. Any students may inform the teacher that they prefer to delay the change, activating the Delay control at her/his terminal. Noticing the Delay counter the teacher may pause and transit to the next chunk with or without discussing the cause of the delay.
Sessions are initiated by the teacher. Upon the start of a session the Main windows of all terminals show the lecture notes chunk that has been processed as the last one during the previous session if any held with the same lecture notes. The Auxiliary windows show the previous chunk or are empty.

The Left and Right controls, except of the Right control at the teacher’s Main window, let all users to pass through the lecture notes forwards, from the beginning to the end and backwards, independently of the teacher’s actions. The consecutive chunks of the lecture notes get into the windows one by one.

6. Control over didactic parsing tempo
Control within the synchronized environment is exercised in the scope of a chunk and in the scope of a session. Either scope is characterized by own policies of control ensuring achievement of the control targets. The features of the scopes and their policies are specified in terms of the following conceptual model.

6.1 Graph model of the didactic parsing
A graph model is introduced to investigate more closely the interaction between the teacher and the students. Flow of the session is represented by a graph that is depicted by a polyline on the chart in rectangular coordinates. Chunks of the comprehension subject being processed are put on the horizontal X-axis in the sequence of their processing. The Y-axis is intended to gauge the time. A node of the graph depicted by a point with coordinates (X,Y) represents a processing of a comprehension subject in the moment Y at the position X within a session. Arrows of the graph connect couples of points, representing consecutive moments of processing performed by a particular session participant. The polylines begin at the coordinate origin.

Sample processing of the $i_{th}$ chunk by a teacher and a student is shown in Figure 5. At any moment the action of every participant is characterized by some instant tempo. Both start processing at the same moment $T_i&S_i$. Initial tempos of the participants may be different, because they play related but different roles: any student acquires comprehension of the parsed lecture notes content and the teacher expresses her/his comprehension of the same material.

![Figure 5 Graph-model of a chunk processing (didactic parsing)](image)
If they could conserve their initial tempos and act independently, they would finish the chunk processing at points $T_{i+1}$ and $S_{i+1}$. But due to mutual cooperation they finish the processing simultaneously though eventually with such negative results as some deficient of comprehension at the student or some useless waist of efforts at the teacher. This point (graph node) of the real common finish is shown as $T_{i+1} \& S_{i+1}$. It is connected with the start point (graph node) by a red solid line segment. The points $T_{i+1}$ and $S_{i+1}$ connected with the start point by black solid dotted line segments are interpreted as virtual outcomes of the process. The comprehension subject is presented in Figure 5 as some lecture notes.

The teacher and the student start chunk processing simultaneously that is from one and the same point. At the first chunk of a session the point is the coordinate origin. At other chunks the starting point is the common point where the teacher and the student finish their real processing with a real tempo depending on their cooperation.

There is neither in definition no in algorithm of the didactic parsing any presupposition to intentional synchronizing students’ actions. If an audience consists of N students than its model in the proposed formalization will be a family of N independent polylines, which represent a cooperation of the teacher with each particular student. The nodes may eventually overlay but not coincide. Each polyline includes $3\times m$ triples of type $\{S_{i}, T_{i} \& S_{i}, T_{i}\}$, where $m$ is the number of chunks in the session and $0 \leq i < m$.

With due regard for the relationship between the considered initial tempos some typical use cases may be recognized and corresponding teacher’s policies specified.

### 6.2 Chunk policies

The use cases are represented in Figure 6 by polylines according to the graphical conventions set above. The synchronization permits to represent a session with multiple students by a single polyline.

![Figure 6 Main use cases of didactic parsing (of 2 chunks of Lecture Notes)](image)
The upper polyline (polyline I) depicts the case when the teacher’s initial tempo is lower than the corresponding student’s tempo. If the students would follow their inherit tempo, they would finish the processing of the first chunk at the point ‘s’. But the teacher has not recognized the situation and finished the processing later, at the point ‘t’. The next chunk is processed by the teacher together with the students from the point “t”. The flow is shown by a thick red solid line. The time span between ‘s’ and ‘t’ appears to be idle for the students.

The polylines II and III depict the case when the teacher’s tempo is higher than the corresponding student’s tempo. The teacher would finish processing at point ‘t’ earlier than the students at point ‘s’. But the students cannot attain comprehension so quickly and will slow the teacher down, so they finish processing simultaneously at point ‘s’. The teacher may choose one of two policies. Whether to accord with student’s pressing or to withstand it. At the first choice, processing of the next chunk begins at point ‘s’. It corresponds to the polyline II and the flow is depicted by overlaid solid red and dotted black line segments. This strategy will be figuratively called “Accord-policy”. At the second choice, processing of the next chunk begins at point ‘t’ and it provides a gain of the total parsing time. The strategy may be called “Breach-policy”. It is supposed that the possible loss of comprehension at students will be compensated through a supplementary training.

By the specification of use cases and policies proposed above it is assumed that the tempo of parsing remains constant throughout a chunk processing. The condition is ensured by an appropriate decomposition of the lecture notes in use. As was shown the inequality of initial parsing tempos is challenging. Evidently, the education process should provide a notable increase of the parsing tempos in general, but under ensuring that the tempos values do not deviate far from one another at every moment of the educational process.

6.3 Session policies

Following policies are proposed to satisfy the requirement of monitoring and control over the didactic parsing process throughout a sequence of chunks of a lecture notes.

Within the scope of a session additional policies are proposed figuratively named “Dry-policy”, “Urge-policy” and “Relief-policy”. They are based on cautious change of teacher’s tempo from one to the next chunk during session. It should be taken into account that the initial student’s tempo may systematically rise as a positive outcome of the education, though it can sometimes slow dawn because of eventual demotivation. A sample of impacts on initial tempos is shown on the panel in Figure 6. The initial student’s tempo grows if it is under teacher’s tempo and falls if it exceeds.

In case of the Dry-policy, the monitoring consists in tracking the delay state. Low values of Tdelay and Tdialog resulted from a session of the didactic parsing indicate that the process was too simple for the audience or the audience was not quite motivated to diligently accept the delivered parsing. If the latter opportunity has been refuted by tests outside of the session bounds, the next session of the didactic parsing can be performed in a someway compressed style. Tpars will be decreased without increasing of Tdelay and Tdialog. That is the duration of the session can get reduced. The compression may be repeated until the session duration ceases to get reduced. This strategy is implemented by transit from the first chunk to the second one at the polyline I in Figure 6.

The Urge-policy can be considered as a propagation of the Dry-policy. If a session results in ultimate low values of Tdelay and Tdialog, and any further compression of the parsing leads to an increasing of the values, the teacher continues to execute didactic parsing in the same or more compressed style in spite of increasing the Tdelay and Tdialog. This is performed
as an attempt of a pressing on the audience coercing it to disclose hidden resources of the assimilation. It is choice of the teacher how long such attempts may be retried. In any case it is advisable to preserve the session duration attained due to applying of the Dry-policy.

The Relief-policy is applied if values Tdelay and Tdialog appear to be high enough to break the preliminary lecture schedule. It indicates that the program being parsed and style of parsing are too hard for the audience. To remediate the situation the teacher may simplify and slow down the tempo of parsing process in particular on account of providing additional explanations. The teacher may do it gradually trying to perceive the level of decreasing complexity, that satisfies the audience.

Otherwise, the teacher may apply the Breach-policy. It pursues preservation of the lecture schedule by delaying achievement of the comprehension at the students beyond the lecture boundaries till the additional consulting or training. It may be especially approved if the deficiency of comprehension originates from a minority of the audience. The teacher may control the impact of her/his efforts aimed at compressing or boosting the parsing process by the Tempo counter and other indicators shown in Table 1.

7. Program simulation

Control effects over didactic parsing of chunk sequences may be investigated through simulation by a specially developed computer program. The investigation is designed as a series of experiments, each performed by a run of the program. A run simulates the process of didactic parsing of a material (of lecture notes) to be comprehended, that is divided into a sequence of chunks. The program accepts values of initial tempos of the participants and rules by which the tempos are changed by transition to following chunks. The program displays charts of the simulated parsing processes as polylines of the graph model specified above as well as total duration of the parsing and last value of the tempos of parsing. Simulation of three sessions consisting of two chunks is shown in Figure 6. Simulation of six session consisting of 12 chunks is shown in Figure 7.

8. Control of comprehension quality

The proposed formalization and simulation program are applicable, first of all, for investigation of the dynamics conveyed by the parsing process at given initial tempos and applied policies. It was noticed that the quality of the resulted comprehension may be to some extent neglected for sake of reducing the total duration of the process or making an desirable impact on the students.

Nevertheless, observing the dynamics of parameters Tdialog , Tdelay, Delay Counter provides hints of degree of self-satisfaction experienced by students of their comprehension state. It may be taken as indirect estimation of the comprehension quality. But this topic is not developed in the paper.

The discussion of the results

1. Conceptual modelling

The paper contains no assumptions pertaining to the natural phenomena, structures and processes related to the scope of psychology and neurophysiology. The conceptual constructions are made exclusively at the cognitive architecture level to provide eventual computer simulation and a dictionary for the current discussion.
Generally, the model follows the classic models as presented in O’Brien [6] and Sweller [7], except of adding the cognitive processor. In the prominent Cognitive Architectures ACT-R, Soar, and Sigma the cycle of cognitive actions is driven by the procedural memory (look J.E. Laird, C. Lebiere & P.S. Rosenbloom [28], and A.-E. Huuhtanen [29 p. 6]), but the decision has been met to separate the agent that executes the cognitive procedures for convenience of referring to it. The short-term memory is omitted because of its activity merging with the working memory activity throughout the interface proposed and considered in the paper.

Some remarks on the concept comprehension have been made in the introduction of this paper. Perhaps, the originators, “fathers” of program comprehension study perceived some oddity to hold the comprehension for a process and introduced a special process, that is the “assimilation” as its strict executor within the cognition. Thereby a door is left open to acknowledge the comprehension to be a cognitive state. The next step is made in the paper: to qualify the state as a sense. We propose to relate the sense with the sense of affordance. Accepting the condition, that the genuine meaning of the term “affordance” pertains to an affordance in space, we exploit the praxis of Isis Chong and Robert W. Proctor [30] to broaden the meaning in a sound accordance with the following everyday spoken praxis. If a process of an analysis and explanation is conveyed addressing to some person, the probable comment upon the conveying will be, highly likely, “I comprehend now”, or “I understand now”, declaring the reached state and staving off any further explanations.

The expressed point of view does not forbid to use informally the term “comprehension” with the meaning “process”.

The comprehension, as a cognitive state, manifests itself subjectively as feeling of readiness to give explanations about the program and satisfaction of the previous process of its assimilation. The trustworthy of the feeling is not absolute, and depends on the personal experience.

The proposed concept of program comprehension as a hole gives way to direct control over the cognitive faculty. Policies proposed for the control are intended to estimate and ensure the objectivity of the faculty manifestations.

2. Didactic parsing

A voice communication is peculiar to any implementation of the didactic parsing whether in a real classroom or in a virtual conference environment. The opportunity to set up the didactic parsing by a communication per written messages are not researched but is doubtful because of additional cognitive load involving.

Didactic parsing is performed as teaching in conditions of cooperative learning in meaning of L. Kester, & F. Paas [14] and P. A. Kirschner et al [15] with leading intervention of the teacher in the form of a specially organized speech as at R.K. Atkinson et al [31]. The major mission of the speech is to evolve an example of the pertinent assimilation process as in comparison of A.-E. Huuhtanen [29]. The speech should demonstrate the course of the teacher’s thought and relocation of the attention through the program text. Externally, the speech resembles “think-aloud protocols” of W. Wundt evidenced by J. Siegmund [8].

The proposition does not prescribe the content of the speech. The teacher is free to use own patterns of internalizing intrinsic details of the studied program, as well as follow old advice or recent findings of N. Javed, et al [32].

The didactic parsing may be exercised throughout a lecture (90-100 minutes) or episodically in a laboratory session. The lecture format is needed to achieve the control
Laboratory sessions may be especially useful for testing comprehension skills of learners upon didactic parsing lectures.

Obligatory condition of the didactic parsing is an active participation of learners. They should not perceive the speech only, but verify the reasoning of the teacher, reading the actual lines of the program text. To ensure the participation the teacher can put short questions admitting short answers: consent, negation or pointing of an object. There may be two kinds of dialog: an intermittent, initiated by the teacher, and extended dialog, initiated by listeners or growing up from the teacher’s questions. The extended dialog interrupts the didactic parsing and can impact its following course as shown by S. Sentance, et al. [33].

3. Synchronization

Necessity of some synchronization activity among participant of a didactic processes in distributed learning systems is well known and variably implemented, for example by “hand raising” in the famous Learning Management system Moodle [34]. Comparison of synchronous and asynchronous modes of distant education in high school in Canada conducted by E. Murphy et al [35]. They indicate that the possibility to communicate by voice is outlined as a valuable advantage of the synchronous mode.

A need to synchronize attention at all participants is a hallmark of the didactic parsing. The teacher plays whereby a leading role. The synchronization may be implemented at the informal level of verbal agreement between the participants. But a partial automation of the synchronization procedures opens additional opportunities for enhancing the education processes. These opportunities are announced in the following item of the paper, as a result of the presented study.

The approach to synchronization applied in the didactic parsing interface was proposed by author’s et al in (25] and evolved in [36]. The synchronization plays thereby a double role. From one side it enables the teacher to affect the audience coordinating student’s attention and equalizing their comprehension tempo.

From the other side it provides an informational channel from students to the teacher enabling her/him to choose the strategy of control over the didactic parsing process. Moreover, apart of the usual way to hamper the lecture by a question, the student may press the Delay button of the interface saving the efforts of composing the question and minimizing thereby the cognitive load of participating in the didactic parsing.

The accepted synchronization architecture may be estimated as rather simplistic and challenged from different points of view.

Firstly, the names of students being originators of the delay requests may be reported to the teacher. This extension seems to enrich the information flow from the audience to the teacher. But it can make some students to hide their need of delay as sign of their weakness.

Secondly, the capability to cancel delay requests may be granted to originators of the requests outstripping an activation of the Right control by the teacher. The capability can be really useful in some situations reducing the total duration of the session. But the gain will be hardly essential, because of recommendation to teachers not to tolerate long delays without opening a discussion about delay causes. Delays are practically to be considered as a sign of a too high tempo of the parsing. The teacher should choose in the situation one of the policies: whether to prevent delays by a deceleration of the parsing or to ignore the delays at all.

Thirdly, the teacher’s right to interrupt a delay and pass to parsing of the next chunk of the session may be regarded as an inadmissible element of coercion. But it should
be taken into account that a pressure on the students is mitigated by positive effects of stimulation of their educational progress and prevents an intentional or unintentional blocking of lectures.

Each node of the polyline with the exception of the first one represents the moment in time when students or the teacher have finished or could finish processing of a chunk.

4. Monitoring and control

Problem of self-regulation in educational process is considered by D. Loksa et al in [37]. A learning system with elements of monitoring the educational process is shown by J.L. Waite, et al. in [38] and S. Sentance et al in [39].

The student’s initial tempo depends on her/his skills and knowledges under condition of the appropriate (mostly suitable) teacher’s assistance. Though the value of the initial tempo can be guessed by experience, it is not known neither to the students nor to the teacher at the beginning of the processing.

The teacher starts processing at her/his initial tempo and can notice its deviation from the student’s initial tempo later by the reaction of the audience, that is by questions and delay requests, or can leave it unnoticed at all.

The proposed control of the didactic parsing process is rooted in an inherent property of the process to be dynamically compressed or inflated due to reduction of inferences, omitting references and examples, from one side, and itemizing deductions and multiplying explanations, from the other side. An experienced teacher is able to apply such transformations purposely having some trustworthy feedback. The feedback can be provided by two channels: the dialog activity of the students and indicators contained in the interface.

Typical combination of conditions influencing the didactic parsing flow and its polyline graphical presentation as result of the simulation are shown in Figure 7. Segments of the polylines represent a linear approximation of the parsing during processing of particular chunks. Gradient of line segments reflects tempo of the parsing. The tempos are specified in Table 2. Each of the six experiments is characterized by the initial tempo and six rates. A tempo is multiplied by one of the rates at the beginning of the next chunk parsing. The choice of the ratio is shown in Table 2.

All the experiments are intended to investigate the qualitative effects of the control over didactic parsing. Therefore, units of time, tempos of processes and size of parsed materials are conditional. The effects are scalable with regard to materials size and processing duration. 12 steps are provided. Five minutes parsing of a page as well as a lecture may be taken for one step. Then the experiment as a whole will simulate a short lesson or short course, correspondently.

The experiment I represents the simplest case, when initial tempos of students and their teacher are equal except of growth of the student’s initial tempo if the teacher’s tempo at the beginning of a step appears to be 10% higher. The resulting chart is a simple straight line.

The experiment II differs from the previous one so, that the students’ tempo is hesitated near the teacher’s tempo, being excited by a 10% lag of the students’ tempo at the beginning of the experiment and compensation of eventual tempos difference at steps within experiment.

The experiment III demonstrates the Urge-policy. The teacher presses the students by 10 % exceeding all kinds of initial tempos, and the students reply by the 10% growth of their initial tempo. The policy application results in considerable acceleration of the course.
Parameters of didactic parsing processes shown in Figure 5

<table>
<thead>
<tr>
<th>Line Mark</th>
<th>Teacher’s comprehension tempo</th>
<th>Student’s comprehension tempo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>initial</td>
<td>Tempo Change depending on Teach. &lt;=&gt; Stud. tempo relation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T &gt; S</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>1</td>
<td>0,85</td>
</tr>
<tr>
<td>III</td>
<td>1</td>
<td>1,1</td>
</tr>
<tr>
<td>IV</td>
<td>2</td>
<td>0,9</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>0,9</td>
</tr>
<tr>
<td>VI</td>
<td>1</td>
<td>1,2</td>
</tr>
</tbody>
</table>

The experiment IV begins at 2-fold prevalence of the teacher over students. The inequality is corrected during 2 steps by retardation of the teacher and acceleration of the students. It is an example the Breach-policy. The experiment V demonstrate the Relief-policy under the same parameter set as the experiment IV.

In the experiment VI the teacher is twice slower than the student, but the initial difference is smoothed quickly out due to mutual acceleration of the student and their teacher even permitting them to overtake other experiments.

A sure conclusion may be made that simulation program may be recommended for lecture planning.

5. Problem analysis and actuality

Undeniable importance of software implies high actuality of programming and of all skills that support the programming, program comprehension among them. As opposed
to other skills, the program comprehension does not manifest itself directly, through such artefacts as debugged programs, protocols of execution, data bases and so on, but indirectly, through outcomes of skills, that take the comprehension as their inner premise. Respectively, the comprehension may be retrieved and enhanced in two modes: directly, internalizing speech of a teacher, or indirectly, executing programming tasks, as in experiments of A. Shargabi et al [5].

The indirect mode is experienced freely by self-preparatory training of a student and during laboratory session in the high school. But the direct mode requires a special environment that can be created only in lecture format.

Till now the program comprehension learning does not exists as academic discipline. Neither the Join Task Force on Computing curricula [40] nor curricula on software engineering of Moscow aviation institute (in [41] for bachelors and in [42] for masters) contain a discipline dedicated to acquiring program comprehension. But 5-6 disciplines, related to the software engineering in the syllabi of some chairs in MAI, with total 220 hours of lessons in lecture format and total 900 hours of laboratory sessions in any case contribute to learning of program comprehension.

Figure 8 Problem environment of program comprehension
Therefore, the actuality of solutions proposed in the paper depends on their feasibility during the lectures. The papers message consists in offering the methodology and corresponding tool for making a choice pertaining to intensions and style of the particular educational course. The paper claims to contribute solving some problems shown in Figure 8.

Problems of teacher’s speech styling, synchronization and lecture notes preparation are derived from the general problem of program comprehension teaching, that is partially solved by proposed didactic parsing technology. Interdependencies of the didactic parsing, cooperative learning and cooperative teaching are considered in previous items of the paper. Problems of teaching in lecture format are recently studied by P. Vlachopoulos, S. Jan in [43], Gurpreet Kau in [44] and M. R. Edwards et al, in [45] paying special attention to the role and impact of the lecture format onto the audience.

6. Future work
To studies that can be advanced on the base of current contribution relate:
• definition, monitoring and control of comprehension quality,
• investigation of general regularities at teaching of program comprehension,
• refinement of methods for program comprehension monitoring and control,
• comprehension by writing programs as proposed by M. J. Davidson [46],
• differences between program comprehension and algorithm comprehension,
• using invasive tools for checking the results of experiments with comprehension as in the study of comprehension experiments of N. Peitek et al [47].

Conclusion
Proposed viewpoint at the program comprehension not only as at a process but also as at a cognitive state which is accountable to the owner, perceivable and expressible by the owner has given way to creation of an environment, methods and tools enabling to positively impact the processes of comprehension attainment. In particular, they permit to control the program comprehension level at a particular audience, adjust style of teaching most to it applicable, precise plans of lectures in the area of program engineering.

REFERENCES


34. Site “Moodle” (2022). Available at: https://moodle.org/?lang=en (accessed 21 December 2022)


41. Curriculum on direction 09.03.04 Software engineering 09.03.04 Moscow Aviation Institute MAI(NIU) 202223. (2022). Available at: https://cp.mai.ru/~jql7f (accessed 22 December 2022)

42. Curriculum on direction 09.04.04 Software engineering 09.0.04 Moscow Aviation Institute MAI(NIU) 202223. (2022). Available at: https://cp.mai.ru/~b0C2l (accessed 22 December 2022)


Информация об авторе
Гагарин Андрей Петрович
(Россия, Москва)
Кандидат технических наук, профессор по кафедре информатики; профессор кафедры вычислительные машины, системы и сети
Московский авиационный институт (национальный исследовательский университет)
E-mail: gagarin_ay@outlook.com
ORCID ID: 0000-0002-0929-2834

Information about the author
Gagarin Andrej Petrovich
(Russia, Moscow)
Cand. Sci. (Tech.), Professor at the Department of Informatics; Professor at the Department of Computers, Systems and Networks
Moscow Aviation Institute (National Research University)
E-mail: gagarin_ay@outlook.com
ORCID ID: 0000-0002-0929-2834