Creation of High-Quality Abstractions in Software Engineering

Alexey Razumowsky [1†],
razumowskya@gmail.com
Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Russia

Abstract
Abstraction is the cornerstone of ideal software engineering (SWE). This paper discusses a problem of forming reasonable generalizations, representations and descriptions in various software development processes through the prism of poor-quality (rash, unconsidered, uncertain and harmful) abstractions. To do this, emphasis is made on an induced strategic connection between the required abstraction and its compact specific formulation based on existing research and the author's introspective experience. A software aim point and characteristic preservation of the solution integrity is the subject of the best formulation and a program module or code associated with it. Moreover, a personal attitude expressed by personal interest, motivation and creativity, is proclaimed to be a fundamental factor in successful software development.

Keywords:
abstraction; purposefulness; integrity; software engineering.

1. Introduction

The real essence of the problem in software (SW) analysis and design is expressed using abstraction. Any model is based on generalization of the structure of an object, abstraction from its unimportant or unknown details, and sufficient distraction from detailed physical content of processes occurring inside it or of elements of its context.

Abstraction is one of the main principles of the object-oriented programming (OOP) to software development. However, inept abstractions are often obtained of data, types or libraries in case of insufficiently thoughtful study of the problem. Using low-quality abstractions, it is impossible to obtain a reliable result both for a software project itself, and even more so at the level of its functioning and application.

When using computing devices, it is impossible to do without abstraction and translate an individual idea of an object into a description “understandable” by a computer. In addition, the very process of obtaining abstraction is considered to be the main means of addressing the complexity of software. Such complexity appears in two forms: first, objective complexity associated with the multiplicity of program elements, as well as their interweaving with each other. Secondly, difficult perception of information which is rooted in the cognitive processes of a person. In this Article, we will focus on the problem of the information perception complexity in software development. The complexity of perception, correlation, understanding or association is practically not given any attention, and yet this is a fundamental problem in achieving any satisfactory engineering solution. In addition, this Article was conceived as a starting point in the development of a new software engineering methodology backed by individual creative activity of a person to create non-deterministic software and hardware solutions. For the best creative effect, special comfortable conditions should be provided that take into account such individual cognitive characteristics as attention, assimilation, understanding, speed and quality of intellectual reaction. The main barrier of cognitive complexity is a small amount of a person’s short-term memory; it is only 7+-2 elements according to Miller [34]. This means that if there is, for example, a larger amount of data in the field of view, the development and adoption of a decision by a person becomes much more difficult, and sometimes completely stops.

Thus, abstraction should be implemented by continuously reconciling the source information and its abstract expression. It is important to ensure that the number of coordinated information elements is small and does not exceed the Miller number. For this purpose, decomposition is used which helps the abstraction process by extracting parts from the whole. Each part can then be considered separately. Abstraction and decomposition replace each other until the original task is reduced to a set of subtasks, the solution for which is known [31]. It should not be forgotten that the final design stage needs to get a full and complete model, that is, all its parts must be combined and well coordinated. The problem of creating high-quality software possibly lies in this

Manuscript received July 5, 2023
Manuscript revised July 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.7.22

need for mandatory final consistency of abstractions among themselves. A reliable, high-quality program will not develop by itself, as if easily, by chance. On the contrary, consistent, detailed abstraction based on clear, individually easily perceived and specific descriptions and definitions can lay a firm foundation for development success.

Historically, the development of approaches to information abstraction goes with attempts to reduce software complexity and development. Different attitudes to abstraction are known that differ in the emphasis of the idea of abstractions, as well as the ways to obtain them:

- "abstraction is manifested in finding similarities" [18];
- highlight some properties and omit others [46];
- separate abstraction from the implementation method [8];
- separate narratives [2];
- define conceptual boundaries from the point of view of the observer [10].

It is clearly seen that such an understanding of abstraction is beyond the bounds of associating objective and, moreover, cognitive complexity with a person's ability to overcome it. Next, let's try to trace what such lack of foresight led and continues to lead researchers who are struggling with complexity while increasing the power of technological engineering with no regard for the general and especially individual human perception, recognition and understanding of things.

2. Literature review and analysis

Surprisingly, there is no evidence in the vast software development literature about negative impact of abstraction on software analysis, design, and programming. Even more surprising that there are almost no studies on inept abstraction by psychologists or educators. One of the rare materials on related topics is the paper [27]. It studied a social aspect of an abstract view of morality. It is argued that a more abstract view of the moral issue increases intellectual deviousness. In other words, an increase in the abstraction level can lead to a distortion of view of the subject both in terms of knowledge about it and reduction of responsibility for the abstraction result.

We were also able to find evidence of the negative impact caused by inept abstraction in multi-agent enterprise environment [26]. This paper draws attention to the fact that those at different levels of management in a multi-level enterprise environment are guided by rules of different abstraction levels, which creates a correspondence problem, namely, rules are more vague and inaccurate at a high level, and requirements are specific at a low level. To translate this conclusion into the language of software analysis and design, we note the importance to maintain specific descriptions, rules, and requirements in the project which include more complete and purposeful content.

The paper [41] proposes a method of joint visual representation of abstractions and non-abstract data. It is argued that such visualization will allow analysts to significantly simplify understanding of information through smooth semantic zoom, that is, a controlled process of simplifying original information.

The very fact that it is necessary to use a special data visualization technique for their better analysis and understanding indicates the need for careful handling of initial information in choosing ways and methods of abstraction.

When analyzing the problem, it is important to understand the classification order before initiating the abstraction process. The paper [48] proposes a classification system that defines the main criteria to choose an abstraction method for subsequent visualization and support of analysis tasks. The authors present a summary table to select appropriate abstraction methods depending on data attributes, a desired form of their presentation, behavior, accuracy and level of detail.

The paper [24] studies the essence of abstraction from the point of view of its practical manifestation in technology, mathematics and everyday life, as well as in software development. The need for multiple abstraction is emphasized which focuses on some details of the subject by ignoring others.

This paper proposes a different approach to abstracting a solution to the problem using specific and closest to everyday life simple words and expressions. Such approach will minimize risks of a poorly organized software development process resulted from inept or inappropriate abstraction.

As noted above, we cannot completely abandon the abstract view of the subject matter of the problem,

since the data must ultimately be transferred to the computer to perform necessary calculations: and the computer will only perceive such formally expressed data as clothed in an “understandable” form. However, we can avoid a complex and highly distributed abstraction, postpone the moment of abstraction, and use clear, informative and specific descriptions and images. It is also important to position specific descriptions together [41] with their abstraction in a single visualization context.

At the time of ancient Babylon, its inhabitants used verbal descriptions of unknown quantities, long before symbolic mathematics was invented, and mathematicians used Boolean statements before Boole formalized the laws of thought [9]. This testifies to the naturalness of abstraction from reality in an attempt to describe and understand it. Moreover, this indicates the effectiveness of simplification while reducing a certain generality, uniformity or plurality to a simple value and naming or depicting it clearly and distinctively. Historically, a significant number of works are generally devoted to the study of abstractions and their application both in information science and cognitive science. First of all, it is necessary to highlight papers cited by many researchers of software development, programming languages, artificial intelligence algorithms and approaches to decision making [16,23,42,49,52,54]. The papers [1,3,5,15,22,25,29,32,44,45,55] are devoted to abstractions formed and used in the algorithm development and software engineering. To study abstractions from the point of view of the cognitive approach, we can refer to the papers [14,19,35,47,57]. Not all of these studies directly address the complexity of objects or solutions, but abstraction is needed as a means of simplification and concentration, so there is often evidence of a superficial understanding of the ways and goals of abstraction. For example, in a quite popular book [7], there is a tendency for reuse based on the existing abstraction. It is a mistake to think that old abstractions are always well suited to new solutions. This is an illusion of similarity. Later in this book, we find the authors’ assurances that hidden implementation details and a formed "black box" are a natural architectural solution. Perhaps, it looks like this from the point of view of impersonal technology, but the question arises: how to interpret the “black box” elements at the right time to restore specifics? The authors have no answer to it. We can agree with the authors that software architecture is primarily an abstraction that "suppresses" details of the elements. However, it would then be necessary to determine possibilities to combine parts into a whole, but the authors fail to mention this again.

Some ideas about abstractions are highly questionable. For example, in the book [10], it is stated with reference to Descartes that people usually have an object-oriented view of the world. This is apparently a gross exaggeration since the very activity of a person including his thinking is consistent and therefore cannot afford to disperse over objects. A wish is also typical to allegorically anthropomorphize the abstraction. For example, in the paper [30] there is a following definition: links between modules are their representations of each other. Such a substitution which seems to serve to better understand the interdependence of modules or objects leads to a direct deception of an inexperienced software engineer. Again, by giving specific content to an abstract connection, it will not be easy for a person to refuse such a stereotype in the future, that is, to put his own interpretation of resulting interdependence.

Nevertheless, the understanding is gradually emerging today that abstraction is not only a technology problem where it can be difficult to return to a specific view of the subject. Along with papers [26,41], a tendency to combine an abstraction process with interpretation is already clearly manifesting itself. The paper [56] presents the “Model-with-Example” approach, which combines the modeling of abstract interaction with given visualization, which, according to the authors, increases the development efficiency.

Today, understanding the importance of a clear abstraction leads to the unification of textual representations and metaphors [51]. There are also papers related to the direct interpretation of abstractions [43]. This suggests that it is not enough today to simplify the idea of the problem, but important to ensure the ease of adequate reverse interpretation.

A very remarkable thing to understand the abstraction was discovered in the paper [33]. It compares reactions to two concepts: "triangle" and "three-sided polygon". The authors emphasize that individual experience that determines understanding of the subject is associated with specific, and not abstract (prototype) representations.

A serious error, in our opinion, is abstraction for the purpose of reuse. We have already pointed out this error made by the authors of the book [7]. As the conviction is strong that reuse is the essential panacea of software development, so the frustration is with reality. In the paper [37], the authors empirically come to the conclusion that it is impossible to reuse abstractions for projects of different types. Creators of business components also face the challenge when reuse is required. Design patterns should also be separately noted as a sublimating abstraction of the mode of action. Such abstraction encounters resistance when understanding details as parts of a whole, software architecture and difficulty of software maintenance. Even when software is discussed for mechanical machines, it is noted that in general, the results obtained for a mechanical system cannot be reused by other systems" [12]. In this paper, the authors give away that the choice of certain characteristics depends on human knowledge. The data reuse when creating humanoid robots is excluded in the paper [50], since a specific movement chosen from the database is associated with large resource costs. A similar conclusion is drawn in the paper [39]: «most of the proposed architectures are special-purpose implementations that lack modularity or standardization, and cannot be reused».

Finally, in the paper [4], one can see understanding of the best abstraction as the representation of a specific, direct, associative and not pre-planned solution. This surprisingly emphasizes the need to focus on human knowledge, skills, experience, features of thinking, perception and creativity in general when choosing an abstraction.

Thus, we are now ready to express the concept of the abstraction complexity in terms of its relevance (confidence), and then to propose a way to harmoniously combine specific and abstract descriptions.

3. Methodological principles

In At first, we will define the abstraction complexity somewhat superficially, as a quantitative set of information details correlated with a subjectively perceived or observed object, as well as links between them. The more such elements in the abstract representation, the more complex the abstraction is.

Next, we will see how the process of abstraction affects subjective reality. The method of our research will be based on Poincaré's principle of convenience proposed by him for geometry: "one geometry cannot be more true than another; one or another geometry can only be more convenient" [36]. The original subject is transformed into a certain idea about it under the influence of the need for its naming, design, description or explanation. A car can thus become a parallelepiped, a railway can become a spline, an industrial plant can become a set of clusters, and a machine tool can become a sequence of executive programs.

3.1 Purpose tracking

Every meaningful activity has a purpose. It organizes, often implicitly, the direction of thought by linking it to meaningful action and intended outcome.

The original essence of the subject, its implicit completeness (incompleteness) and indefinite integrity (fragmentation and misconceptions) turns into something fundamentally different under the influence of attraction or striving for a specific goal which a person observes (thinks, imagines) or loses sight of (forgets, changes significance of, or blurs). In other words, this is where the source of a misconception about the subject lies. Consequently, the abstraction process requires special supporting elements introduced into it which contribute to a verified correlation of created abstractions with their real images.

It is also necessary to point out the fact of strict individualization of abstraction, since any convenience is purely subjective. And finally, in abstracting, we should strongly avoid secondary stereotypization processes or correlating the study subject with a certain prototype or pattern. This is necessary due to the inadmissibility of missing important individual nuances and roughness. A person can only use such patterns for himself or within a narrow group of allied specialists while acting very carefully. In addition, care and responsibility are required so that the idea of reality is not distorted by such pattern.

For the initial description of the problem, as well as its features and relationships with the

environment, it is important to use concise and, at the same time, understandable and specific natural language expressions aimed at identifying the ultimate essence of the subject. Such expressions may, at their best, contain two or three terms by conforming to their everyday use, while indicating the main goal of the solution. A representation of a goal-oriented abstraction scheme is shown in Figure 1.

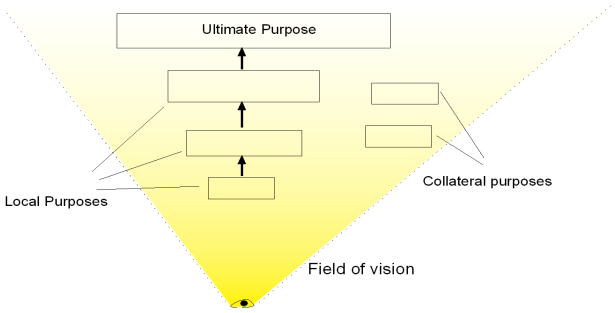

Fig.1 Schematic illustration of purposive abstraction

Consider the following example.

To formulate the main goal of an algorithm for a robot courier required to move from point A to point B, we should highlight the limiting fact of the impossibility of such a move. The algorithm main goal will then receive the following expression: "avoid collision". A corresponding functional abstraction might look like this:

```
bool avoid(){
 if(delta_x(distance) || delta_y(distance))
    return false;
  return true;
}
```

where, distance is the distance that characterizes occurrence of the limiting collision situation, and delta_x and delta_y are approximation measures along the X and Y axes.

It is also possible to improve the abstraction by refining the original wording: "avoid collision". Then the semantics of negation is easily converted to a comparison operator. The comparison operator combined with the logical choice operator will instantly complete the abstraction:

```
bool avoid(){
 if(delta_x< distance || delta_y< distance)
    return false;
  return true;
}
```

One more abstraction which turned out as if by itself also attracts attention. It is about expressing the change in the robot's state (position) not by a velocity vector but by the path increment. The language of the route or the action plan is used by a person confidently and every day. By using everyday expressions while abstracting from reality, we get rid of another problem of incorrect abstraction, namely, what is typical for natural behavior will become more adequately correlated with the study subject of the problem, which means it will be of great benefit, even if the price is an excess resource (for example, software-based and technical computing).

In the above example, we can observe how an accurate and concise formulation of the problem requirements that is consistent with the actual goal can reduce the search for an algorithm or software engineering solution to elementary and clear (understandable in ordinary sense) details. And vice versa, false identity of concepts to real subjects (for example, a velocity vector to a robot's movement) from which an abstraction is formed is ready to destroy a solution or seriously complicate it, which often happens in real projects.

The example also demonstrates how a developed program code "covers" the pursued goal and includes it in a solution. Such an abstraction can be called purposeful, an abstraction that coordinates at each moment of time the fact of achieving the goal with the means to achieve it. Such agreement makes an abstraction acceptable.

Another example of a purposeful abstraction is redundant data structures used to develop algorithms for complex computer graphics, such as for finding equidistant curves and surfaces or shading complex contours. Redundancy as a means of goal setting is important and applicable to improve cognitive capabilities of information perception or diverse but related content, as well as to increase attention to critical details [40].

It should always be remembered and understood that the fight against complexity for which an abstraction is used must be in the context of the "principle of preserving complexity including infinite complexity of any reality" [6].

3.2 Integral abstraction

The decision-making process that is consistent with responsibility for such a decision requires the completeness of a visible picture. However, this picture should be described. The process of searching and selecting such a description is strongly affected by the language a person speaks. There are studies on the difference in the perception of time between native speakers of Chinese and English [11]. This also indirectly confirms the cognitive basis of any correct decision, since the affluence of a language affects the color range of representations and possibilities of the subject. Accordingly, the same thing can be expressed with a significant difference using terminological and phraseological techniques depending on language forms, vocabulary and intonations.

Linking words into phrases and sentences, highlighting main and key expressions, dividing into details and inverse associations, or searching for analogies and examples - this is all determined by functions of our brain and nervous system as a whole. In this regard, today's ideas about brain functions are not very comforting. Thus, according to the paper [28], the brain is engaged in forecasting. Human perception depends on “hierarchical predictive coding” [13,20,53]. As A. Clark writes [17]: “We structure our worlds and actions so that most of our sensory predictions come true.” Doesn't this mean that we create an error and then justify it? Is it even possible to minimize our errors? How to correct them?

There is a whole layer of studies devoted to answering the question: how to solve problems. Let's point out the important ones. First, this is the bestseller by G. Polya [38] which details the order and context of solving mathematical problems. One of his main ideas is that generalization simplifies implementation. Or, to put it differently, seeing the whole picture, to catch its detail. Second, this is the book by A. Goldman [21] which proposes a simulation theory stating that special mental states are created in the mind during thinking processes that resemble or tend to resemble those that are their goals. These states are then assigned or projected to goals. A holistic space of a goal and paths to it are formed as various contexts (Fig. 2). Thus, it is important to regularly align formed and implemented right abstractions with their goals, especially as goals can independently or intentionally change.

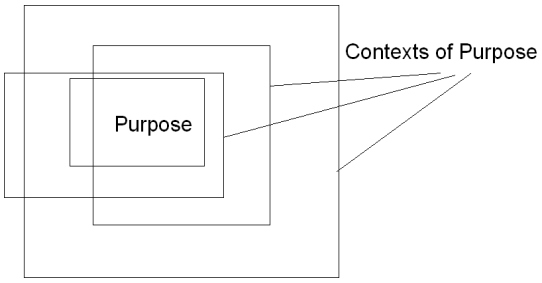

Fig.2 Schematic illustration of holistic abstraction

Consider an example that illustrates the representation integrity of an abstracted subject with the principles of solving mathematical problems by G. Poya [38]: understand, plan, execute and look back. We assume that a text array of structured data is given, that is, data that carry certain information, and not meaningless bits. Then it can be argued that syntactic information is comprehended into a single emergent quality. In the case of incompletely defined data with a lack of important details, the lack of information is subject to causal circumstances that are often overlooked. Then a valuable meaning arises when information space is expanded. The resulting abstraction becomes complete. Such a holistic abstraction is stable both for new elements added to the algorithmic and structural content and in the perception of holistic data. The complexity of such a decision is dependent on the meaningfulness of each action taken by a software engineer: when acting according to a template, the complexity can increase uncontrollably, and on the contrary, the complexity will be under control in conditions of individual decisions. Thus, complexity is subject to individual control.

4. Results and Discussion

Abstraction as a process of implementing software and hardware solutions has two orthogonal directions: any action according to a technological template, for example, in line with SADT standards or flexible Agile methodology; or an activity associated with searching for individual characteristics of the problem and considering available resources or other exceptional circumstances and restrictions.

Our study has shown, firstly, that researchers of formed abstractions bypass or only casually mention the significance of purposefulness in software development processes. Secondly, there is no importance directly related to interpretations to maintain the integrity of development processes and their outputs.

Based on the Poincaré convenience principle, there are two main vectors of reliable software development. Firstly, to meet the original goal (as well as its subgoals). An observed clearly visible goal will only allow using those resources and operations that are best aligned with this goal, that is, more comfortable for creative perception. Secondly: to observe the integrity of the development and its individual acts. The integrity, like purposefulness, is able to equalize and balance unjustified subjective assessments and abstractions created from them. Since there is a direct connection between insight into the subject and its successful definition, striving for integrity and tracking the goal will also correct unsuccessful descriptions and plans using mobile corrections or replacements.

5. Conclusion

The abstraction process definitely consists of two stages: removal from the study subject specifying its essential properties, and then formalization of the created information content as an information model. However, the specification and description of properties is unable in itself to form a clear picture of the solution and certainly its step-by-step implementation. Moreover, search for similarity [18], differentiation of properties [46], separation of meanings [2], or vice versa, finding conceptual boundaries [10] will not also allow one to form a clear and distinct picture of a solution. All of these together and separately can give some superficial tone to an attempt of software development. Individual technological requirements will only make the design solution heavier.

Therefore, we have proposed here a different view of the problem of right abstractions as entities that simultaneously combine the content capacity of the subject and possibilities of a perspective view of the subject by a software engineer. In other words, a person must receive sufficient informational support when looking at an abstraction. This means that the right abstraction will allow us to "see" everything, and to head for a goal without losing track of it.

The final conclusion is that a holistic and purposeful abstraction does not fully ensure a successful software development project, however it will, in any case, allow us not to deviate from the target path and not to waste energy on any unhelpful technological attempt.

References

[1] Abbot, R. J. (1987). Knowledge abstraction. Communications of the ACM, 30(8), 664-672.

[2] Abelson, H., Sussman, G. J., & Sussman, J. (1996). Structure and interpretation of computer programs. Justin Kelly.

[3] Aho, A., & Ullman, J. (2022). Abstractions, their algorithms, and their compilers. Communications of the ACM, 65(2), 76-91.

[4] Aldalur, I., Winckler, M., Díaz, O., & Palanque, P. (2017). Web augmentation as a promising technology for end user development. In New Perspectives in End-User Development (pp. 433-459). Springer, Cham.

[5] Amahan, P., & Sanqui, R. (2021). Syntax to Syntax: Assessment of Orthogonality in the Design of Object-oriented Programming Languages using Code Listing Method. In 2021 The 4th International Conference on Software Engineering and Information Management (pp. 52-55). DOI:10.1145/3451471.3451480.

[6] Babichev A.V., Butkovskiy A.G., Pohjolainen Seppo, (2001), Towards Unified Geometrical Theory of Control. Nauka, Moscow.

[7] Bass, L., Clements, P., & Kazman, R. (2003). Software architecture in practice. Addison-Wesley Professional.

[8] Berzins, V., Gray, M., & Naumann, D. (1986). Abstraction-based software development.

[9] Birkhoff, G. (1969). Mathematics and psychology. SIAM Review, 11(4), 429-469.

[10] Booch, Grady (1997). Object-Oriented Analysis and Design with Applications. Addison-Wesley. ISBN 978-0-8053-5340-2.

[11] Boroditsky, L. (2001). Does language shape thought?: Mandarin and English speakers' conceptions of time. Cognitive psychology,43(1), 1-22.

[12] Cabrera, D., Sancho, F., Li, C., Cerrada, M., Sánchez, R. V., Pacheco, F., & de Oliveira, J. V. (2017). Automatic feature extraction of time-series applied to fault severity assessment of helical gearbox in stationary and non-stationary speed operation. Applied Soft Computing, 58, 53-64.

[13] Carbajal, G. V., & Malmierca, M. S. (2018). The neuronal basis of predictive coding along the auditory pathway: from the subcortical roots to cortical deviance detection. Trends in Hearing, 22. DOI:10.1177/2331216518784822

[14] Carden, T., Goode, N., & Salmon, P. M. (2019). Accounting for memes in sociotechnical systems: extending the abstraction hierarchy to consider cognitive objects. Ergonomics, 62(7), 849-863. DOI:10.1080/00140139.2019.1603403

[15] Cardone, F. (2020). From Curry to Haskell: Paths to Abstraction in Programming Languages. Philosophy and Technology. 34(1), 57 - 74. DOI:10.1007/s13347-019-00385-4.
[16] Clancey, W. J. (1983). The Advantages of Abstract Control Knowledge in Expert System Design (No. STAN-CS-83-995). STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.
[17] Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. Behavioral and brain sciences,36(3), 181-204.
[18] Dahl, O. J., Dijkstra, E. W., & Hoare, C. A. R. (1972). Structured programming. Academic Press Ltd..
[19] Demetriou, A. (2020). Abstracting abstraction in development and cognitive ability. Behavioral and Brain Sciences, 43. DOI:10.1017/S0140525X19002930
[20] Gilead, M., Trope, Y., & Liberman, N. (2020). Above and beyond the concrete: The diverse representational substrates of the predictive brain. Behavioral and Brain Sciences, 43.
[21] Goldman, A. I. (2006). Simulating minds: The philosophy, psychology, and neuroscience of mindreading. New York, NY, US: Oxford University Press.
[22] Han, L., Song, M., & Pedrycz, W. (2021). An Approach to Determine Best Cutting-points in Group Decision Making Problems with Information Granules. In 2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (pp. 1-6). IEEE.
[23] Hayes-Roth, F., & McDermott, J. (1978). An interference matching technique for inducing abstractions. Communications of the ACM, 21(5), 401-411.
[24] Jackson, M. (2012). Aspects of abstraction in software development. Software & Systems Modeling, 11(4), 495-511.
[25] Kim, Y., Kim, J., Jeon, H., Kim, Y. H., Song, H., Kim, B., & Seo, J. (2020). Githru: Visual analytics for understanding software development history through git metadata analysis. IEEE Transactions on Visualization and Computer Graphics, 27(2), 656-666.
[26] King, T. C., De Vos, M., Dignum, V., Jonker, C. M., Li, T., Padget, J., & van Riemsdijk, M. B. (2017). Automated multi-level governance compliance checking. Autonomous Agents and Multi-Agent Systems, 31(6), 1283-1343. DOI:10.1007/s10458-017-9363-y
[27] Lammers, J. (2012). Abstraction increases hypocrisy. Journal of Experimental Social Psychology, 48(2), 475-480.
[28] LeDoux, J. E. (2020). How does the non-conscious become conscious?. Current Biology, 30(5), R196-R199. DOI:10.1016/j.cub.2020.01.033
[29] Lennon, B. (2021). Foo, Bar, Baz…: The Metasyntactic Variable and the Programming Language Hierarchy. Philosophy & Technology, 34(1), 13-32. DOI:10.1007/s13347-019-00387-2
[30] Liskov, B. H. (1972, December). A design methodology for reliable software systems. In Proceedings of the December 5-7, 1972, fall joint computer conference, part I (pp. 191-199). ACM.
[31] Liskov, B., & Guttag, J. (1986). Abstraction and specification in program development (Vol. 180). Cambridge: MIT press.
[32] Liskov, B., & Zilles, S. (1977). An introduction to formal specifications of data abstractions. Current trends in programming methodology, 1, 1-32.
[33] Lupyan, G. (2017). The paradox of the universal triangle: concepts, language, and prototypes. The Quarterly Journal of Experimental Psychology, 70(3), 389-412.
[34] Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological review, 63(2), 81.
[35] Park, S. A., Miller, D. S., Nili, H., Ranganath, C., & Boorman, E. D. (2020). Map making: constructing, combining, and inferring on abstract cognitive maps. Neuron, 107(6), 1226-1238.
[36] Poincaré, H. (1905). Science and hypothesis. Science Press.
[37] Pollo-Cattaneo, M. F., Pesado, P., Britos, P., & García-Martínez, R. (2017). Process Model Proposal for Requirements Engineering in Information Mining Projects. In Colombian Conference on Computing (pp. 130-145). Springer, Cham.
[38] Polya, G. (2004). How to solve it: A new aspect of mathematical method (Vol. 85). Princeton university press.
[39] Randelli, G., & Nardi, D. (2010). Introducing ontology best practices and design patterns into robotics: USAREnv. In Proceedings of the 2010 conference on Modular Ontologies: Proceedings of the Fourth International Workshop (WoMO 2010) (pp. 67-80). IOS Press.
[40] Razumowsky, A. I. (2019). Creativity-oriented software development. Amazonia Investiga, 8(22), 629-639.
[41] Rind, A., Wagner, M., & Aigner, W. (2019, October). Towards a structural framework for explicit domain knowledge in visual analytics. In 2019 IEEE Workshop on Visual Analytics in Healthcare (VAHC) (pp. 33-40). IEEE. DOI:10.1109/VAHC47919.2019.8945032
[42] Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. Artificial intelligence, 5(2), 115-135.
[43] Sacha, D., Al.Masoudi, F., Stein, M., Schreck, T., Keim, D. A., Andrienko, G., & Janetzko, H. (2017, June). Dynamic visual abstraction of soccer movement. In Computer Graphics Forum (Vol. 36, No. 3, pp. 305-315).
[44] Sakr, S., Bonifati, A., Voigt, H., Iosup, A., Ammar, K., Angles, R., ... & Yoneki, E. (2021). The future is big graphs: a community view on graph processing systems. Communications of the ACM, 64(9), 62-71.
[45] Shankar, K. (1984). Data Design: Types, Structures, and Abstractions. C. Vick and C. Ramamoorthy New York: Van Nostrand Reinhold.
[46] Shaw, M. (1984). Abstraction techniques in modern programming languages. IEEE software, (4), 10-26.
[47] Shillcock, R. (2020). A modern materialist approach to abstraction, concreteness, and explanation in cognition. Behavioral and Brain Sciences, 43. DOI:10.1017/S0140525X19003066
[48] Shurkhovetskyy, G., Andrienko, N., Andrienko, G., & Fuchs, G. (2018, February). Data abstraction for visualizing large time series. In Computer Graphics Forum (Vol. 37, No. 1, pp. 125-144).
[49] Stern, C. R., & Luger, G. F. (1997). Abduction and abstraction in diagnosis: a schema-based account. J. Expertise in context, 363-381.
[50] Takano, W., & Nakamura, Y. (2015). Symbolically structured database for human whole body motions based on

association between motion symbols and motion words. Robotics and Autonomous Systems, 66, 75-85.
[51] Viola, I., & Isenberg, T. (2017). Pondering the concept of abstraction in (illustrative) visualization. IEEE transactions on visualization and computer graphics, 24(9), 2573-2588.
[52] Vreeswijk, G. A. (1997). Abstract argumentation systems. Artificial intelligence, 90(1-2), 225-279
[53] Xia, L., & Collins, A. G. (2021). Temporal and state abstractions for efficient learning, transfer, and composition in humans. Psychological review. 128(4), 643 - 666. DOI:10.1037/rev0000295.
[54] Yang, Q. (2012). Intelligent planning: a decomposition and abstraction based approach. Springer Science & Business Media.
[55] Yin, H., Liu, P., Liu, K., Cao, L., Zhang, L., Gao, Y., & Hei, X. (2020). NS3-AI: Fostering artificial intelligence algorithms for networking research. In Proceedings of the 2020 Workshop on ns-3 (pp. 57-64)..
[56] Ziegler, D., & Peissner, M. (2018, July). Modelling of Polymorphic User Interfaces at the Appropriate Level of Abstraction. In International Conference on Applied Human Factors and Ergonomics (pp. 45-56). Springer, Cham.
[57] van Houwelingen, G., van Dijke, M., van Hiel, A., & De Cremer, D. (2021). Cognitive foundations of impartial punitive decision making in organizations: Attribution and abstraction. Journal of Organizational Behavior, 42(6), 726-740. DOI:10.1002/job.2480