MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Inginerie Software și Automatică

Admis la susținere
Sef departament: Fiodorov I., dr., conf. univ.

____

"__" ________ 2022

Modele software pentru soluții de învățare nelingvistică

Teză de master

Student: ________________ **Brînzan, Leon, gr. IS-211M**

Coordonator: ________________ **Ciorbă, Dumitru, conf. univ., dr.**

Consultant: ________________ **Catruc, Mariana, lect. univ.**

Chisinau, 2023

Brînzan, Leon

Software models for non-linguistic learning solutions

Abstract

Software engineering is a multifaceted discipline. Mastering it requires memorizing a lot of factual information, knowledge of at least one programming language, and learning a particular set of skills that enable the learner to tackle complex engineering tasks. Programming can only be learned by solving problems specifically designed to develop these kinds of skills. This makes efficiently teaching software engineering difficult. The traditional way of teaching programming heavily relies on *content delivery*. Traditional programming courses do not go too far from the lecture form, failing to innovate learning, only supplementing information delivered during lectures with various visualization techniques, and only sporadically. They do not illustrate the relations between different concepts presented to students from one lecture to the next, leaving it up to students to infer those connections. This paper demonstrates how teaching programming techniques without using textual explanations can be more effective. It identifies several key methods of achieving that, and showcases a few aspects of designing software that allow teaching major programming concepts without using textual information, appealing to students' *computational thinking* abilities. Using methods described in the first part of this work, a software prototype is developed showcasing a non-textual interactive approach to teaching *engineering thinking*.

Keywords*—non-linguistic learning; educational software; cognitive load theory; computational thinking;*

Brînzan, Leon

Modele software pentru soluții de învățare nelingvistică

Adnotare

Ingineria software este o disciplină cu mai multe fațete. Stăpânirea acesteia necesită memorizarea multor informații concrete, cunoașterea a cel puțin unui limbaj de programare și învățarea unui anumit set de abilități care îi permit studentului să abordeze sarcini complexe de inginerie. Programarea poate fi învățată doar prin rezolvarea unor probleme special concepute pentru a dezvolta aceste tipuri de abilități. Acest lucru face dificilă predarea eficientă a ingineriei software. Modul tradițional de a preda programarea se bazează în mare măsură pe *livrarea conținutului*. Cursurile tradiționale de programare nu se îndepărtează prea mult de forma de prelegere, nereușind să inoveze învățarea, doar completând informațiile livrate în timpul prelegerilor cu diverse tehnici de vizualizare și doar sporadic. Ele nu ilustrează relațiile dintre diferitele concepte prezentate studenților de la o prelegere la alta, lăsând la latitudinea studenților să deducă acele conexiuni. Această lucrare demonstrează modul în care predarea tehnicilor de programare fără a folosi explicații textuale poate fi mai eficientă. Textul tezei identifică câteva metode cheie pentru a realiza acest lucru și prezintă câteva aspecte ale proiectării software-ului care permit predarea conceptelor majore de programare fără a utiliza informații textuale, apelând la abilitățile de *gândire computațională* ale studenților. Folosind metodele descrise în prima parte a acestei lucrări, este creat un prototip de software care prezintă o abordare interactivă non-textuală a dezvoltării *gândirii inginerești*.

Cuvinte-cheie*—învățare nelingvistică; produs software educational; teoria încărcării cognitive; gândirea computațională;*

Table of contents

Introduction

Since the time of its inception the domain of software engineering has been evolving at an accelerating pace. Each year more and more people decide to pursue a career in software development. According to the US Department of Labor, software developers make up 1622200 (1.6 million) jobs in the country, with this occupation being the fastest growing (25% growth rate compared to an average 5% growth rate in other occupational fields) [BLS]. The number of software specialists in the US is expected to reach 2 million by the year 2031. At the same time, according to some sources working in the field of software development, software quality is in a steady decline year after year[1] (both, directly compared to the quality of the earlier software projects, and relative to the pace at which hardware complexity and performance increases). There's a growing demand for software engineers as software penetrates more and more domains. As a result, software engineering training becomes more important with each passing year. As more people enter the software engineering domain with the goal of becoming professional developers, the entrants' average level of expertise in related domains (mathematics and physics, primarily) naturally keeps dropping. In such a context, this paper poses – with the intention of answering – three questions.

The first question that needs to be answered aims at the fundamental forms of teaching that have become prevalent in modern education. Education largely revolves around the tried-and-true practice of gathering groups of people in a room and reading a lecture to them. Teachers are required by the curriculum to cover a certain amount of content to prepare their students for subsequent courses or examinations. Even though lectures can make up only part of a course (with other parts being dedicated to hands-on practice in the form of laboratory work or seminars where students drive the learning process), it is a significant part that students tend to pay the most attention to. As such, lectures become the core of each course and shape the students' impressions of the subject. This often has drastic consequences for students' success in later courses that rely on material from previous courses, even more so in cases where students are expected to have mastered certain skills during a course [OLSSON]. Information passed from instructors to students during lectures is represented in textual form. This requires the ability to comprehend textual information to be on a high enough level. But text isn't the only way to communicate ideas to students. In fact, it may not be the best form for that purpose. Thus, a question should be asked: are visual forms of presenting information better than textual forms in the context of teaching engineering? There is a plethora of evidence that points at that being the case. A number of research papers dealing with the subject will be discussed in the first chapter of this work.

[1] The Thirty Million Line Problem

The second question deals with the issue of teaching people programming through different paradigms, the choice of a paradigm usually being an afterthought – by the virtue of picking an existing programming language, with its quirks and concepts. This paper argues that the order, in which those two things (programming language and programming paradigm) are considered, should be reversed. And before that first choice is made, one question has to be answered: is procedural programming better than structured programming better than functional programming better than object-oriented programming?

The third question deals with the application of interactive tools to the domain of teaching. It could be formulated as follows. Does interactivity impact in a positive manner the understanding of the issue that is being explored by learners? By extension, it needs to be determined what kind of interactivity specifically boosts understanding (simple visualization techniques; gamification; use of virtual or augmented reality etc.).

This paper cannot hope to definitively answer all three questions posed above. Rather, this paper will provide several concrete examples of how software enables students to learn the "spirit of engineering and problem solving"[2]. It argues that teaching programming without using textual explanations can be more effective in some contexts, and identifies several key aspects of using software that allow teaching major programming concepts without using textual information in order to make the whole learning process more efficient.

[2] Talk: Video Games and the Future of Education

References

1. ABELSON, H., SUSSMAN, G. J., SUSSMAN, J., HENZ, M., WRIGSTAD, T. et al. *Linear Recursion and Iteration*. In: Structure and Interpretation of Computer Programs, Comparison Edition [online], ©2022 [accessed 17.12.2022]. Link: https://sicp.sourceacademy.org/.
2. ALBANO, G., and FORMATO, F. *E-learning from Expertize: a Computational Approach to a non-textual Culture of Learning*. In: Advanced Learning Technologies Conference [online], 2001, pp. 241-242, doi: 10.1109/ICALT.2001.943911 [accessed 17.11.2022]. Link: https://ieeexplore.ieee.org/document/943911.
3. AS. *Developer Survey 2019*: *Open Source Runtime Pains 2019* [online], ActiveState, ©2019 [accessed 17.12.2022]. Link: https://www.activestate.com/resources/white-papers/developer-survey-2019-open-source-runtime-pains/.
4. BASHKIN, V. A. *Logic*. In: Lambda-calculus [online], Yaroslavl State University, 2018, p. 24, UDC: 510.56(075), ББК: 3973.2-018я73 [accessed 11.12.2022]. Link: http://www.lib.uniyar.ac.ru/edocs/iuni/20180409.pdf.
5. BERGER, E. D., HOLLENBECK, C., MAJ, P., VITEK, O., VITEK, J. *On the Impact of Programming Languages on Code Quality: A Reproduction Study*. In: ACM Transactions on Programming Languages and Systems, vol. 41 (4) [online], Association for Computing Machinery Digital Library, 12.10.2019, pp. 1-24 [accessed 17.12.2022]. Link: https://dl.acm.org/doi/10.1145/3340571.
6. BLS, U.S. Bureau Of Labor Statistics, "*Software Developers*, *Quality Assurance Analysts*, *and Testers*. In: Occupational Outlook Handbook [online], ©2022 [accessed 10.10.2022]. Link: https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm.
7. BULIGA, M. *Graphic lambda calculus*. In: Complex Systems 22(4) [online], 24.05.2013, pp. 311-360 [accessed 17.12.2022]. Link: https://arxiv.org/abs/1305.5786.
8. BULIGA, M., KAUFFMAN, L. H. *Chemlambda, universality and self-multiplication*. In: ALIFE 14 [online], July 2014, pp. 490-497 [accessed 17.12.2022]. Link: https://arxiv.org/abs/1403.8046.
9. BULIGA, M. *Chemlambda*, Chorasimilarity [online], Wordpress, ©2020 [accessed 11.11.2022]. Link: https://chorasimilarity.wordpress.com/chemical-concrete-machine/.
10. BURNETT, M., BAKER, M. *VPL: Visual Programming Languages*. Visual Language Research Bibliography [online], Department of Computer Science, Oregon State University, ©2009 [accessed 11.11.2022]. Link: https://web.engr.oregonstate.edu/~burnett/vpl.html.

11. CASTRO-ALONSO, J. C., DE KONING, B. B., FIORELLA, L., and PAAS, F. *Five Strategies for Optimizing Instructional Materials: Instructor- and Learner-Managed Cognitive Load*. In: Educational Psychology Review 33(1) [online], December 2021 [accessed 11.11.2022]. Link: https://www.researchgate.net/publication/349938150_Five_Strategies_for_Optimizing_Instructional_Materials_Instructor-_and_Learner-Managed_Cognitive_Load.
12. CHURCH, A. *An Unsolvable Problem of Elementary Number Theory*. In: American Journal of Mathematics, 58(2) [online], April 1936, pp. 345-363 [accessed 12.11.2022]. Link: https://www.ics.uci.edu/~lopes/teaching/inf212W12/readings/church.pdf.
13. CITRIN, W., HALL, R., ZORN, B. *Programming with Visual Expressions*. In: Proceedings of Symposium on Visual Languages [online], 1995, pp. 294-301, doi: 10.1109/VL.1995.520822 [accessed 12.11.2022]. Link: https://ieeexplore.ieee.org/document/520822.
14. CORNELL University. *The Lambda Calculus*. In: CS 312 Recitations [online], ©2022 [accessed 12.11.2022]. Link: http://www.cs.cornell.edu/courses/cs312/2008sp/recitations/rec26.html.
15. COX, P. T., GILES, F. R., PIETRZYKOWSKI, T. *Prograph: a step towards liberating programming from textual conditioning*. In: [Proceedings] 1989 IEEE Workshop on Visual Languages [online], 1989, pp. 150-156 [accessed 21.12.2022]. Link: https://ieeexplore.ieee.org/document/77057.
16. DANYLIUK, I. *Rethinking Visual Programming (with Go)*. In: GopherCon Europe 2019 [online], ©2019 [accessed 21.12.2022]. Link: GopherCon Europe 2019: Ivan Daniluk - Rethinking Visual Programming.
17. DWECK, C. S., and BLACKWELL, L. S. *You Can Grow Your Intelligence*. In: Brainology Curriculum Guide for Teachers [online], Mindset Works, ©2014 [accessed 20.12.2022]. Link: https://www.kyrene.org/cms/lib/AZ01001083/Centricity/Domain/1891/you%20can%20grow%20your%20intelligence.pdf.
18. GRAMS, C. *How Much Time Do Developers Spend Actually Writing Code?* In: Tidelift [online], The New Stack, 15.10.2019 [accessed 10.11.2022]. https://blog.tidelift.com/how-much-time-do-developers-spend-actually-writing-code.
19. GUZDIAL, M. *Why should non-CS majors learn functional programming?* In: Computing Education Research Blog [online], 31.07.2015 [accessed 15.10.2022]. Link: https://computinged.wordpress.com/2015/07/31/why-should-non-cs-majors-learn-functional-programming/.
20. GÎRBA, T. *Moldable development.* In: Curry On! London [online], 2019 [accessed 21.11.2022]. Link: Tudor Gîrba - Moldable development.

21. HUIZINGA, J. *Homo Ludens*, Progress, Moscow, 1992, pp. 21-45.
22. IVANOVA, A., SRIKANT, S., SUEOKA, Y., KEAN, H. H., DHAMALA, R., O'REILLY, U.-M., BERS, M. U., and FEDORENKO, E. *Comprehension of computer code relies primarily on domain-general executive brain regions*. In: eLife 9 [online], 2020 [accessed 22.12.2022]. Link: https://elifesciences.org/articles/58906.
23. KAY, A. C. *The Early History of Smalltalk*. In: History of Programming Languages II [online], Association for Computing Machinery, January 1996, pp. 511-598 [accessed 21.12.2022]. Link: https://dl.acm.org/doi/10.1145/234286.1057828.
24. KEENAN, D. C. *To Dissect a Mockingbird: A Graphical Notation for the Lambda Calculus with Animated Reduction*. Dave Keenan's Home Page [online], 27.08.1996 [accessed 21.12.2022]. Link: https://dkeenan.com/Lambda/.
25. KELSO, J. K. *A Visual Representation for Functional Programs*. Murdoch University [online], December 1994. Link: https://www.researchgate.net/publication/2816386_A_Visual_Representation_for_Functional_Programs.
26. KORNILOV, Y., and VLADIMIROV, I. *Instrumental experience as a component of the experience of practical change*. In: Yaroslav Psychology Herald [online], ed. 16, RPO, Moscow, Yaroslavl, 2005, pp. 21-28 [accessed 20.11.2022]. Link: http://cafedra.narod.ru/kornilov_vladimirov_instropyt.pdf.
27. LAVRENOV, A. *What's Konva?* In: Starting with Konva, Konva Documentation [online], ©2022 [accessed 12.12.2022]. Link: https://konvajs.org/docs/index.html.
28. LEGRAND, B. *Most Symbols Have a Double Meaning*. In: Mastering Dyalog APL: A Complete Introduction to Dyalog APL [online], Dyalog Limited, ©2009 [accessed 12.11.2022]. Link: https://mastering.dyalog.com/README.html.
29. LINIETSKY, J. *Judging from all the feedback from the recent days regarding the Visual Script removal in Godot [...]*. In: @reduzio, Twitter [online], ©2022 [accessed 12.11.2022]. Link: https://twitter.com/reduzio/status/1562703467784728578.
30. LISZKOWSKI, U. *Three Lines in the Emergence of Prelinguistic Communication and Social Cognition*. In: Journal of Cognitive Education and Psychology, vol. 10, no. 1 [online], Springer Publishing Company, 2011, pp. 32-43 [accessed 22.12.2022]. Link: https://psycnet.apa.org/record/2011-06676-003.
31. LIU, Y.-F., KIM, J., WILSON, C., and BEDNY, M. *Computer code comprehension shares neural resources with formal logical inference in the fronto-parietal network*. In: eLife [online], 15.12.2020 [accessed 12.12.2022]. Link: https://elifesciences.org/articles/59340.

32. LOGO Foundation. *A Logo Primer*. In: What Is Logo [online], The Logo Summer Institutes Archive, ©2022 [accessed 12.12.2022]. Link: https://el.media.mit.edu/logo-foundation/what_is_logo/logo_primer.html.
33. MASSALOGIN, V. *Visual Lambda Calculus*. In: Master's thesis [online], Institute of Computer Science, University of Tartu, ©2008 [accessed 12.12.2022]. Link: https://bntr.planet.ee/lambda/work/visual_lambda.pdf.
34. MAYER, R., and FIORELLA, L. *Principles for reducing extraneous processing in multimedia learning: Coherence, signaling, redundancy, spatial contiguity, and temporal contiguity principles*. In: Cambridge Handbook of Multimedia Learning [online], pp.279-315, 2014 [accessed 11.11.2022]. Link: https://psycnet.apa.org/record/2015-00153-012.
35. MCCRUDDEN, M., and RAPP, D. N. *How Visual Displays Affect Cognitive Processing*. In: Educational Psychology Review 29, 2017, pp. 623-639 [accessed 11.12.2022]. Link: https://link.springer.com/article/10.1007/s10648-015-9342-2.
36. MINSKY, M., PAPERT, S. *Progress Report on Artificial Intelligence*. In: Project Mac Progress Report VIII [online], AI Laboratory, MIT, 11.12.1971, pp. 219-224 [accessed 12.12.2022]. Link: https://web.media.mit.edu/~minsky/papers/PR1971.html.
37. OBY, E. R., GOLUB, M. D., HENNIG, J. A., DEGENHART, A. D., TYLER-KABARA, E. C., YU, B. M., CHASE, S. M., and BATISTA, A. P. *New neural activity patterns emerge with long-term learning*. In: PNAS [online], ©2019 [accessed 11.11.2022]. Link: https://www.pnas.org/doi/10.1073/pnas.1820296116.
38. OLSSON, M. , and MOZELIUS, P. *Learning to Program by Playing Learning Games*, European Conference on Games Based Learning, vol. 11 [online], 2017, pp. 498-506 [accessed 12.12.2022]. Link: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1147746&dswid=5497.
39. PETERSEN, C. I., BAEPLER, P., BEITZ, A. J., and WALKER, J. *The Tyranny of Content: "Content Coverage" as a Barrier to Evidence-Based Teaching Approaches and Ways to Overcome It*. In: CBE life sciences education [online], vol. 19(2), 15.05.2020 [accessed 10.10.2022]. Link: https://www.lifescied.org/doi/10.1187/cbe.19-04-0079.
40. RAY, B., POSNETT, D., FILKOV, V., DEVANBU, P. *A Large Scale Study of Programming Languages and Code Quality in Github*. In: International Symposium of Foundations of Software Engineering [online], November 2014, pp. 155-165 [accessed 12.11.2022]. Link: https://web.cs.ucdavis.edu/~filkov/papers/lang_github.pdf.
41. RAPPIN, N. *Prograph*. In: Noel Rappin Writes Here [online], 26.11.2018 [accessed 12.10.2022]. Link: https://noelrappin.com/blog/2018/11/prograph/.

42. REPENNING, A. *Computational Thinking [does not equal] Programming*. In: Swiss Informatics Society Digital Magazine [online], 15.02.2017 [accessed 14.12.2022]. Link: https://magazine.swissinformatics.org/en/computational-thinking/.
43. ROWLAND, T. *Church-Turing Thesis*, In: MathWorld – A Wolfram Web Resource [online], ©2022 [accessed 11.11.2022]. Link: https://mathworld.wolfram.com/Church-TuringThesis.html.
44. RUBTSOV, C. A., ROMERIO, G. F. *Ackermann's function and new arithmetical operations*. In: NKS [online], 15.09.2004 [accessed 21.11.2022]. Link: http://www.rotarysaluzzo.it/Z_Vecchio_Sito/filePDF/Iperoperazioni%20(1).pdf.
45. SHIBLI, D., and WEST, R. *Cognitive load theory and its application in the classroom*. In: Impact Journal of the Chartered College of Teaching [online], Making Learning Stick: Open Access Cognitive Science, 22.02.2018 [accessed 12.12.2022]. Link: https://my.chartered.college/impact_article/cognitive-load-theory-and-its-application-in-the-classroom/.
46. SIEGMUND, J., KASTNER, C., APEL, S., PARNIN, C., BETHMANN, A., LEICH, T., SAAKE, G., and BRECHMANN, A. *Understanding Understanding Source Code with Functional Magnetic Resonance Imaging*. In: ICSE 2014: Proceedings of the 36th International Conference on Software Engineering [online], May 2014, pp. 378-389 [accessed 22.11.2022]. Link: https://dl.acm.org/doi/10.1145/2568225.2568252.
47. SOLOWAY, E., SPOHRER, J. C. *Studying the Novice Programmer* [online], Lawrence Erlbaum Associates, Inc., New Jersey, 1989, 504 p, ISBN: 9781315808321 [accessed 11.11.2022]. Link: https://www.taylorfrancis.com/books/mono/10.4324/9781315808321/studying-novice-programmer-soloway-spohrer.
48. SQUEAK Wiki. *The History of Smalltalk and Squeak* [online], ©2018 [accessed 22.12.2022]. Link: http://wiki.squeak.org/squeak/3139.
49. STANFORD Encyclopedia of Philosophy. *Turing Machines* [online], 24.09.2018 [accessed 12.11.2022]. Link: https://plato.stanford.edu/entries/turing-machine/.
50. STANFORD Encyclopedia of Philosophy. *Recursive Functions* [online], 23.04.2020 [accessed 12.11.2022]. Link: https://plato.stanford.edu/entries/recursive-functions/.
51. STEPANOV, A. *Generic algorithms*. In: Notes on programming [online], ©2018, p. 51, p. 78 [accessed 10.10.2022]. Link: http://stepanovpapers.com/notes.pdf.
52. STOVER, C., WEISSTEIN, E. W. *Function*. In: MathWorld – A Wolfram Web Resource [online], ©2022 [accessed 11.11.2022]. Link: https://mathworld.wolfram.com/Function.html.

53. STUART, T. *Programming with Nothing*. In: Ru3y Manor Conference [online], ©2011, Link: https://tomstu.art/programming-with-nothing.
54. SUTHERLAND, I. E. *Sketchpad*: *A man-machine graphical communication system*, PhD thesis [online], University of Cambridge Computer Laboratory, 1963, ISSN: 1476-2986 [accessed 12.12.2022]. Link: http://wexler.free.fr/library/files/sutherland%20(1963)%20sketchpad.%20a%20man-machine%20graphical%20communication%20system.pdf.
55. THEES, M., KAPP, S., STRZYS, M. P., LUKIWICZ, P., KUHN, J., and BEIL, F. *Effects of augmented reality on learning and cognitive load in university physics laboratory courses*. In: Computers in Human Behavior [online], vol. 108, 2020, ISSN: 0747-5632 [accessed 12.11.2022]. Link: https://www.sciencedirect.com/science/article/pii/S0747563220300704.
56. TORO, E. *Teaching Functional Programming*: *Two Big Picture Approaches*, DEV Community [online], 06.11.2018 [accessed 11.11.2022]. Link: https://dev.to/eddroid/teaching-functional-programming-two-big-picture-approaches-3nli.
57. TROMP, J. *Lambda Diagrams*. In: John's Lambda Calculus and Combinatory Logic Playground [online], ©2020 [accessed 11.11.2022]. Link: http://tromp.github.io/cl/cl.html.
58. TROMP, J. *Functional Bits*: *Lambda Calculus based Algorithmic Information Theory.* In: John's Lambda Calculus and Combinatory Logic Playground [online], ©12.01.2022 [accessed 11.12.2022]. Link: https://tromp.github.io/cl/LC.pdf.
59. UNREAL Engine 5 Documentation. *Blueprint Editors and Graphs*. In: Introduction to Blueprints [online], ©2022 [accessed 12.12.2022]. Link: https://docs.unrealengine.com/5.0/en-US/introduction-to-blueprints-visual-scripting-in-unreal-engine/.
60. VERNON, M. D. *Cognitive Inference in Perceptual Activity*, British Journal of Psychology [online], vol. 48, no. 1, February 1957, pp. 35-47 [accessed 19.11.2022]. Link: https://bpspsychub.onlinelibrary.wiley.com/doi/abs/10.1111/j.2044-8295.1957.tb00597.x.
61. VILLAREALE, J., BIEMER, C. F., EL-NASR, M. S., and ZHU, J. *Reflection in Game-Based Learning*: *A Survey of Programming Games*. In: FDG 2020: International Conference on the Foundations of Digital Games [online], September 2020 [accessed 10.09.2022]. Link: https://dl.acm.org/doi/fullHtml/10.1145/3402942.3403011.
62. VYGOTSKY, L. *Thinking and Speech*. State Socio-economic Publishing, Moscow, Leningrad, 1934, p. 88.

63. WEISSTEIN, E. W. *Quaternion*. In: MathWorld – A Wolfram Web Resource [online], 2022 [accessed 11.10.2022]. Link: https://mathworld.wolfram.com/Quaternion.html.
64. WING, J. M. *Computational Thinking*. In: Communications of the ACM [online], vol. 49, no. 3, 2006, pp. 33-35 [accessed 11.08.2022]. Link: https://www.cs.cmu.edu/afs/cs/Web/People/15110-s13/Wing06-ct.pdf.
65. WING, J. M. *Computational Thinking Benefits Society*. In: Social Issues in Computing [online], New York Academic Press, 10.01.2014 [accessed 11.08.2022]. Link: http://socialissues.cs.toronto.edu/index.html%3Fp=279.html.