BugHunt – A Motivating Approach to Self-Directed Problem-solving in Operating Systems

Simone Opel, Matthias Kramer, Michael Trommen, Florian Pottbäcker, Youssef Ilaghef
University of Duisburg-Essen
D-45127 Essen
{simone.opel, matthias.kramer, michael.trommen, florian.pottbaecker, youssef.ilaghef}@uni-due.de

Abstract: Competencies related to operating systems and computer security are usually taught systematically. In this paper we present a different approach, in which students have to remove virus-like behaviour on their respective computers, which has been induced by software developed for this purpose. They have to develop appropriate problem-solving strategies and thereby explore essential elements of the operating system. The approach was implemented exemplarily in two computer science courses at a regional general upper secondary school and showed great motivation and interest in the participating students.

Keywords: Educational software, operating system, student activation, problem-solving, interactive course, interactive workshop, edutainment, secondary computer science education

1 Introduction

The effective and efficient use of modern digital technologies has become a key competency in today's society (OECD, 2005), for both the private and the professional sector. In 2013, the Educational Testing Service (ETS) published an analysis (Burrus et al., 2013) on the most important 21st century workforce competencies by comparing three international 21st century skill frameworks, namely ATC21S, Finegold & Notabartolo and P21. A key component identified in view of the working requirements is "information processing" with the leading variable "computers and electronics". Students are being introduced to

the underlying principles of such electronic systems in the context of computer science education. This should enable them to use this technology in a qualified way and further developing it in the future.

In Germany, recommendations for national educational standards for lower secondary computer science (Brinda et al., 2009) were developed by a task force of the German Informatics association (GI), and most current German computer science school curricula published since then are oriented on these recommendations. In these educational standards, informatics systems are established as one of five school-relevant content areas. Therefore, students of all ages are to understand the basics of the structure of informatics systems and their underlying working principles with the aim to apply these systems in an effective and efficient way and to be able to learn and to understand further systems. The process of identifying school-relevant topics focuses on long-lasting computer science concepts and principles (Schwill, 1994), particularly as specific products as key aspect have also been criticised, because products evolve and only little transferable knowledge can be developed from them. Nevertheless, an essential aspect in the educational process is to link such product knowledge with conceptual knowledge (Hartmann et al., 2006).

Although more and more people use mobile devices such as tablets, traditional computers and laptops are still of great importance. Their qualified usage requires conceptual and product knowledge of the software needed (e.g. to solve a given problem) and of the underlying operating system software. This is also supported by international curricula such as the ACM K12-Curriculum (Tucker et al., 2003), or the IFIP Curriculum (van Weert, Tinsley, 2000), which list knowledge on operating systems and file management as their desired outcomes.

There are numerous systematic approaches to teach the use of software in full width (such as courses preparing for the European Computer Driving License (ECDL), (ECDL, 2013a, 2013b)). However, such offers are rarely made for secondary school students and also do not focus on presenting content in a motivating and explorative way.

At a project course for student teachers of computer science at the University of Duisburg-Essen, Germany, during the summer term of 2013, the students (split in small groups) designed motivating computer science learning units for secondary school students. One of these teams (the last three authors of this paper) developed the idea of using special “educational viruses” (“bugs”) to stimulate students to explore the underlying operating system. Students should deal with undesired system behaviour produced by these bugs and solve the arisen problems. In this paper, we (course tutors and students)

present the educational concept, the design and implementation of the learning software for the "BugHunt" project as well as first results and experiences with its use in two computer science classes at a regional general secondary school.

2 Related Work

To find ideas on how to explore an operating system combined with the fostering of problem-solving skills, existing approaches in this field were analysed. It soon became clear that ideas for this approach were very rare, if not unique. Although universities teach courses in virus programming (e.g. Aycock, 2013), it does not seem probable that they are used with educative intention, least of all in secondary education.

In a first step, international curricula were reviewed. According to the ACM curriculum (Tucker et al., 2003), students should gain a conceptual understanding of principles of computer organisation and its major components, such as storage or the operating system. The "IFIP-Unesco: ICT Curriculum for Secondary Schools" (van Weert, Tinsley, 2000), appendix A1, contains the demand that "students should understand how computers and the basic operating system work and demonstrate that the computer is under their control". Appendix A8 demands "[that] students are expected to understand basic concepts such as […] computer security (theft, hacking, and viruses)". Another modularised ICT curriculum and course offer is the "European Computer Driving License" (ECDL), which contains learning modules on "computer essentials" (DLGI, 2013a) and "IT-Security" (DLGI, 2013b). Specific educational objectives are being listed, such as "understanding how to use an operating system to organize drives, folders and files in a hierarchical structure" or "knowing how malware can be hidden in the system".

Based on these overall objectives, in the second step it was necessary to theoretically establish the competencies needed to successfully complete the planned learning unit in a second step. Within the framework of the MoKoM project (Linck et al., 2013), a competence model for informatics modelling and system comprehension was theoretically derived and empirically refined. The final model consisted of the five dimensions system application (K1), system comprehension (K2), system development (K3), dealing with system complexity (K4) and non-cognitive skills (K5), with several competencies subsumed under each dimension. In detail, the model provides competence goals such as "systematically explore system functions (K1.2.1)", "independently explore systems (K2.3)" or "know & analyse architecture & organization (K2.5)" to which the educational concept of the BugHunt project intends to contribute.

In a third step, reports on existing approaches were reviewed. Tulodziecki (2000) wrote on how computer-based media can be used to teach not only computer science, but all kinds of subjects. He stated that exercises were more effective when having a personal relevance for the students (such as occurring in their everyday life), and when the problem was on the one hand not solvable with the current knowledge, but on the other hand not too sophisticated. Westram (2006) described the importance of the subject "IT security" and demanded that it should be compulsory in secondary education. In her opinion, questions like what viruses, worms or Trojan horses are and how to deal with them is something computer science education has to convey. To promote these goals, teachers have to be provided with the necessary teaching materials. Schlüter (2006) gave an example of how a unit on internet risks could be structured. She proposed that in a first lesson the students should have a look on the topic "computer viruses". As the unit progresses, the students reflect on the behaviour and risks of viruses as well as an appropriate behaviour in case of a virus attack. This is just one possibility for students to increase their awareness on dealing with a malware situation in general.

In summary, it can be stated that operating systems and security aspects as well as the understanding of the underlying computer science concepts and principles are relevant educational goals to which teaching approaches have been described, but not in the integrated way suggested in this work.

3 Requirement Analysis

Starting point of the development was the idea to create a motivating learning unit in which students can explore selected aspects of different versions of the operating system Microsoft Windows in a short period of time. The basic idea was to place special "educational viruses" ("bugs"), which cause undesired system behaviour, as a motivating element in a protected environment on the computers. To remove the undesired system behaviour, the students should explore the settings of the operating system on their own and with the help of a particular text book. Due to possible safety concerns in school or university networks, the operating system should be run in a virtual machine, although the software leaves no undesired system behaviour in the operating system after its termination and should not need any network or internet access.

The so-called "BugHunt" learning unit should be designed for group sizes of up to 30 students (working in pairs) and a duration of 90 minutes. The course should be implementable both in traditional teaching at a secondary school as well as in the context of university courses, so the software should include dif-

ferent levels of difficulty. In order to create real experience, it is necessary to provide the students not only with an operating system simulation, but to bring the undesired, virus-like system behaviour (such as slowing down the system or executing undesired functions) to the real system. Each student team should be provided with appropriate tasks ("bugs") and have to try to deal with them as independently as possible. This requires a variety of bugs in varying degrees of difficulty to support an internal differentiation of the learning group. Moreover, the undesired system behaviour should be reflected in clearly recognisable symptoms that can be identified easily and directly. Furthermore, an additional teaching text should be provided in which the symptoms of the induced undesired system behaviour are linked with hints to an appropriate problem-solving strategy. To ensure that the students have to remove the bugs following the given instructions, the software must be secured against unauthorized termination. For this reason, there must not be any loopholes or workarounds the students could use to terminate the bugs without solving the actual problem. Therefore, the teacher must be able to start the controlling software, configure the level of difficulty and the quantity of bugs and to terminate it separately from the actual bug programme. By processing the tasks, the students should build up or enhance computer science competencies as they have been described for example in the MoKoM project (Linck et al., 2013). For example, the students should systematically explore system functions (K1.2.1) and know and analyse architecture and organization (K2.5) of an operating system. They should learn by exploring the operating system independently (K2.3) and not by being taught the solutions and techniques in traditional teacher-oriented lessons. Being confronted with a series of problems on a running operating system, they should know about and evaluate consequences of informatics systems (K5.1.1.5) and finding solutions to the arisen problems in a self-directed way, we hope to increase their affinity and enthusiasm (K5.1.2.1) and make them willing to improve their informatics abilities and knowledge (K5.3.2.1). In order to prevent the students from copying the solutions from other teams, the sequence of released bugs must be randomized.

To minimize the preparation time of such a course, the teacher or university tutor should only have to prepare the computers by activating the software on each computer without the necessity of an installation process. During the course, the teacher should be able to configure the difficulty level and the selection of bugs provided to a student team. After showing the students how to use the software and the teaching material, the teacher should not need to give any further assistance except for answering questions or solving general technical problems.

Finally, the software should be executable on any computer or laptop with one of the operating systems Windows 8, 7, Vista, XP, 2000 installed and not require any special resources. It should be developed modularly for an easy extension by more or improved bugs.

4 Design and Implementation

According to the requirements, the software should be set up with minimal effort and without an installation process or any necessary runtime environment. At first, we had to decide which language should be used to implement the software. We decided to use AutoIt, a Basic like programming language, which provides access to the Microsoft Windows API by using C++ syntax in dll-calls (Aristides de Fez Laso, 2013; Petzold, 2013). Using this technology enables easy access to every input device and the desktop environment. Thus it was possible to develop bugs, which would be able to take control of the mouse cursor, the keyboard, the file system and the most common system functions. To secure the software against manipulation by students, it is developed to run in only one hidden process with a single thread. Other advantages of this solution are that the software needs just a small amount of system resources and that its tasks can hardly be recognised.

The Software is split into three parts: The BugHuntMaster, the BugHunt main process and the BugHunt recovery process (cf. Fig.1).

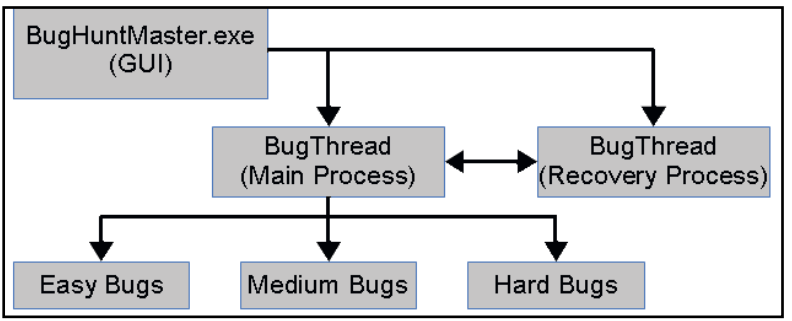


Figure 1: Sketch of the BugHunt architecture

The BugHuntMaster.exe runs portable from the BugHunt USB device. It ensures the teacher's control over the bugs. Each bug is programmed in a separate file, which contains the polled function of the bugs and its global variables.

The simple GUI (cf. Fig. 2) is built by an algorithm, which scans for implemented bugs during compilation, and is configured dynamically.

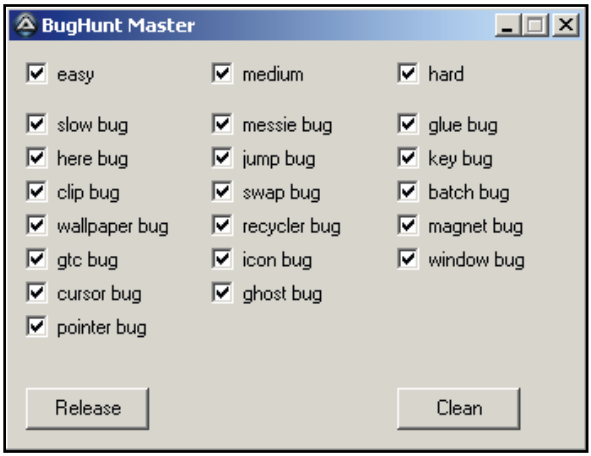


Figure 2: BugHuntMaster.exe GUI

The desired bugs can be activated individually or as a whole difficulty group by checking or unchecking the box on top of the column.

The main process runs the bug functions, which are arranged in three levels of difficulty. The bugs are not separate applications, but different functions in the main process. Despite this architecture, more than one bug can run simultaneously. The BugHunt main process executable is copied to the system by the BugHuntMaster.exe. That main process executable contains the bug functions along with a random bug activation algorithm according to the written configuration. Only one bug of each difficulty level can be active at the same time to prevent an unsolvable scenario. The main process simulates the symptoms and decides whether a bug has been solved or is still active. Furthermore, it alerts the student in case of success. The main process also ensures that the recovery process is running.

The recovery process monitors the main process in a 10ms interval and keeps it running. Moreover, it restores the BugHunt system files which may be deleted due to manipulation or system failure. If the main process is being terminated, the recovery process will start it again. In case of missing BugHunt files, it automatically restores them from backup files. This virus-like behaviour is to challenge the students to systematically explore possible problem-solving approaches without giving them the opportunity to simply stop the task and delete the BugHunt programme.

Despite those security features, the teacher should be able to stop all bugs immediately at any given time. Both BugHunt processes scan for the BugHunt USB device at the beginning of every iteration. If the device is being detected, both loops will terminate immediately. So the teacher can easily start the BugHuntMaster again and change the configuration or stop the programme.

5 The Bughunt Software

The teacher has to connect the BugHunt USB device to a computer. The BugHunt-Master.exe, which has to be executed, is located in the root directory of the BugHunt device. When the "RELEASE" button (cf. Fig. 2) is clicked, the BugHuntMaster.exe copies all BugHunt files to a hidden directory and creates two hidden backup copies of all BugHunt files at different locations. The BugHuntMaster.exe resides on the USB device and is never copied to the system. Thus, it is unreachable after the device has been removed. After the BugHunt configuration is written to the system, a message box will appear asking to disconnect the BugHunt USB device and click "OK". When the device has been disconnected the BugHunt main and the BugHunt recovery process (cf. Fig. 1) will be invoked after a short time of 10 seconds.

The "CLEAN" button stops each running BugHunt action. Additionally, it deletes all files from the system and resets all changes, which have been done. The software is fully removed in less than one second by only one click.

Exemplary "Bugs"

Subsequently, we describe the induced behaviour of selected bugs and the learning objectives to be achieved by their removal.

The *GTC Bug* (General Terms and Conditions Bug, category "easy") shows a dialog box, which asks the student to click "YES" to remove unwanted software (apparently the bug itself). There is an obligatory checkbox at the bottom of this dialog box, which is already checked to accept the terms and conditions. This common situation postulates the student to know and evaluate consequences of informatics systems (K5.1.1.5). If the student just clicks "YES" without at least having a quick view at the information, a random number of folders named "WASHING MACHINE MODEL_XYZ" will be created on the desktop. The dialog box will be shown again after a few seconds. If the student decides to read the information, he will notice that he just has to uncheck the terms and conditions box to remove the bug. Accepting all given terms without reading the text itself is a common but careless behaviour for most computer

users – not only for school students. This bug should encourage the students to reconsider their own behaviour and to learn how to act in a reasonable way.

The *Slow Bug* (category "easy") slows down the whole system. It is designed to simulate the well-known effect that hidden processes consume lots of resources without being identified as the root of the problem. To ensure that the bug is being noticed even on very powerful systems, the mouse cursor movement judders due to cursor manipulation by this bug. The student has to systematically explore system functions (K1.2.1) to find and open the task manager. All CPUs will show a usage of 100 %. The bug is removed by closing all "Slow Bug" processes, which are in the list.

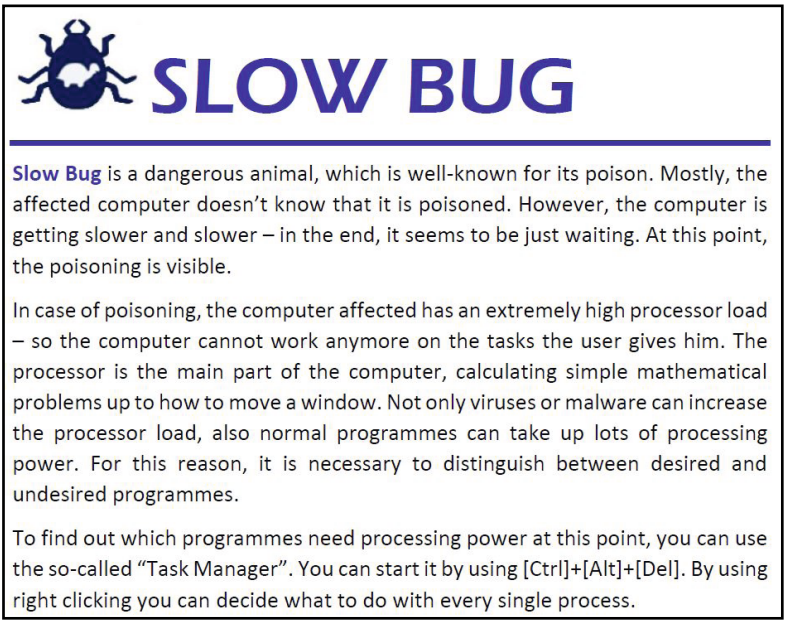


Figure 3: Bug description in the textbook, category "easy"

The *Swap Bug* (category "medium") makes the mouse cursor move in a random interval between 15 and 30 seconds. The movement of the cursor is not random; it always follows the same path. The student has to independently explore the system (K2.3) to find suitable graphic editing software. The path can be made visible by clicking and holding the mouse in a drawing area. The cursor will draw the letters "Ctrl B U G". The student has to identify the output as hotkey combination. The written capital letters must be identified as a combination of upper case characters. Hence, the actual hotkeys are "ctrl +

shift + character". The bug will be solved when the three hotkey combinations are pressed in order of appearance. Since the students can hardly control the mouse cursor, they have to use the keyboard to find and open the graphic software. Although all modern computers have mice or other graphical input devices, students become acquainted with an efficient way to use their computer by removing this bug.

The *Key Bug* (category "hard") encrypts the keyboard with a random Caesar cipher. Encryption in general is a fundamental concept in computer science and computer security, which is modelled by this bug. A text document called "Message from Key Bug.txt" will appear on the desktop. It contains an encrypted text, which in its decrypted version says "Who should I be afraid of?". The correct answer ("julius caesar" or "alan turing") has to be written to the file with the keyboard still being encrypted. The "bug identification textbook" contains some basic information about the bug and the idea of the Caesar cipher. The student has to identify the decryption and write the correct answer into the file to remove the bug. That means that the student has to understand and use decryption as well as encryption to solve the problem.

6 Implementation and Evaluation of the Learning Unit

6.1 Implementation of the learning unit

The course was carried out twice with 10th grade students at ages between 14 and 17 years of two elective computer science classes of the same general upper secondary school ("Gymnasium") in a computer room supplied with personal computers and laptops. The students were divided into pairs with one pc or laptop per group. The pcs and laptops were prepared by creating a new user account on the operating systems for the course. On these accounts, the "BugHunt" software was started and easy bugs were chosen for every group to begin with. At the beginning of the course, an additional teaching text (the "bug identification textbook", cf. Fig. 3) was given to each group. After that, the students got a short introduction on how to use the book. We explained the different levels of difficulty and the general handling of the software. Then the students were asked to solve the problems caused by the bugs. They should first try to identify the type of the active bug and after that they should remove it, using the information given in the textbook. Afterwards, the students started to work. Since some questions were asked repeatedly, additional hints were written on the board. After about half of the time, several groups finished all bugs from the easy category and started with the medium bugs. Other groups

however had difficulties with different bugs from the easy category, so they removed only these easy bugs.

6.2 Evaluation

Since a main component of the concept was to motivate the students and to interest them in operating system issues, the participating students were interviewed using a questionnaire after the implementation of the learning unit. For this purpose, we developed a questionnaire consisting of three parts to explore the motivation and interest of the students in partaking in this course.

In the first part we asked for personal information such as age and sex as well as experience in computer science lessons and career aspirations of the students.

The second part consisted of questions about their attitudes towards the course (six items) and their personal interests (four items). Exemplary items are "the course was exciting and interesting" or "I am interested in solving difficult problems". We used a 4-point scale answering format with the options "yes" (4), "generally yes" (3), "generally no" (2) and "no" (1). The items are not standardised. For this reason, the internal consistency of the questionnaire about attitudes is only $\alpha = .641$ and about interest $\alpha = .800$.

In the third part, we used the questionnaire on subjectively perceived boredom in mathematics at elementary schools by Sparfeldt et al. (2009). Todt (1990) describes boredom as the opposite of interest (Todt, 1990), beyond that a highly negative correlation between interest and boredom has been identified (Lohrmann, 2008; Pekrun et al., 1998). Whereas interest and school grades are highly correlated, no correlation has been found between boredom and school grades (e.g. Dickhäuser, Stiensmeier-Pelster, 2003; Todt, 2000). Thus, perceived boredom seems to be a reasonable predictor to evaluate the interest of the students in partaking in the developed course. This part of the questionnaire consisted of 14 items and used a 5-point scale format from "never" (1) to "always" (5). The original questionnaire had a reliability of $\alpha = .957$. We modified this questionnaire for the purposes of the BugHunt course ($\alpha = .959$) by replacing "mathematics" with "this course". Exemplary items are "In my opinion, the course was boring" or "During the course, I looked out of the window because I was bored".

6.3 Results

As reported before, we performed this learning unit in two elective computer science courses (10th grade) of the same general upper secondary school ("Gymnasium") with 20 students each. 38 of these students participated in the course. Two-sided t-tests showed that both courses originated from the same population. For that reason, both courses have been evaluated as one. The courses consisted of 30 boys and eight girls at ages between 14 and 17 (M = 15.47, *median* = 15). Only 16 students related that they had had elective computer science lessons in former school years.

The students reported a positive attitude towards the course (M = 3.265; SD = 0.413). In particular, the students related that they would like to have more similar courses, also on other topics from computer sciences. The students were mostly interested in computer science (M = 2.906; SD = 0.723).

Although the second part of the evaluation had a low internal consistency, it nevertheless confirmed the results of the questionnaire about the perceived boredom: The students reported only little boredom during the course (M = 1.705; SD = 0.845). These findings were also confirmed by several annotations on the evaluation sheets: "It was cool, completely different!", "The course was very interesting and informative. Surely, it would be possible to cover different problems, too" or "Very cool, it would be nice if you visited us again!". The most astonishing feedback was that one of the students encrypted his statement by using a Caesar cipher.

Furthermore, our study confirmed the findings of Pekrun & Hoffmann (1999) and Lohrmann (2008). We also found a negative correlation between perceived boredom and interest in computer science (r = -.567; $p < .001$) and between perceived boredom and attitudes towards the course (r = -.599; $p < .001$). This result was also confirmed by informal feedback and observation of the behaviour of the students during the course.

The previous knowledge of the students about operating systems was quite diverse. For this reason, the students began with very different approaches.

A few groups with advanced experience with the operating system tried to terminate the whole software by terminating the tasks in the task manager. Being unsuccessful, they attempted to figure out how the BugHunt software works by using all skills they had. After that, these groups started to solve the bugs the designated way. Generally, these groups approached all bugs by trying different strategies on their own and rarely by reading the "bug identification textbook".

Students with fewer previous skills mostly started with trial-and-error strategies to solve the problems caused by the bugs, but during the course they managed to develop more useful strategies, e.g. “first describe the problem, then read the identification textbook, after that start working by interpreting the given instructions”. Following these rules, they were able to remove at least all easy bugs.

While observing the students, we noticed increasing problem-solving strategies. Furthermore, some students found creative ways to solve the given problems. For instance, to fix one of the bugs, it is necessary to freeze the picture by making a screenshot. Several students did not know how to do this, so two of them used their smartphones to take a photo of the screen. Even though this was not the intended way to solve the problem, it was a very creative solution.

All things considered, most students seemed to work in a motivated and interested way on the different given problems. The groups were very focused during the course. There was much intra-group and cross-group communication. At the end of the time, most of the groups reached the medium category. There were only a few groups who started the bugs from the hard category and also those who did not finish the easy one.

7 Summary and Outlook

In this paper, we have described a differing approach to operating systems and other computer science concepts that is based on the hypothesis that removing virus-like behaviour on the computer is motivating for students. For this purpose, a special learning aid, the BugHunt software, was designed which induces the system behaviour required on the respective computer. The process of removing the bugs required individual problem-solving strategies and made the students explore various computer science concepts. A written survey after an exemplary implementation showed that the students worked with great interest and motivation on the tasks and would wish for more such concepts. Although these results are very encouraging, is it necessary to evaluate and improve this concept with more students of different age.

Of course, we are aware that the induction of virus-like behaviour on school computers can be discussed controversially with regard to safety. For this reason, we recommend to run the software in a virtual machine, although it is safe to run it on a real computer.

The approach presented here does not claim to be a systematic approach to operating systems concepts, but it can be used to support corresponding teaching sequences and to promote the motivation of the students. The software

has been designed in a way that the integration of other bugs is easily possible. More information on the BugHunt project can be found on the website http://udue.de/bughunt.

References

Aristides de Fez Laso, E. (2013). *Instant AutoIt Scripting*. Birmingham: Packt Publishing.

Aycock, John (2013). CPSC 527 – *Computer Viruses and Malware. University of Calgary, Department of Computer Science*. Retrieved February 10, 2014 from http://pages.cpsc.ucalgary.ca/~aycock/virus-info.html

Brinda, T., Puhlmann, H., Schulte, C. (2009). Bridging ICT and CS – Educational Standards for Computer Science in Lower Secondary Education. *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education (ITiCSE'09)*, 288–292.

Burrus, J., Jackson, T., Xi, N., Steinberg, J. (2013). *Identifying the Most Important 21st Century Workforce Competencies: An Analysis of the Occupational Information Network (O*NET)*. Educational Testing Service: Princeton.

Claus, V., Schwill, A. (2006). *Duden Informatik A–Z*. Mannheim: Dudenverlag.

Dickhäuser, O., Stiensmeier-Pelster, J. (2003). Wahrgenommene Lehrereinschätzungen und das Fähigkeitsselbstkonzept von Jungen und Mädchen in der Grundschule. *Psychologie in Erziehung und Unterricht, 50*, 182–190.

Dienstleistungsgesellschaft für Informatik – DLGI (2013a). *ECDL / ICDL Computer-Grundlagen*. Retrieved February 10, 2014 from http://www.ecdl.de/fileadmin/redaktion/Syllabi/ECDL_new/ECDL_Computer-Grundlagen_2013.pdf

Dienstleistungsgesellschaft für Informatik – DLGI (2013b). *ECDL / ICDL IT-Sicherheit.* Retrieved February 10, 2014 from http://www.ecdl.de/fileadmin/redaktion/Syllabi/ECDL_new/ECDL_IT-Sicherheit_2013.pdf

Hartmann, W., Näf, M., Reichert, R. (2006). *Informatikunterricht planen und durchführen*. Berlin: Springer.

Linck, B., Ohrndorf, L., Nelles, W., Neugebauer, J., Magenheim, J., Schaper, N., Schubert, S., Stechert, P. (2013). Competence model for informatics modelling and system comprehension. *Proceedings of the 4th global engineering education conference*, IEEE EDUCON 2013, 85–93.

Lohrmann, K. (2008). *Langeweile im Unterricht*. Münster: Waxmann.

Nussbaum, M., Infante, C. (2013). Guidelines for Educational Software Design That Consider the Interests and Needs of Teachers and Students. *Proceedings of the 13th International Conference on Advanced Learning Techniques, IEEE ICALT* 2013, 243–247.

OECD (2005). *The Definition and Selection of Key Competencies*. Retrieved February 10, 2014 from http://www.oecd.org/pisa/35070367.pdf

Pekrun, R. (1998). Schüleremotion und ihre Förderung: Ein blinder Fleck der Unterrichtsforschung. *Psychologie in Erziehung und Unterricht, 44*, 230–248.

Pekrun, R., Hoffmann, H. (1999). Lern- und Leistungsemotionen: Erste Befunde eines Forschungsprogramms. In Jerusalem, M., Pekrun, R. (Eds.), *Emotion, Motivation und Leistung* (pp. 247-268). Göttingen: Hogrefe.

Petzold, C. (2013). *Programming Windows: 6th Edition*. Redmond: Microsoft Press.

Schlüter, K. (2006). Gefahren im Internet. *LOG IN – Informatische Bildung in der Schule, 140*, 35–44.

Schwill, A. (1994). Fundamental ideas of computer science. *Bulletin of the European Association for Theoretical Computer Science, 53*, 274–295.

Sparfeldt, J., Buch, S., Schwarz, F., Jachmann, J., Rost, D. (2009). Rechnen ist langweilig – Langeweile in Mathematik bei Gundschülern (German). *Psychologie in Erziehung und Unterricht, 1*, 16–26.

Todt, E. (1990). Entwicklung des Interesses. In Hetzer, H., Todt, E., Seiffge-Krenke, I., Arbinger, R. (Eds.), *Angewandte Entwicklungspsychologie des Kindes- und Jugendalters (2. Aufl.)* (pp. 213–264), Heidelberg: Quelle & Mayer.

Todt, E. (2000). Geschlechtsspezifische Interessen – Entwicklungen und Möglichkeiten der Modifikation. Empirische Pädagogik, 14, 215–254.

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., Verno, A (2003). *A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee*. Retrieved February 10, 2014 from https://csta.acm.org/Curriculum/sub/CurrFiles/K-12ModelCurr2ndEd.pdf

Tulodziecki, G. (2000). Computerbasierte Medien. LOG IN – *Informatische Bildung in der Schule, 5/2000*, 8–12.

van Weert, T., Tinsley, D. (eds.) (2000). *Information and Communication Technology in Secondary Education – A Curriculum for Schools*. Paris: UNESCO.

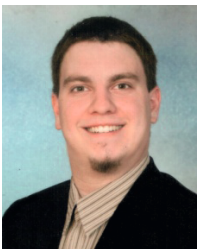
Westram, H. (2006): IT-Sicherheit im Unterricht. *LOG IN – Informatische Bildung in der Schule, 140*, 20–24.

Biographies

Simone Opel studied Information Technology at the University of Applied Sciences of Nuremberg and Vocational Education for Electrical Engineering and Computer Science at the University of Erlangen-Nuremberg. She worked as a trainer for computer science and teacher at several vocational schools. Since 2010, she is working as a scientist in the "Didactics of Informatics" groups at the Universities of Erlangen-Nuremberg (until Oct. 2012) and Duisburg-Essen (since Nov. 2012).

Matthias Kramer studied Computer Science and Mathematics for grammar schools at the Friedrich-Schiller-University in Jena. He completed his internship of teaching practice in January 2013. Since April 2013, he has been working as a research assistant at the chair of "Didactics of Informatics" at the University of Duisburg-Essen.

Michael Trommen took his O-Level at the Gymnasium Rheinkamp in 2001 and got his apprenticeship diploma as IT system technician in 2005. He took his high school diploma (Abitur) at the Abendgymnasium Duisburg in 2010. Since 2010 he is studying Computer Science and Mathematics for grammar schools at the University of Duisburg-Essen.



Florian Pottbäcker took his high school diploma (Abitur) at the Krupp-Gymnasium grammar school in Duisburg (in June 2009). Since October 2010 he is studying Computer Science and Mathematics for grammar schools at the University of Duisburg-Essen.

Youssef Jlaghef got his high school diploma (Abitur) in June 2006 at the Mercator Berufskolleg in Moers. He is studying Computer Science and Mathematics for grammar schools at the University of Duisburg-Essen since October 2010.