

Providing Help for Novices Using Expert Knowledge

Simon Bruns
RWTH Aachen University
Simon.Bruns@rwth-aachen.de

Arne Döring
RWTH Aachen University
Arne.Doering@rwth-aachen.de.com

ABSTRACT

In this paper we will guide you through some technologies that are useful in providing help for novices. It will introduce many technologies that are useful and some that are still in development and might be useful in the future. This paper is all about technology, it does not give information about how experts and novices should work together in a team. The technology we present can be separated into three approaches *Support*, *Recommendation* and *Simplification*. *Support* deals with problems the user encounters while the application is running. He is able to access a help system, which gives an appropriate answer to his problem. Additionally the application considers the users behavior and gives him tips, how to enhance his abilities, which is known as *Recommendation*. Besides the application based help, the user can learn from experts, how to use an application. This paper introduces some techniques of the presented approaches and states the problems in currently used realizations. *Simplification* is the last approach, which simplifies the process of learning programming. The novice learns basic structures from which he can advance to more complex programming languages.

AUTHOR KEYWORDS

Help Systems, Online Help, Support, Simplification, Recommender System, Design, Human-Factors

INTRODUCTION

Modern applications become more and more complex, thus many users are overwhelmed by them. Although these applications have help providing interfaces, they are often unable to solve the problem of the user. There are many approaches to deal with this complexity. We introduce three approaches. The application suggests the user a solution to his current problem, *Support*. It supplies him tips to improve his working process, *Recommendation*. A simplified version of the application is provided to the user, *Simplification*.

SUPPORT

A well known approach in providing help to novices is *support*. Beside its obvious use in computing, support is also present in many different areas. The origin of computer support has started with simple copies of papers. This has been the first approach of a computer help system. Afterwards these documents have been modified with references, which have improved the usage of them. Today *help systems* are more complex. They use their own database and have a unique structure. The collected information of the database is based on multiple sources. However, the behavior of modern users is not to use help systems [9]. They prefer to find an answer on their own, for example when they are confronted with a new applet, they look for similar problems in a board. Responsible for this attitude are certain flaws of current help systems. These flaws can be divided into three aspects.

The first aspect is the lack of appropriate help [9]. The given help is not based on the user's knowledge and does not consider his current situation. It is mostly based on answers for *generic problems*, which consist of generic information created by the designers. Hence the user needs a certain amount of background knowledge to understand these generic problems and their answers. The user has to refer his problem to the generic one. In case the user can identify his own problem, the given answer might also not be fully satisfying or understandable, especially for a novice. Furthermore these generic problems are often outdated, so the user cannot get reliable help for up-to-date issues.

The second aspect deals with design flaws [9]. Most help systems do not use a customized design to present their database. It is more common to have a simple interface like a help document. However most users find it rather difficult to define their problem or even to use the help system, because of its insufficient design.

The third aspect is about efficiency of solving a problem. Some problems cannot be solved or updated by computer assistance, a human expert is then necessary. But a human expert is more expensive and is limited to the amount of requests that he can process. As a result, the creator of a help system needs to find the right balance between human experts and computer assistance. Help systems are not able to define the actual problem of the user, which results into frustration for him and

unnecessary run time [10]. Additionally, they do not save the log of a support session, which could shorten the assistance process. The following help systems are using unique approaches to deal with these aspects.

FAQ

One of the first help systems is *Frequently Asked Questions* (FAQ) [4] which has been created on the ARPANET in 1982 [4]. Today it is well known in the Internet, but also in the Usenet it was already known. FAQ is based on common problem patterns that have been encountered by human assistance. In order to avoid repetitive questions, the human assistances write a list of them with their relevant answers. The created FAQ can be modified through observation of further problem encounters or immediate by the expert, if he thinks it should be added to the FAQ. The resulting FAQ consists of generic problems and is growing over time. FAQ is one of the oldest help systems and therefore it has some design flaws. For instance not every problem can be classified in a question-answer format. Another flaw is that FAQ in practice has no specific order, so the advantage of an evolving database works against the idea of a quick supply of relevant answers. This problem is caused by the simple structure of FAQ as it is a collection of questions, which has no apparent connection with each other. In spite of these disadvantages FAQ is still one of the more commonly used help systems, because of its easy to grasp structure and its reliability. There has been further development on FAQ to solve the design flaws of it. The resulting help system is called Answer Garden.

Answer Garden

Answer Garden is a concept from 1990, which solves the problems of FAQ. It is designed to be used in many different areas of application such as companies, programming and general support. Answer Garden is a self evolving FAQ database, which organizes the question flow of the users through categorizing the questions into groups. Answer Garden is maintained by three key ideas:

Structure:

Answer Garden [1] uses a tree structure to organize its database also called branch network. Each root consists of multiple leafs, which split the content range into subdivisions. In the process this structure is dynamic. Thereby popular problems are pulled up within the structure to guarantee a quick help supply. Furthermore experienced users can skip parts of it with a small model of the database (tree model) within the Answer Garden. Because of its dynamic structure, the Answer Garden is easy to modify or to add new content.

Evolving Database:

The database grows over time. Each user can inform the admins with a notification, if the path he has chosen has led him to a dead end. If the missing problem is valid to the network, the admin can add a new question branch

to it. Furthermore in case the user is not satisfied with the relevant answer or its position within the network, he can notify the admin and give an annotation, what is not acceptable. Hence the resulting database is up-to-date and consists only of expert knowledge.

Defining the Problem:

Answer Garden uses a tree structure to define the problem of the user. Each level of the tree represents a more detailed definition of the user's problem. This definition is realized through a repetitive question-answer process, which is an option panel with multiple answers to the questions of the database. Between each repetition, Answer Garden shows the user, what the current answer of the defined problem is. The user can choose, when he is satisfied with it.

The concept of Answer Garden solves most of the flaws of FAQ. It can be modified by the user to improve its utility and content. In the process the user can even change the structure of it, so popular problems can be found faster. But as there has been no detailed implementation of it, its use in practice is not evaluated.

CHIC

A general problem of classic help systems is the gap between user and designer. To reduce this gap the idea of "users help themselves" arouse [9], which is a fundamental idea of community based help systems. Community help systems benefits from a fast response to not foreseen problem's at the time of design and their growing collective memory. Normally these help systems have an unstructured collective memory; in this case the usage of a wiki system architecture is advisable. However community based help systems are hardly integrated into current applications. Integrating a help system (build-in one) increases the usability of the help system and allows a context-sensitive "one-click" help. *Community help in context* (CHIC) [9] is the result of combining these two approaches. The system architecture is divided into three generic modules:

AIM:

The Application Integration Module integrates CHIC into a desired application. It is responsible for the interaction of user and CHIC system. Furthermore it realizes a context-sensitive help with a "single-click" access to other help texts on the Internet or the net based CBHS-System.

CBHS:

A Community based help system can be a modified version of a traditional community with further functionality to provide a context-sensitive help.

CAM:

The Context-Aware Adaption Module is the interface between AIM and CBHS. It associates the existing query of the AIM with an appropriate entrance point within the CBHS.

AIM uses two modes to integrate information into the application. The first one is re-active mode, which modifies the interface of the application with a characterizing link of the help content. This link is manually set by the designer or is automatically set by the CHIC system, which considers runtime information of the application. The second one is pro-active mode. In this mode the help context dynamically changes in respect to the current state of the application (figure 1). The observation of the application is realized by using context identifiers for each state (1). These context identifiers are also used to associate help entries from the CBHS with the context (2). Each help entry is connected to its corresponding Wiki page inside the help system (3). In addition to the actual help special links are used to add new help entries to a given context.

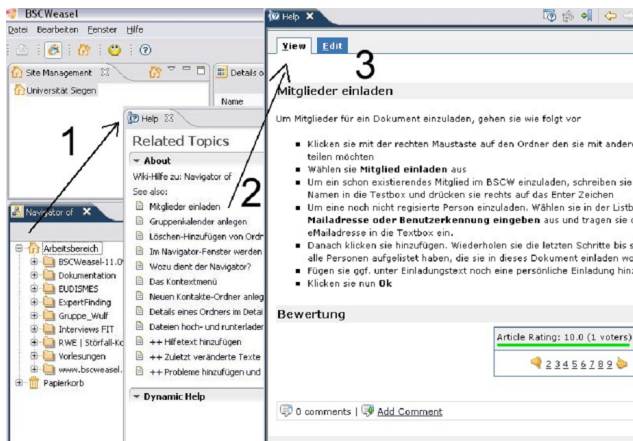


Figure 1. After accessing CHIC a list of context identifiers is presented (1). The user selects his current problem (2) and is presented with the corresponding wiki page (3). [9]

To ensure this the structure used for the CBHS is Atlasian Confluence Wiki. The strength of this wiki is a web service interface that allows retrieving and adapting the content via remote procedure calls. Furthermore, it has a hierarchical organization of the Wiki pages, which enables each page to have sub-pages. A context identifier is mapped as the name of a Wiki page. The corresponding help entries are mapped as its sub-pages. Besides these extensions, the user is able to add a comment or even a question to a community. In this case the participating users are notified about changes of a help page. The association of the help text and context is produced by integrating context identifiers to a particular state of the application. These context identifiers have to be unique within the system to distinct between help entries.

CHIC is a new help system concept, which cannot be compared to current ones. Because of its structure, it solves most issues of support. The only problem is the level of the community users, so it will be necessary to use experts to ensure the quality of each help page.

AIDE

As CHIC is a new approach to help systems, it is still based on the classical cognitive theory that knowledge can be taken out of context. So as long the information is generic enough for users to adapt to their particular situation, the help system is sufficient. In this way the amount of information required to support software is reduced. But if the user is not as experienced as the designer of the software and his actions cannot be foreseen by one, the computer assistant will be insufficient. The reason for this is that in the process of de-contextualization, to achieve a generic enough information, the so called local knowledge is lost. This knowledge describes the usage of it to its particular information. Hence the user is not able to adapt to the information unless he has a certain amount of knowledge [10].

In contrast to the classical cognitive theory stays the situated action theory, which states that knowledge can only be interpreted in its context. A help system based on this theory would keep the contextual information to a given concept to ensure an intuitive adaption by the user. To realize this theory you have to produce a very detailed description of the environment and actions that take place in it so its implementation is the main problem. Furthermore, it is difficult to build software that answers the needs of the user, unless the designer himself does it. Solving this problem is done by further analyses on human assistance as they have a situated process [10].

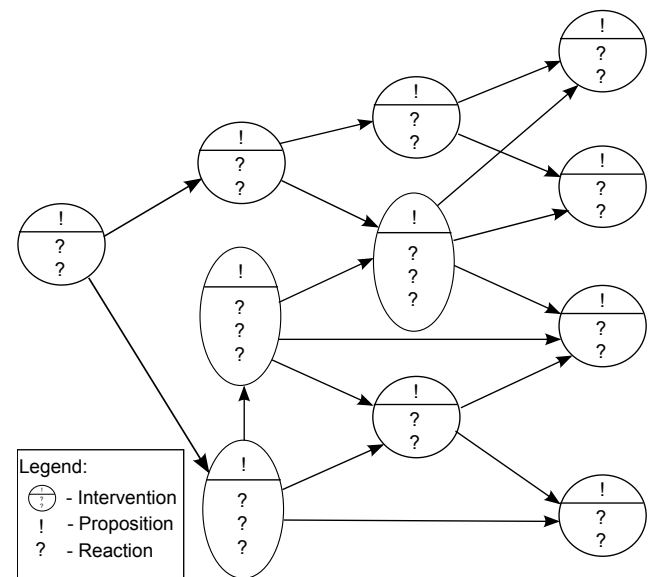


Figure 2. Each proposition is connected to multiple reactions. A chosen reaction is symbolized by a link, which leads to the next intervention. [10]

Human assistance classifies each problem into groups, which induce different kinds of requests. To define the problem, human assistance is a multiple-step process, in which the demander asks repeatedly questions as well

as the assistant asks follow-up questions to them. This way the definition of the problem is a co-constructed process, where both ends contribute to it. As this process is a learning one, the assistant gets better as he tackles new requests.

The realization of these concepts is *AIDE* [10]. The assistance is provided through a dialog between the user and the system. In the process the demander gradually describes his situation and his needs. The system follows-up with an intervention composed of a proposition and a series of reactions from which the demander can choose from to further describe his request. This dialog is repeated till the demander is satisfied with the proposition of the system. Figure 2 is an overview of this process. The circles are the interventions, the exclamation marks (!) are the propositions and the question marks (?) are the offered reactions of the system. The corresponding intervention to the selected request of the user is represented by a link (→) between them.

The course of *AIDE*'s assistance can be divided into three parts. At first it asks the user questions to determine the user's problem. After this it asks further questions to specify the kind of request. At last it tries to identify the specific needs of the user based on the selections before and make a proposition to which the user either reacts to further define his problem or accept it. Not every user can correctly identify his problem, therefore the system adds to the selectable reactions further ones, who are not directly connected to the defined problem so far. Furthermore the user is able to add new requests with the option "Suggestion", which would have improved the course of his definition process. These suggestions are monitored by the editor to ensure a correct database. In case an experienced user is working with *AIDE*, it is possible for him to skip parts of the dialog with a hierarchical overview of the database, figure 3.

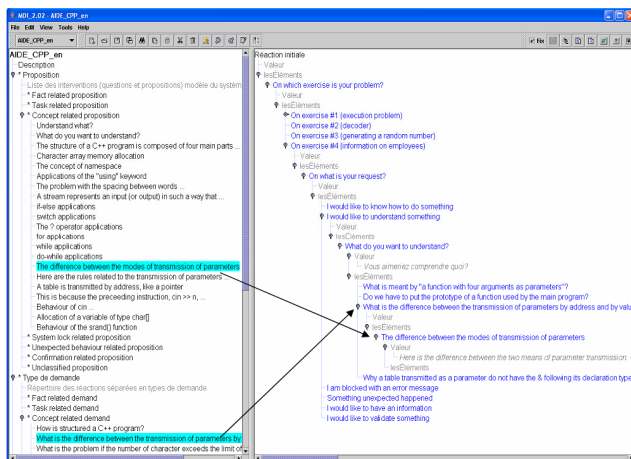


Figure 3. From the left frame the user can access a deeper level of the tree structure with his desired information. [10]

As *AIDE* is one of the first help systems, which tries to mimic a human assistance, its design benefits greatly the support idea. However the database behind *AIDE* is not growing per request so the issue of up to date help is not fixed. And it has to be administrated by an experienced user to ensure an appropriate result.

Sikuli Search

Another approach, which benefits from analyzing human behavior, is *Sikuli Search* [11]. In a human-to-human communication, the description of an object is done by visualizing it to each other. This approach is commonly known as image search. Instead of you describing the object (e.g. GUI element), the user can define it by using the picture of the object as a query. In particular novices benefit from this approach, because they aren't necessary able to identify the unknown object especially if it is a GUI element.

The first step in creating a help system based on image search is to extract images from a wide variety of tutorials, documentations, books, etc.. In the case of *Sikuli Search* the medium for the images are screenshots, which have been taken by the user or creator. To organize the system each taken screenshot is indexed by it.

After filling the database, the screenshots are modified with a set of visual words. A visual word is a vector of values describing the visual properties of a small patch in an image. The patches are well-defined, so scale, translation, brightness and rotations variations won't influence them. As a screenshot consists of many patches, each patch is indexed explicitly. Many GUI elements contain text; thereby they can be identified based on embedded text extraction by optical character recognition (OCR). In spite of using raw extracted string by OCR, the string is computed into 3-gram characters.

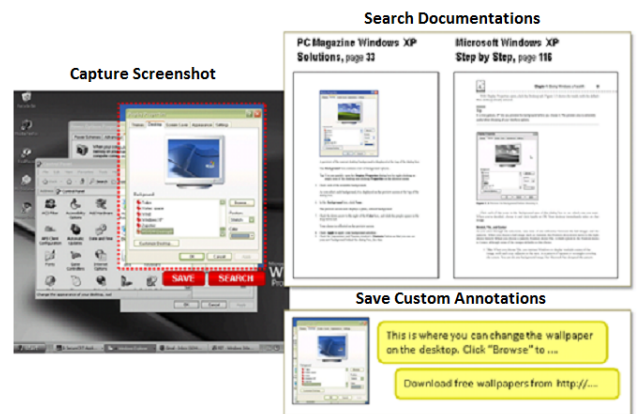


Figure 4. The desired part of the desktop is selected (Capture Screenshot) and compared to the help database of Sikuli (Search Documentation). The found article can be modified (Save Custom Annotations). [11]

Sikuli Search can be accessed anytime and anywhere in the running system through a predefined hot-key. The region of interest is pointed out by using a rubber-band rectangle to mark it. In the process small errors are corrected, since the screenshot representation scheme allows inexact matches. The image within the rectangle is used as a query by the system. After comparing the query with the screenshots in the database a web browser is opened to display the results. An example for this is shown in figure 4. The user can add annotations afterwards to the images, which allows customizing the database.

In comparison to word based search engines, the results of image engines have the same quality, but a faster query. Sikuli Search represents a new approach to search engines in general. As mentioned before the advantages of visualizing the problem of a user results in a more intuitive use of the system. However again the database is static and has to be maintained by an experienced user.

Yahoo Answers

Until now we have discussed new technologies in support that provide help to novice users. Each of them had a unique approach with a specific structure and design. However modern users do not use help systems to solve their problems, they prefer to seek their help through search engines like Google or communities like Yahoo Answers.

Yahoo Answers [2] users can interact with each other through a question and answer format. A user posts his questions to a specific problem and other users reply directly to it. However the other users are only able to reply once to a question. These interactions are posted within categories. The distinction between a user, who answers or asks questions, depends on the current category e.g. in the programming category only user with higher level of knowledge answers [2]. The users rate the replies and determine in the process the best given answer, which is not necessary an expert answer. The best given answer is not only the correct one, it also has some other traits like detailed description or a practical example.

Communities like Yahoo Answers have many benefits for novice users or in general users. It is more comfortable to interact with other human beings and to select one solution from different sources. Furthermore, the users rate the replies of other ones, so they can find their desired answer in old posts.

RECOMMENDATION

A recommendation is a tip that improves the abilities of the user. Responsible for giving recommendations are so called recommender systems. A good recommender system has to pay attention to certain aspects. It has to consider the knowledge of the user and the quality of the recommendation. Hence the recommendation

should either be a novel one or a useful one [7]. Novel recommendations are unfamiliar commands for the user and useful ones are commands that are useful to the user immediate or in future work. The combinations of both recommendation aspects is a two dimensional space with three categories of recommendations, figure 5.

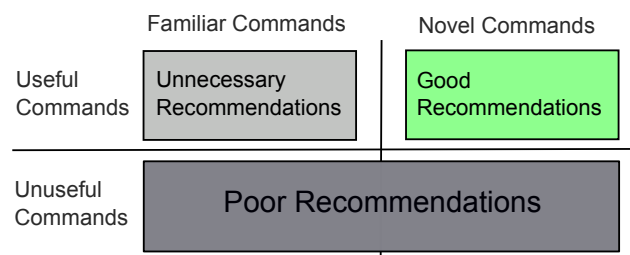


Figure 5. Quality of recommendation. [7]

The first category are the good recommendations consisting of commands fitting both, useful and novel recommendations. The second category are the poor recommendations, which are of no use for the user. The third category are the unnecessary recommendations that are useful known commands for the user. An unnecessary recommendation can either improve the user's confidence or can irritate the user as it is already known to him [7]. Besides these aspects it is important for a recommender system to have a good design. A recommender system should be installed in the background of the interface so it does not disturb the user when he is working. Furthermore the user should be able to access it within his current applet all the time. [7] The following recommender systems consider these aspects.

OWL

OWL, *organization-wide learning* [3], is a recommender system from 1998. Before using OWL the participants are divided into peer groups, which are equivalent to their specific division with the company. Each group consists of experts, who have variable level of knowledge. The resulting database records how often the participant has used certain commands in his code resulting to an individual user model. In doing so the recommendation for each individual is about 'what to learn next' considering the command use in his respective peer group. The recommendation is based on the usability and frequency of the command within the database. Furthermore an expert model is created by comparing each user model of a peer group by giving each command a certain expected value. This expert model is further modified by marking the importance of each command to avoid unnecessary recommendations, it is called expected model. OWL decides the importance of a command by considering its appearance in the peer group.

The presentation of recommendations is handled by two methods. The first one is the active method, Intelligent Tips, which displays a recommendation when the util-

ity of a recommendation is rather certain. The second method is the passive one, also called skillometer, which provides an interface. The user himself explores his and other expected models to learn new commands.

As OWL is a very old concept and has never been implemented or evaluated in detail, it is not clear if its approach is correct, because the recommendations are limited to the commands used within the company. Hence it is unlikely for this concept to work for an entire user community [7].

Community Commands

A new similar recommender system is *Community Commands* [7], which also personalizes the command recommendation using collaborative filtering algorithm. As OWL, the database of Community Commands collects data of monitored users for a certain amount of time. After this the collected data is rated for each user individually. The rating is based on the frequency the user has used the command. The collaborative filtering algorithm evaluates the modified data and predicts how the user would “rate” commands not known to him. The system then generates a top ten list of commands the user can access when he desires. This is shown in figure 6.



Figure 6. Generation individual top ten list. [7]

The two most commonly used collaborative filtering algorithms are user-based and item-based, which inputs are the command history of each user in the community and the command history of the current user, who gets the recommendation, who is called active user. User-based collaborative filtering creates a group of users most similar to the active one. It averages the group’s command frequency to generate the group’s command expected frequency table for the user. The table is changed after a certain amount of time to avoid suggesting known commands to the active user. Item-based collaborative filtering creates a group of unused similar commands to the ones the active user knows. The resulting top ten list is sorted by the similarity of the unknown commands and already known commands of the active user.

Both algorithms have two modifications in Community Commands to improve the quality of recommendation. One modification is upgrade, which ignores recommendations when the recommended command is less efficient as a known one. The other one is equivalences. It avoids a recommendation of a command that is syntactical different but has the same semantic as a command known to the user.

The concept behind community commands elegantly deals with the requirements of recommender systems. Especially the idea of an individualized recommendation greatly improves the quality of it.

HelpMeOut

HelpMeOut [5] is a crowd sourced recommendation system for programmers. The idea is to suggest possible solutions for errors in their programs. All suggestions are created from collected data of the users.

Novices start programming by modifying existing code. They seek for help in e.g. online forums [5]. The hypothesis behind HelpMeOut is suggesting possible solutions from similar problems directly into the programming environment helps users to correct their problems faster.

HelpMeOut tries to collect code changes that made error state code into error free state code. HelpMeOut realizes this with syntax errors as well as with runtime errors. Every time the user starts or compiles his program and he gets an error message. HelpMeOut saves this message its source file. As soon as the program surpasses the last point of failure it marks the error as fixed and sends the code difference to the Server. For compile errors, the error message is enough to determine whether an error has been solved or not. Stack trace, execution count of lines and input are important to run time exceptions. Each error results in a query to collect the best matching and error correcting suggestions. Suggestions are ordered by source similarity, stack trace similarity and user voting. They are displayed into the IDE (figure 7).

Error Message:
java.lang.ArrayIndexOutOfBoundsException

Suggestion 1

Before (Broken)	After (Fixed)
<pre>4 for(int i=0; i<200; i++) { 5 myArray[i] = 0; 6 }</pre>	<pre>4 for(int i=0; i<200; i++) { 5 if(i<myArray.length) { 6 myArray[i] = 0; 7 } 8 }</pre>

[more info](#) | [vote up](#) | [vote down](#) | [find line](#) | [copy fix](#)

Problem: The variable i which is used to index the array could be greater than the length of the array. **Solution:** insert an if-statement to check that i is not greater than the length of the array.

Figure 7. Suggestion for runtime error, which includes an explanation of the fix. [5]

The user study showed that 49% of the queries were useful suggestion. The study only lasted over 39 person-hours and the system heavily relied on user contributions, so it was surprising to achieve such a number. But the ratio between useful and not useful queries did not increase over time (figure 8). A reason for this high ratio in the beginning could be that all novices had the same tasks and might have run into the same problems. Furthermore the lack of improvement over time might be explained with a bad structured database where right suggestions get lost in too many not useful suggestions.

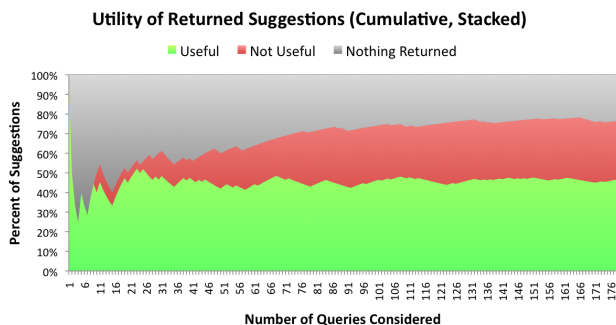


Figure 8. Evolution of useful suggestions over time [5]

SIMPLIFICATION

With Simplifications we mean simplifications of programming languages. The past has shown that modern programming languages like C++ are often too complex for novices to start with them. The novices despair of compiler errors or runtime errors, because they do not understand enough to solve their problems. To accommodate this problem, simpler programming languages have been invented. Early attempts were programming languages with a simplified syntax like *Logo* and later *Pascal*. Modern simplified languages also introduce visual elements into the language, to make the language feel more natural. *Sikuli Script* allows to have thumbnails within the letters of the source code, *Scratch* completely waives textual input and only uses movable, visual components.

Scratch

In the late 70s to 80s there was an initial interest to teach every child how to program. But soon these projects died out for several reasons. Early programming languages were too difficult for children to learn. Even the teachers in those days were not programmers themselves, so they were often not able to give an appropriate support. Additionally beginner programming courses were connected to activities like sorting lists of integers or generating prime numbers. Those activities were not appealing to the students. Today many programmers learn by downloading and modifying code from the internet. They get motivated through the comments of other people using their software. This was not possible in the past. But since computers become more and more important in our everyday life, the interest that everybody is able to program increases. Today's

“digital natives”, people who grew up in an environment, where computers are very common, are very comfortable in sending text messages, playing online games, and using all sorts of interactive media. But they only interact, they cannot create this type of media. It is as people could only read, but not write [8].

Scratch is a software project that tries to accommodate those problems. It is designed to nurture a new generation that is comfortable using programming to express their ideas. It should appeal to people who don't imagine themselves as programmers, especially young children. It has to be easy for everyone, no matter what ages, backgrounds and interests they have. It is not designed to create the computer scientist of the future, it is designed to make everybody comfortable with this way of thinking.

Scratch consists of two main modules. The IDE is the programming language and a community based web interface.

IDE and language

Many people without deep understanding of programming cannot distinguish the GUI of a system and the system itself. Even programmers often call for example Dev-C++ as their compiler. This type of thinking is described as “the GUI is the System”, transferred to programming this can be called “the IDE is the programming language”. In consideration of this Scratch has been developed. Scratch does not intend to distinguish between IDE and language. So each time language appears in this section, both IDE and the language is meant.

The language's design is inspired by Alice and Squeak Etoys. They are visual programming languages to simplify programming for starters. But those projects were not satisfying enough to the Scratch team, so they designed their own language. In visual programming languages you do not write your code with a text editor, you drag all code segments per drag and drop together. Scratch also gave them special form and color. Instructions have little knobs like Lego bricks, to illustrate how they fit together. Start Blocks, similar to something like `int main()` in C, have knobs under them, to illustrate that there might be placed a block below them. Control structures and loops are shaped like a big C and have those knobs inside of them, to show that instructions are meant to be placed inside of them. Arguments of expressions are also specially formed to indicate their data type. Boolean expressions are hexagonal shaped and red, while Number type arguments are round shaped and blue. Objects are directly selected with a drop down menu. Shapes of instructions not only give a hint where they can be placed they even make it impossible to place them at positions that are syntactically incorrect. In my experiments Scratch was even resistant to all runtime exceptions, so programs that have been written in scratch can't even crash.



Figure 9. One block in two languages

New instructions can be dragged from a categorized list of commands into the editor and back again. Stacks of code can be placed wherever you want them to be. You can move the parts freely across your desktop. They are not bound to any raster like text based languages. This allows to hang out parts of the code but leave them beside the code where it came from. Those free hanging code blocks appear like commented lines of code in textual languages, but in scratch you can still activate those blocks by double clicking on them. This is one core strength of the language. You can execute every block of code separately, you can even change the code while it is running. This feature allows more interaction with the program itself, and makes it much easier to try something out. Even concurrent programming is essentially easy, because everything you need is to give two stacks of code an equal starting block, and they start concurrently. Executing stacks are highlighted so that is always visible what is going on. This debugging feature by default also helps to understand the flow of a program. On the right side of the IDE there is an always visible window that shows the scene to interact with. Scratch does not have a command line like most programming languages have, it serves this little 2D scene to interact with. On the scene you can create and place new objects. The program controls those objects. The scene also represents the final project. This sort of graphical media has been chosen, because it is much more appealing to children than lines on a black and white console. At last there is the publish button. This button publishes the complete project onto the scratch website and shares it with the community. This is done with just one button click.

Scratch Website

The website allows browser based access to all Scratch applications. Its structure is very similar to YouTube, but instead of having videos it serves interactive scratch projects. A sample project can be seen in figure 10. Another key difference is that all projects are forced to be open source. The reason that leads to this decision was that the people should learn from each other. And many studies have shown that learning by modifying found examples is easier than creating from scratch all the time. To give tribute to the original author of modifications, there has been introduced the remix functionality. Af-

ter downloading, modifying and uploading the project again, the website marks it as a remix with a link to the original one. Also remixed projects get links to their remixes. The names remixing and scratching both have their origin in activities from a DJ, because remixing a scratch project should feel like remixing tapes like a DJ.

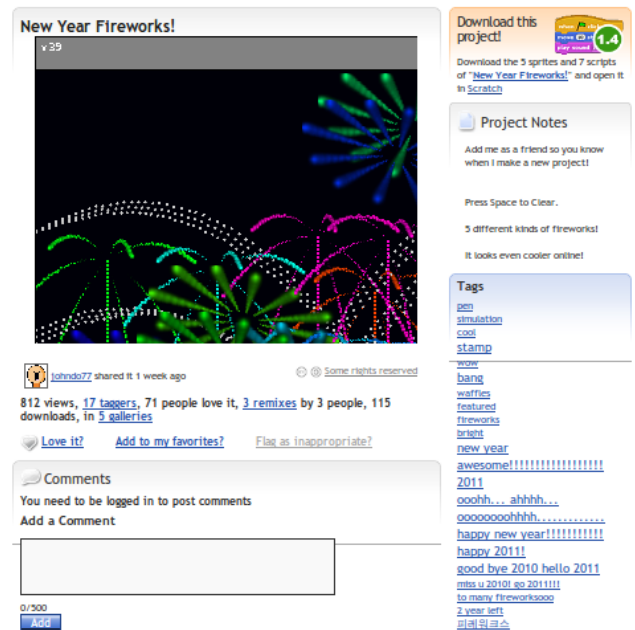


Figure 10. A sample project from the website

Even scratch is still in development, its first version was released in 2007. Since then the website has grown and has more than 1,500 new projects are added to the site each day, there are currently nearly 1,500,000 projects hosted on their website. The age of the community is between 8 and 16 with a peak at 12, but Scratch's simplicity also attracts many adults too. Already some schools started to teach scratch to their students. With this development scratch can only be seen as great success. [8]

Alternatives

Scratch is not the only simplified programming language that works completely on visual elements. Two of the alternatives are *Alice* and *App Inventor*. They both share the concept of programming with command bricks, but they both have their advantages and disadvantages compared to Scratch.

Alice was invented much earlier than Scratch. Its first release was in 1999. The developers designed syntax of the command bricks close to Java syntax, in order to be able to export the source into Java files and back again. Additionally this allows the user an easier transition to pure Java. Alice is designed to animate sketches in a 3D environment (figure 11). In this environment the user can place objects very similar to the scene in Scratch. Creating content for a 3D environment is

also more complicated than making simple sprites like the scratch content. Thus the University, where Alice has been developed, has entered to a collaboration with Electronic Arts to enable the users to import all types of content from the video game “The Sims”. The close development of the language to the Java syntax has also its disadvantages. Today Java is indeed one of the most common programming language at universities, but they are not intended to teach the students to program Java, their main focus is to teach them programming in independent of the language.



Figure 11. a scene created in Alice [You Tube]

App Inventor for Android is the newest invention that uses blocks to program. It is developed by Google and got its first release in December 2010. This program allows to write applications for the Android operating system. The language is designed with much influence of Scratch, but Instead of having a scene with movable objects, App Inventor uses a classical GUI builder to create programs. So the main focus of applications developed with App Inventor are intended to be more mature than the Scratch creations. The main disadvantage of App Inventor is that it only supports Android mobile devices to develop software for at the moment. But in the future App Inventor could develop into a complete software development IDE that is not limited to a single platform. Figure 12 shows how a source file looks like.

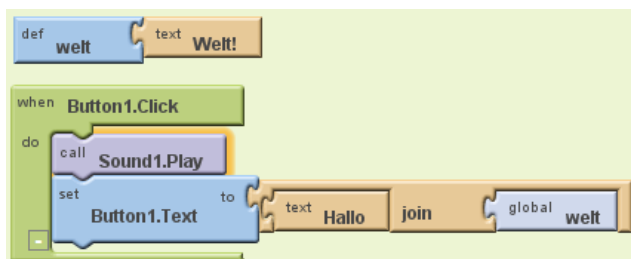


Figure 12. App Inventor Hello World

Sikuli Script


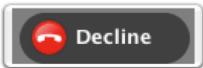

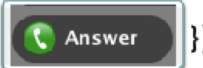
Sikuli Script [11] is an image based technology. It is an extension of the scripting language python with image searching algorithms and direct image representation in the language itself. Sikuli Script uses its image search capabilities to use the GUI directly as an API. Instead of calling named functions in an API you can import images of certain GUI elements directly into the scripting language. Then you can perform input operations on them like clicking and typing.

Most programming or scripting languages require knowledge over an API to automate some tasks in the environment. But users are often only confronted with a GUI. So there is always the burden to learn the API before any script writing is possible. Sikuli Script can directly operate on any GUI, therefore there is no burden to learn the API anymore. Though Sikuli Script was developed to automate tests for GUIs, it is also a great improvement in teaching programming. The common way of programming requires a certain amount of knowledge before the first useful program can be written. The simplicity and the visual integration of images in the code removes a bigger part of this required knowledge. After the first lesson a student might already be able to write a useful script that automates a simple task.

The scripting language itself is just a dialect of python that allows to have thumbnails of images (patterns) within the source code. With the select tool it is possible to select a region on the screen that is then transformed into a pattern in the code. Patterns are primary used for searching. e.g.: `click()` gets one pattern as parameter. At first it locates the pattern on the current screen, then one mouse click will be simulated on the location of the pattern. Patterns can also be generalized. `Pattern(a).similar(0.8).anyColor().anySize()` is a Pattern that accepts all images with at least 80% similarity, that may have a different color and a different size. But the less accurate a pattern is, the longer it will take the computer to find it. A simple snippet that filters phone calls is presented in figure 13. Here are patterns used as a key in a dictionary.

Sikuli Script is not a universal solution. Because it is working directly on the screen data, it is limited to everything that is also visible to the user. If something is hidden by another window, Sikuli Script cannot find it. On the other hand using only screen shots for searching makes Sikuli Script very portable to all platforms. Currently it is available for Mac, Linux and Windows. But the easier the porting of Sikuli Script is, the harder the porting of the scripts is. All scripts are very dependent of the skin or style of the GUI on the developer’s computer. The change of one color in the developers desktop environment can cause that all scripts will not be able to work anymore. This dependence on the style of the system and the limitation to screen data, make these scripts very useless in real application develop-

ment. Here Sikuli Script should stay inside of the test environment where it is developed for.

```
v = VDict({
   : ,
   : 
})


while True:
  call = find()
  if call:
    face = capture(call[0].x, call[0].y, 65, 65)
    doubleClick(v[face][0])
```

Figure 13. A snippet in Sikuli Script

CONCLUSION

In the field of *support* most concepts could solve the flaws of currently used help systems. However a bigger part of these concepts haven't been realized in practice. Therefore potential approaches to improve distribution of help are wasted. Responsible for this is the lack of sufficient user studies. The exceptions to this are AIDE and CHIC, who have sufficient user studies and a detailed implementation. Another problem is the general attitude towards help systems, which makes them less interesting to analyze. Modern users prefer to use search engines or communities to seek an answer to their problem. So future research should consider a combination of help systems and such media.

In contrast *Recommendation* are often used by users to improve their working experience. The presented concept of community commands introduces many new aspects like using a community based database. Therefore it can deliver an individual and dynamic recommendation according to the needs of the user. Additionally it considers old concepts like OWL to learn from their flaws. On the other hand, HelpMeOut helps the user to understand errors and debug them. In the future these systems could be great tools to get used to new programming languages.

Simplification provides an easy start to learn basic programming. In providing development tools suited for children the way how programming will be taught at schools could change very much in the future. But if the children get used to a simplified general programming language, they might not want to learn new more complex languages. In the past pascal was developed as a teaching language, and was later used in the industry, because the people were already used to that

language [6]. This development could also happen to languages like Scratch.

REFERENCES

1. M. S. Ackerman and T. W. Malone. Answer garden: a tool for growing organizational memory. In *Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems, COCS '90*, pages 31–39, New York, NY, USA, 1990. ACM.
2. L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *Proceeding of the 17th international conference on World Wide Web, WWW '08*, pages 665–674, New York, NY, USA, 2008. ACM.
3. D. J. Frank Linton, Andy Charron. Owl: A recommender system for organization-wide learning. *AAAI Technical Report WS-98-08*, pages 65–69, 1998.
4. C. A. Halverson, T. Erickson, and M. S. Ackerman. Behind the help desk: evolution of a knowledge management system in a large organization. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work, CSCW '04*, pages 304–313, New York, NY, USA, 2004. ACM.
5. B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the 28th international conference on Human factors in computing systems, CHI '10*, pages 1019–1028, New York, NY, USA, 2010. ACM.
6. M. Hill. Why pascal is not my favorite programming language. paper.
7. J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. Communitycommands: command recommendations for software applications. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology, UIST '09*, pages 193–202, New York, NY, USA, 2009. ACM.
8. M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: programming for all. *Commun. ACM*, 52:60–67, November 2009.
9. G. Stevens and T. Wiedenhofer. Chic - a pluggable solution for community help in context. In *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles, NordiCHI '06*, pages 212–221, New York, NY, USA, 2006. ACM.

10. L. Vouligny and J.-M. Robert. Online help system design based on the situated action theory. In *Proceedings of the 2005 Latin American conference on Human-computer interaction, CLIHC '05*, pages 64–75, New York, NY, USA, 2005. ACM.
11. T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using gui screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology, UIST '09*, pages 183–192, New York, NY, USA, 2009. ACM.