Towards An Efficient Evaluation of the Usability of Android Apps by Cognitive Models

Stefan Lindner*, Philippe Büttner*, Gabriele Taentzer, Steffen Vaupel**, Nele Russwinkel***

**Technical University Berlin, Berlin, Germany (e-mail: slindner@zmms.tu-berlin.de).*
***Philipps-University Marburg, Marburg, Germany*

Abstract: Mobile apps offer new specific design challenges that may change some aspects of the learning process of the user. When less information can be presented at once due to smaller screen sizes, more context information has to be kept active in memory. Moreover, apps are often used as secondary task and thus have to be more robust to distraction, interruption and lower availability of attentional and other memory-related resources. How can we evaluate the cognitive suitability of mobile applications? Our goal is to develop a methodology that helps designers to test alternative design variants of their application in order to optimize it. Specific cognitive modeling approaches can address these challenges and can explain human behavior relevant to usability criteria. Furthermore, modeling approaches in this domain can help us to understand the cognitive mechanisms driving our behavior. This paper focuses on three central questions: (1) How can software developers and cognitive modeler contribute to each other's work? (2) How do cognitive models have to look like to be well suited to mobile apps? (3) How can implemented apps be connected to cognitive models?

1. INTRODUCTION

The number of new mobile applications (frequently referred to as "apps" in the text) on the market is strongly increasing and the user expectations on app usability are rising as well. In general, a number of tools have appeared to help developers designing and implementing good software systems in little time. However, the quickly developing market of mobile apps is working with products that have affordances different from user interfaces coming from applications running on e.g. personal computers. First of all, mobile applications have to be easy to use, otherwise they are not used. The interaction time should be as short as possible, because people often use apps while performing other tasks or waiting. The interface is relatively small such that less information can be displayed compared to other interfaces. Also, the use of mobile apps should not suffer from interruptions, since they are often used as secondary tasks and may be disrupted by external cues such as interaction with other people and noise.

In summary, usability factors are even more important for mobile apps than for other user interfaces, due to smaller devices and more disruptions that may take place. Apps with poor interfaces are very likely not used and not recommended to others. Hence, consumers would switch to other products. But how can the usability of an application be evaluated leading to a better interface? Our goal is to use the method of cognitive modelling to evaluate the usability of smart phone applications and to support developers to decide between design alternatives.

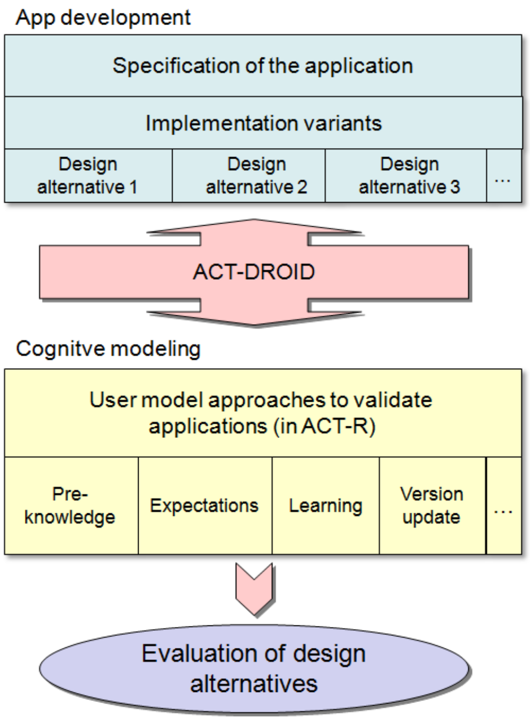

Figure 1: Methodology for evaluating applications with a cognitive modeling approach

In this paper, we want to show how this kind of evaluation can be achieved and how cognitive processes can be analysed. The three main contributions are: (1) the interdisciplinary approach showing how software developers and cognitive modeler can profit from each other, (2) the introduction of a new tool ACT-DROID to connect a running app with a cognitive model and (3) new modeling approaches of expectations and use phases, especially for mobile apps.

We propose four steps to reach our goal. Software developers usually start with a requirement specification of a planned application. There are numerous possibilities of design alternatives that would fulfil the specification, but which alternatives are the best? Different design alternatives can lead to several implementation variants of an app (see Figure 1).

In step one cognitive user models are developed. The requirement specifications help the modeller to specify certain aspects of these models. It provides vital information concerning what the user knows before she uses the app, what kind of expectations and what goals she has. Cognitive models are implemented in the cognitive architecture ACT-R. How this is done and what aspects have to be taken care of is described in detail in Section 4.

Then, we need the cognitive model to interact with the application. Some existing tools, as e.g. *cogtool* (Teo et. al. 2007) and *Memo* (Möller et. al. 2008), need prototype generation before this is possible. We introduce a new tool called *ACT-Droid* that can connect the cognitive models directly to the application. How this works and what needs to be done to use this tool is explained in detail in Section 5.

In the next step, some candidate variants of the implemented app are tested in a usability test (not shown in Figure 1). Test cases that are described in the requirement specification can help in designing the right usability tests.

Thereafter, the obtained model results (i.e. reaction times, user errors and kind of performance) are compared with experimental data. If the experimental data deviates too much from model predictions, we return to the beginning, improve the model and go through all subsequent steps again. If, on the other hand, the model predicts the experimental results well, it is very likely that we understood the cognitive mechanisms guiding users when they interact with mobile apps, and have a general modelling approach that can capture this aspect.

In a first approach, we are especially interested in how the usability of an application changes in respect to certain general features of apps, e.g. their menu depth or the number of items on each screen. To demonstrate how modelling can help with this question, we detail all steps and methods involved in our approach in Section 6.

2. DEVELOPEMENT OF MOBILE APPLCIATIONS

2.1 Development Steps

Modern software development is performed in an agile way where functioning software is developed and tested early (e.g. Cockburn 2002). Customers are involved in the development process continuously by testing and giving feedback to new versions of the software. Typically, software is developed in a number of iterations adding functionality to running software step-by-step. While first iterations concentrate on the realization of core functionality, later iterations complete the functionality towards a mature software system. Typically, the development of mobile apps follows this agile process as well.

To understand what kind of functionality shall be realized and which other requirements the customer has, software development should start with a requirements elicitation phase. Interviews, observation of work flows, standards and other kinds of documents help to find out what kind of software system the customer wants to have and lead to a requirement specification. It usually consists of two main parts: The functional requirements describe what functionalities the software shall have, while non-functional requirements are additional statements concerning the user interface, quality of services, technology to be used, etc. Functional requirements are structured along so-called use cases that can be considered as a kind of functional units. The description of a use case is structured by giving information about the kind of users, also called actors, pre- and post-conditions and its usual work flow. Especially this work flow can be used to deduce a cognitive model for the application to be developed.

Following the paradigm of test-driven development (TDD) (Beck 2003) (being a major ingredients of agile development processes), a software system is not only tested at the end of its development but continuously throughout the development. One task in TDD is the specification of test cases already in the requirement specification phase. A test case specification belongs to a use case specification. Specific examples for use cases and test cases are given in Section 6.1.

These test case specifications are also appropriate to specify some aspects of the cognitive tasks to be performed, such as expectations and goals of the user. They can also help in designing appropriate user tests by specifying typical user input and device output.

2.2 Challenges

Functional requirements for a mobile app define the broad structure of the applications and specify what tasks the user must be able to perform using the application. This means that a functional requirement specification determines what shall be implemented in the app. It does not, however, specify how an app shall be implemented. When deciding on the specific implementation, the developer has to make decisions concerning the user interface of the app, in particular. It includes decision on menu structures, especially number of items on each menu level; arrangement and graphical appearance of items and information, on recurring control elements, and on the realized workflow. Throughout the paper we will focus on one specific UI aspect: the depth of menu structures.

When choosing between implementation variants that have the same functional abilities, deciding factors are non-functional requirements such as usability, efficiency and reliability. It is unrealistically expensive to implement and test the usability of a wide range of app variants having the same functions. Cognitive modelling, however, offers a way to perform this kind of testing in a non-expensive way.

3. USABILITY OF MOBILE APPLICATIONS

3.1 Usability Criteria

ISO 9241 defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."

There are several ways to ensure usability. There are usability heuristics (e.g. Nielson, 1994) that can lead developers to consider different aspects such as “visibility of system status” while they develop user interfaces. There are also newer heuristics that are especially developed for mobile applications (Kuparinen, Silvennoinen & Isomäki, 2013). For developers, however, it is very difficult to put themselves into the position of someone who does not know the program structure and the exact purpose of the application and who probably has little technical background. The best way to evaluate an application is to conduct user tests with participants. But these tests are usually time consuming and cost-intensive. Moreover, the test result often just shows that people favour one app over the other but they do not provide information about reasons for that. Therefore often expert interviews are conducted instead. Here, so-called experts give their opinion about what can be solved better and where problems might arise. Different experts, however, can give different, if not contradicting, opinions about the same case. In this paper we will present an approach of evaluating usability that lets cognitive models "test" the technical devices. To this end we use a specific cognitive modelling approach of app-evaluation linking cognitive models with smart phone applications (described in detail in Sections 4 & 5).

Typical ways to measure usability in human computer interaction include the collection of quantitative data, the measurement of the user's subjective evaluations and the collection of environmental (context) data (Kronbauer et. al., 2012).

Criteria that best establish the usability of a certain technical device may vary between devices and may change over time with changing technologies and user expectations. A method for establishing usability criteria for a new technology is presented in Rusu et. al. (2012). In our approach, we stick to a rather traditional set of usability attributes that, according to Zhang & Adipat (2005), play an important role in the use of smart phones. Those usability attributes include learnability, efficiency, memorability error, user satisfaction, effectiveness, simplicity, comprehensibility and learning performance. Table 1 (appendix A) contains a short description of each usability criterion and typical experimental ways to measure it. Most of these attributes can be addressed by cognitive modelling.

Usability of smart phone apps is linked to several factors including the structure of an app, its use context, and the knowledge, abilities and goals of the user. We are specifically interested in the question how usability is affected by certain structural and design decisions concerning the app. We also want to stress the changing usability of smart phone apps depending on the point in the user's interaction history. This interaction history is divided into distinct use phases.

In our cognitive modelling and experimental efforts, we consider the entire time span of using a technical tool. Our goal is to emphasize the often neglected importance of considering changing cognitive demands during different use phases for design decisions.

3.2 Use Phases

The description of learning processes is central to the modelling effort.

Prior to the *first use* of the application (i.e. the *pre-use phase*), the user already has certain knowledge and expectations concerning the application. More general expectations may stem from previous experiences with smart phone applications, while some more specific expectations may stem from the fact that the user decided to use the application in the first place.

The *first interaction* is, depending on the prior knowledge and expectations of the user, a more or less guided exploration that involves both, goal-directed behaviour as well as trial and error. When users try to explore the application, they are guided by preconceived expectations as well as their app-specific goals. A divergence between reality and either expectations or goals triggers a process that leads to a change in expectations, goals and/or the manipulation of the app.

By *continuously using* an application, the user gets both, a better understanding of its abilities as well as of its structure. The application gets easier to use; speed and accuracy improve while the number of errors drop. Correct knowledge and expectations about the app, in general and in specific situations, can be accessed from memory more easily and more reliably.

When the application is *not used* for a certain amount of time, decay processes set in for declarative knowledge elements. Proceduralized knowledge, i.e. subconscious and rehearsed cognitive and action routines, also decay, but at a much slower rate (Arthur et. al., 1998). If, in addition, applications similar to the original application are used, certain actions and cognitive routines can be altered in a way being disadvantageous for the use of the original application. Both, the length of pauses in use and the use of similar apps in that time span, can influence the use of the original app and ultimately, its usability.

4. COGNITIVE MODELING OF THE USE OF ANDROID APPS

In order to better understand and predict human behaviour, cognitive modelling strives to create computational models that, as a main focus, reflect human cognition. Very extensive and well specified computational frameworks of cognition are called cognitive architectures. Many cognitive architectures stress cognition as embedded in and interacting with bodily functions and the environment.

4.1 Cognitive Modelling and ACT-R

The cognitive architecture ACT-R (Anderson et. al. 2004) is especially suited for describing memory mechanisms and thus lends itself well to model learning processes and expectations that play a role in the use of applications. ACT-R is supported by a big, active research community that continually provides new input and ideas. The architecture is open source, so it allows for the integration of a self-developed components and extensions. ACT-R is implemented in a free-to-download programming environment (Bothell, 2007).

ACT-R assumes a modular structure of the brain, i.e. separate subsystems being responsible for different tasks of perception, cognition and motion. Its set of *modules* includes declarative memory, motor, vision and intentional (goal) modules. They interact by their interfaces called *buffers*. These buffers are manipulated by the means of *productions*. Productions are rules that contain a conditional part (the LHS) specifying the combination of buffer states at which the rule fires, and an action part (the RHS) that specifies resulting buffer changes. Models in ACT-R consist of these production rules and, once started, run continuously until no conditional part (the LHS) of any production matches the current combination of buffer states. Productions have a property called *utility* that rules the probability that they are used - should the LHS of multiple productions match a certain buffer state.

Declarative knowledge is stored in ACT-R as knowledge units called *chunks*. Each unit has a property called *activation* attached to it. It rules the probability and the speed of recall. The activation of each chunk constantly changes, depending on when, how often and in which context the chunk is used. In general, this leads to knowledge being more easily available when the probability of it being needed is high.

4.2 Advantages of ACT-R

Modelling human-machine interaction in ACT-R offers a great range of advantages. ACT-R is grounded in an extensive body of research findings and is continually being expanded as the understanding of the mind increases. ACT-R is implemented in a programming environment (currently the ACT-R 6 environment; Bothell, 2007), allowing for the construction of complex models and the modelling of complex and extensive behaviour. It thereby exceeds the scope and possibilities of most traditional theoretical models in both regards. Another advantage of the ACT-R architecture is the possible coupling of the model of a technical device with the cognitive model of the user.

4.3 Cognitive Modelling of Android App: Expectations

Expectations play an important role in all use phases; they are certainly a main issue of the presented modelling effort. In the following we will demonstrate some functions and abilities of ACT-R by describing how expectations can be implemented in the architecture.

In the description as well as in the modelling of the learning processes taking place during the use of an app, we distinguish between three major types of expectations: *control expectations*, *universal expectations* and *experience-based expectations*. In the following, we will shortly define each expectation type and describe their importance in the different use phases. The presented framework is novel, but grounded in prior research and concepts.

Expectations play a prominent role in action-perception feedback loops. Friston & Kiebel (2009), for instance, describe all human actions in a framework on continuous reconciliation between expectations and observed outcomes. *Control expectations* are active in an immediate action-perception feedback loop. They are general rather than subject-specific and, as the name suggests, help with the short-term, immediate control of the environment. Control expectations are especially important during the initial use phase, since first uses consist of a lot of (goal-oriented) trials and errors.

Kurup et. al. (2012) provide an interesting expectation framework that can be used for describing some control expectations. They introduce two new buffers to ACT-R, which track the elements of the environment that are not reflected in current expectations or goals. Their framework constantly produces expectations and actions. The action results in turn are compared with the expectations and goals of the individual (in that order) and new behaviour and expectations are produced. The framework allows for the easy implementation of general control expectations that can be formulated as general productions.

Universal expectations are expectations that result from the universally inherited pre-structuring of the environment. One type of very general universal expectations are the so-called "Gestalt" laws (Brunswick & Kamiya, 1953). They describe a very general structuring and interpretation of visual patterns that can be found in a wide array of subjects. Universal expectations play the most important role in the initial use phase when specific expectations about the app are not readily available yet.

There are several ways universal expectations already find their way into ACT-R models. The Lisp-Environment that typically interacts with model already presents an environment to the model that is heavily pre-structured by the modeler herself. Typically the model is given whole objects (such as a button) and their relevant features and interactions; it does not have to piece them together by itself out of a complex environment. This heavy pre-structuring provided by the modeler is also visible in the productions, as the logic of the productions are to a certain degree considered self-evident (this is how a human would plausibly process this state of buffers/ state of environment).

We are currently devising additional concepts to implement universal expectations in ACT-R. With our specific task in mind, we are especially interested in modeling "Gestalt" laws that play a role in smart phone app interface use. One main idea is, once a certain control element is used, to establish new productions concerning similar or close control elements. To be more specific, we take as an example the law of proximity (close items are perceived to be similar in functionality). Whenever an element of the graphical interface is used successfully while a certain goal G is present in the goal buffer, it might be feasible to automatically create additional productions for all goals related to G that contain the control elements close to the current element. Their utility would be proportional to the inverse distance of the control element to the current element. This way we establish action and cognitive routines that would prefer close items for similar or related tasks.

Experience-based expectations are expectations that result from individual knowledge and experiences in an individual's lifetime. In contrast to universal expectations, experience-based expectations are highly dependent on the individual learning history and the specifics of a user's environment. Most expectations in a certain situation may stem from personal experience in situations that are similar to the present situation. They may also result from the past observation of other's behaviour, other’s verbal account or personal inferences.

Looking closer, there is also a fine distinction between those expectations (being referred to as *general* experience-based expectations) and expectations that result from the personal learning history in the exact situation (being referred to as *specific* experience-based expectations).

General experience-based expectations play a big role during pre-use and the initial use phase, when specific experience-based expectations are not available or scarce. Those general expectations are overtaken by specific ones when specific knowledge is acquired during repeated use. When the use of the application is interrupted for a longer time span, general expectations gain some ground again, as explicit knowledge about the app is partially being forgotten.

In general, experience-based expectations can be both reflected in productions and declarative chunks. Both the utility- and the activation mechanisms reflect strong expectations about our environment: what was recently or frequently encountered will likely be encountered again; behaviour that worked before will work again; behaviour that was displayed shortly before a successful completion of a goal will likely cause the completion of this goal again. Again, additional mechanisms might be needed to reflect the full range and functionality of expectations.

To reflect the preparatory nature of (explicit) expectations, it might be plausible to call expectations (chunks detailing the events in the immediate future) into the *imaginal buffer* (a sort of short term memory in ACT-R) to enable faster and better reaction in case of the expected event occurring. (Lindner & Russwinkel, 2013)

5. *ACT-DROID*: INTERACTION OF ACT-R WITH ANDROID APPS

The ACT-R system includes the capability to create simulated environments, such as screen interfaces. Production rules have the ability to interact with this environment by perceiving objects and making motor movements through perceptual and motor buffers.

The externalization of all necessary methods involved in the perception of objects and the facilitation of motor movements makes it possible to extend the environment to a world outside of ACT-R, without requiring a modification of its architecture. An interaction with production rules can be enabled with any device that meets the specifications of these externalized methods. Due to the fact that Lisp supports communication via the TCP/IP network protocol, it becomes possible for ACT-R to interact with any environment also possessing the capability to communicate via TCP/IP.

We can take advantage of this trait and have developed a tool for linking cognitive models written in ACT-R to Android applications written in Java. This tool is based on the concept of "Hello Java" (Büttner 2010) linking ACT-R to Java applications.

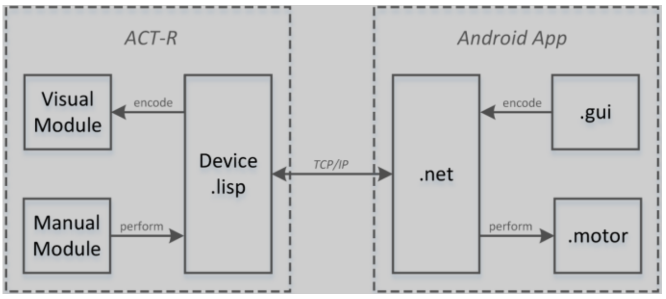

Figure 2: Structural Chart of ACT-Droid

5.1 Design

To link a cognitive model written in ACT-R to an Android application, two elements that coordinate the interaction between the model and the simulation are required. The first of these elements is a Java package that must be added to the simulation. The second element is a device for the ACT-R software that bridges the incompatibility between Java and Lisp with the TCP/IP network protocol.

The line of communication between Java and Lisp is established through a TCP/IP connection, which is a protocol commonly used to connect computers to the internet. All information sent with this protocol can be transmitted as a string representation. This string representation carries data pertaining to perception and action that is exchanged between the cognitive model and the application. In order to be able to communicate via TCP/IP, each side must encode and decode information using a common vocabulary and unified grammar. ACT-R receives data pertaining to perception and sends back data pertaining to the execution of motor movements. Java, meanwhile, receives data instructing it to perform the cognitive model's desired action and provides a visual representation of all objects visible in the frame.

5.2 Extending the Android app

In order to access an Android app from ACT-R, the app itself must be extended by a package that offers functions to observe all objects of an app, performs actions on the app, and exchanges information with ACT-R. In order to use this functionality, the package must first be added to the app project. It consists of three sub-packages. Their descriptions and roles are listed below:

Robot: This sub-package triggers actions like touch events and stroking a key, if the cognitive model decides to perform one of these actions.

GUI: GUI contains the information of all objects visible in the app. By recursively accessing objects of the app, data pertaining to the following objects becomes accessible: labels, text fields, buttons, radio buttons and toggle buttons. In principle, every Java object can be accessed. Therefore, information pertaining to the kind, value, colour, size and relative position of an object must be encoded to string representations. This information is necessary for the visual icon of ACT-R. The visual icon provides information about the kind, value, colour, size and relative position of visible objects in the environment.

Net: This sub-package provides all methods responsible for encoding and decoding information and handles the network connection with ACT-R. A socket process is started that waits for a connection from ACT-R on a predefined port. If the socket process receives information, it proceeds to parse it, thereby enabling it to perform actions based on methods obtained from the robot sub-package. Furthermore, it can send back visual information from the GUI package.

5.3 ACT-R device

As described above, ACT-R 6 provides externalized, accessible methods responsible for the perception of objects and the execution of motor movements. These methods can be implemented and utilized as devices, resulting in an interaction between a particular device and the production rules.

To update the visual icon of the visual module, it is necessary to call an update-method. This method regulates the gathering of visual information, but is not one of ACT-R's device methods. The update-method sends a request to this tool instructing it to collect data pertaining to all visible objects present in the Android app. Once this visual data is transmitted to ACT-R, it is then written into the visual icon of ACT-R.

6. CASE STUDY: A SHOPPING LIST APP

To test our cognitive modelling approach in the application domain, we use a demonstration Android application. The application supports the shopping of daily goods. The user may select goods that are put into the cart. Different variants of this application are used to demonstrate the impact of certain design features on its overall usability. Here we will present two variations that vary most significantly in their menu depth.

6.1 Requirement Specification of the Shopping List App

Throughout the app development, we specified all the requirements to this app that have to be satisfied: In the following, we concentrate on functional requirements and specify the main use cases of the demo app in Figure 3.

Title:	Search for goods
Actor:	Buying user
Description:	The user searches for goods and selects them.
Pre-conditions:	none
Process:	
	1. The user selects a category of goods, either alphabetically or along the kind of shops 2. The user selects one or more goods. 3. The user continues at action 1 or finishes.
Post-conditions:	A set of selected goods exists.

Title:	View shopping list
Actor:	Buying user
Description:	The user assembles a shopping list from all selected goods.
Pre-conditions:	A set of goods has been selected by the user.
Process:	
	1. The user triggers the view of a shopping list. 2. The set of selected goods is assembled in a shopping list arranged by shops.
Post-conditions:	The shopping list is saved.

Title:	Assemble shopping list
Actor:	Buying user
Description:	The user selects goods and assembles a shopping list.
Pre-conditions:	none
Process:	1. The user searches for goods and selects them. 2. The set of selected goods is assembled in a shopping list ordered by shops.
Post-conditions:	The shopping list is saved.

Figure 3: Selected use cases of shopping list app

Note that use case "Assemble shopping list" has use cases "Search for goods" and "View shopping list" as sub-use cases. In addition, two test case specifications for use case "Assemble shopping list (ASL)" are shown in Figure 4.

Name: TestASL_NoSelectedGoods
Input values:
no selection
Expected result:
an empty shopping list is viewed

Name: TestASL_SelectedGoodsOfDifferentShops
Input values:
selection of goods "Bio-Roggenbrot", "Nagellack (Rot)", "Pecorino", "Kirschsaft", "Rose (trocken)", and "Grüner Tee".
Expected result:
a shopping list is shown containing exactly the indicated goods ordered by shops. This means that the first good is assigned to "Bäckerei", the second to "Drogerie", the third to "Frische & Feinkost", the fourth and fifth to "Getränke", and the last one to "Tante Emma".

Figure 4: Selected test cases for shopping list app

We will see in Sections 6.3 and 6.4 how these use cases and test cases influence the cognitive model and the user tests.

6.2 Implemented Variants of the Shopping List App

Based on the given requirement specification, two variants of the shopping app have been implemented. Both variants allow users to choose products to buy. The chosen products are then added to a list. Menu depth differs between the two variants: Variant A has one menu level more than variant B. The first (overview) page of the app is the same for both versions and contains the buttons "shops" and "my list". If you click on "shops", then a list of seven shops (bakery, drugstore, deli, greengrocer, beverage store, stationery, and corner shop) appears in both variants.

Selecting one of the shops in variant B results in an alphabetically ordered list of the products available in that particular shop. For example, clicking on greengrocers all items that can be found in a greengrocers store are presented (apples, bananas, blueberries, cherries…). A click on a small checkbox on the right of the product selects it. In variant A, the shops have subcategories. When selecting greengrocers, for example, it is presented with the subcategories exotic fruits, domestic fruits, tuber vegetables, herbs, seeds and nuts, mushrooms and salads. When selecting a subcategory, a list of products that can be found under this subcategory, appears. A click on a small checkbox in the right of the product selects it.

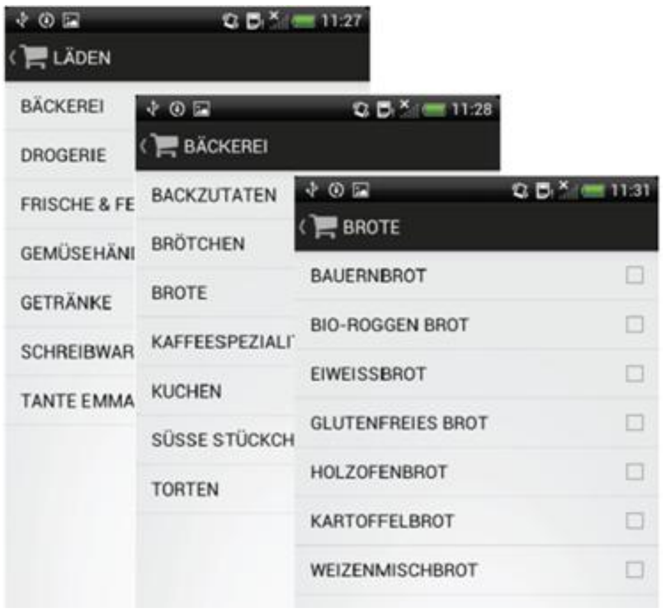

Figure 5: Screenshot of Shopping List App – Variant A (Variant B is missing the middle layer)

6.3 User Modeling of the Shopping List App

The structure of the shopping list app was designed to be limited in complexity but to still be expansive enough to test important aspects of existing genuine mobile applications with it.

The use of the application is connected to the overall goal of attaining a complete shopping list. This motivational background can be used to formulate goals and sub-goals in ACT-R. These goals and sub-goals are subsequently retrieved by the goal buffer and "drive" the model by helping to select the next production rules at each step of the cognitive process.

The use case descriptions in the requirement specification are used to hint to goals and sub-goals. The overarching use case "Assemble shopping List" provides for the main goal of attaining a saved shopping list at the end of the app use. It also specifies the sub-goals of searching and selecting goods. The use case "Search for Goods" provides some sub-goals concerning a repeated selection process of items.

The model itself starts off with a production that contains the overall goal of attaining a shopping list in the goal buffer. At the same time, the model "sees" the starting screen and its elements through its visual module. To use this information by other modules, however, its elements are called into the visual buffer. On the first use, the model first searches through available buttons and chooses one to click. Then, a production transfers the location of the chosen button to the motor module which in turn, initiates a touch on that button. *ACT-Droid* delivers this motor information to the shopping list app which in turn, switches to the next screen and provides the model (again via *ACT-Droid*) with the new visual information.

When starting to use a smart phone app, behavioural and cognitive routines might be applied at first that were successful in using other apps or even computer programs. With continued use, new productions are created, and existing productions are altered and merged. Successful productions, i.e. those productions that lead to the successful accomplishment of a goal or sub goal, rise in importance and are used more frequently over time. All of these processes reflect the growing experience and skill in using the app with continued use.

Another aspect of the models deals with knowledge about the products in the shopping list app. This knowledge can play an important role during the app usage. This knowledge exists prior to use and is changed throughout use. In general, we assume a certain knowledge base, but some specific knowledge differs between participants. Behaviour of the participants is affected, for example, by their pre-knowledge about purchasable items in the shopping list app. Therefore, in the experimental setup we ask participants via questionnaire, whether they recognized certain rare items. Depending on their answer, we alter the number and composition of chunks containing these items in the declarative memory.

The activation of ACT-R mechanism allows us to model how the knowledge structure changes during the use of the shopping app. This is true both regarding explicit knowledge that was acquired prior to use and regarding knowledge that is acquired during use (such as where certain products are to be found within the app).

The app also features general affordances and symbols like checkboxes and the back-button. Certain graphical elements (like the separation of items on the shopping list or the varying similarity of menu structures in the app) can trigger general experience-based expectations as well as universal ones (here specifically "Gestalt" laws). The modelling of expectations is discussed in more detail in Section 3.3.

6.4 Experimental Validation - General Approach

When modelling human behaviour, a common general approach is used to ensure high standards of the models.

In the first phase, one model or several alternative models are created. The building process of these models can be informed by exploratory experiments and/or by theories in the relevant fields of psychology. Next, these models are run. They interact with a simulated environment that mirrors the important features of the environment that real participants would be in. All model runs produce data about the predicted behaviour of humans in this environment (in our case when using a mobile app). Produced data includes decisions and reaction times.

In the next phase, the experiment is conducted with human participants and dependent variables are collected. These experiments can be informed by the test cases, as they provide possible user tasks and the desired end results. Upon completion of the experiment, the human data is then compared to model predictions. If a model predicts actual behaviour well, it is assumed that both, the mechanisms used in the model and their implementation reflect human cognition well. Further prediction-experimentation cycles can be used to refine the model or to make it applicable in a wider array of situations. As a practical application, the model can then be used to predict human behaviour in novel (but similar) situations without the need for constant experimentation with human participants.

A future goal of the modelling effort is to not only evaluate specific variants of apps but to get a better understanding how specific structural features (like menu depth) affect usability over a wide range of applications.

6.5 Experimental Validation - Menu Depth in the Shopping App

As a first experimental demonstration of our concept we decided to explore the effects of menu depth on usability. In a series of initial experiments, we presented participants the two different versions of the shopping app that are described in Section 5.1 and that vary in their menu depth.

The first tests suggest interesting effects of both changing menu depths and swapping between both variants (Prezenski & Rußwinkel, 2014). Using the version with the shallower menu is slightly faster than its counterpart, but the benefit decreases as user experience increases. Version update from the deeper menu to the shallow one has additional time cost in the beginning, whereas switching the other way has not. Large-scale testing and modeling of the effects of menu depth and other implementation choices within the shopping app are planned for the immediate future.

7. CONCLUSION

In this paper, we present a new approach of evaluating the usability of mobile applications by cognitive models that are inspired by the requirement specification for apps. We demonstrated some specific steps and principles and illustrated them at an example of a simple shopping list app.

Our approach allows to evaluate the usability of mobile app variants in an early stage of development. It also requires less user testing than traditional approaches. Cognitive modelling makes it possible to take changing knowledge and expectations of the user during different use phases into account. It therefore enables us to predict usability criteria in a broad context.

However, the approach needs some preparations. It requires the development and validation of cognitive models, a potentially lengthy process that requires capable modelers along the way. Our project therefore constitutes a long-term approach, as the introduction of cognitive modelling will be effortful in the beginning but opens up new and advantageous ways of usability evaluation in the future.

One of the next steps will be to improve ecological validity of our modelling approaches by creating models predicting behaviour and usability under smart phone-specific, typical use situations. This can include the modelling of disruptions of use, mental load during use and changes between app versions. We also plan to develop cognitive models that specifically model users of old age as well as technical novices and experts. Another promising research direction is to model individual differences between users. Some users are completely new to smart phone usage; they have no prior knowledge how to use the interface and have to transfer knowledge from other mobile software. Other users are less precise in their motor movements. And other users again are technically very experienced and try all possible options. All these different users could be captured by specific kinds of cognitive models. In general, it would be helpful to evaluate applications accounting for the different user groups that the applications are designed for.

REFERENCES

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C. & Qin, Y. (2004). *An integrated theory of the mind.* Psychological Review, 2004, 111 (4), 1036–1060.

Arthur, W., Bennett, W., Stanush, P. L., McNelly, T.L. (1998). *Factors that influence skill decay and retention: a quantitative review and analysis.* Hum. Perform. 11:79–86

Beck, K. (2003). *Test-Driven Development: by Example*. Addison-Wesley-Vaseem

Bothel, D. (2007). *ACT-R 6.0 reference manual.* Working draft. From the ACT-R web site. <act-r.psy.cmu.edu/actr6/reference-manual.pdf>

Brunswik, E., & Kamiya, J. (1953). *Ecological cue-validity of 'proximity' and of other Gestalt factors.* American Journal of Psychology, 66, 20-32.

Büttner, P. (2010). *"Hello Java!" Linking ACT-R 6 with a Java simulation.* In: Proceedings of the International Conference on Cognitive Modeling (ICCM), 2010. Philadelphia.

Cockburn, A. (2002). *Agile Software Development*. Addison-Wesley.

Friston, K. & Kiebel, S. (2009a). *Predictive coding under the free-energy principle.* Philosophical Transactions of the Royal Society, Biological Sciences, 364, 1211-1221.

Kronbauer, A. H., Santos, C. A. S., & Vieira, V. (2012). *Smartphone applications usability evaluation: a hybrid model and its implementation*. In: Proceedings of the 4th international conference on Human-Centered Software Engineering, ser. HCSE'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 146–163. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34347-69

Kuparinen, L., Silvennoinen, J. & Isomäki, H. (2013) *Introducing Usability Heuristics for Mobile Map Applications.* In Proceedings of the 26th International Cartographic Conference (ICC 2013).

Lindner, S., Russwinkel, N. (2013). *Modeling of expectations and surprise in ACT-R*. Proceedings of the 12th International Conference on Cognitive Modeling, pp. 161 - 166. Available online: http://iccm-conference.org/2013-proceedings/papers/0027/index .html.

Möller, S., Engelbrecht, K.-P., & Schleicher, R. (2008). *Predicting the Quality and Usability of Spoken Dialogue Services*. Speech Commun., 50(8-9), 730–744. doi:10.1016/j.specom.2008.03.001

Prezenski, S., Rußwinkel, N. (2014). *Cognitive Modeling offers Explanations for Effects found in Usability Studies*. In submission.

Rusu, C., Roncagliolo, S., Rusu, V., & Collazos, C. (2011). *A Methodology to establish usability heuristics*. Proc. 4th International Conferences on Advances in Computer-Human Interactions (ACHI 2011), IARIA, pp. 59-62, ISBN: 978-1- 61208-003-1, 2011.

Teo, L., John, B. E., & Pirolli, P. (2007). *Towards a Tool for Predicting User Exploration*. CHI '07 Extended Abstracts on Human Factors in Computing Systems (pp. 2687–2692). New York, NY, USA: ACM. doi:10.1145/1240866.1241063

Zhang, S., Adipat, B (2005). *Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications*. International Journal of Human-Computer Interaction, 2005, 18 (3), 293–308.

Appendix A. USABILITY ATTRIBUTES

Table 1: Usability Attributes (adapted from Zhang & Adipat, 2005; table 2, page 304)

Usability Attribute	**Meaning**	**Measuring Variables**
Learnability	- ease of finishing a task the first time using an application - speed of improvement in performance levels	- time used to accomplish tasks at the first use - time spent on training users until reaching a level of satisfaction - amount of training - learning curve of several uses
Efficiency	- speed in accomplishing a task after having gained some experience in the mobile application	- task completion time - time used to finish given exercises - time spent on each screen
Memorability	- level of ease with which users can recall how to use an application after discontinuing its use for some time - ease of re-establishing the skill of using an application	- time, number of button clicks, pages, and steps used to finish tasks after not using applications for a period of time
Error	- number of mistakes that users make while using a mobile application - severity levels of mistakes - ease of correction for the user	- number of errors made (e.g., detour steps, deviating button clicks from the right path, wrong answers, percentage of correctly completed tasks)
Satisfaction	- attitude of users toward using a mobile application	- attitude of users toward applications after using them (e.g., level of difficulty, confidence, like/dislike)
Effectiveness	- completeness and accuracy of goal achievement - improvement in a new version of a mobile application	- comparison of user performance with a predefined level (e.g., finishing tasks in 9 minutes, using no more than 2 clicks) in terms of - speed - errors - number of steps - tasks solved in a time limit
Simplicity	- degree of comfort for accomplishing tasks - quality of menu structures as well as navigation design of mobile applications	- amount of effort to find a solution - numbers of menu levels that users have to go through to solve a task - numbers of button clicks and selections to reach a destination page - time used to search for a button to perform a specific function
Comprehensibility	- ease of understandability of the content	- reading speed - percentage of correct answers in a predefined test