ANTETYPE-PM: An Integrated Approach to Model-Based Evaluation of Interactive Prototypes

Dieter Wallach[1,2(✉)], Sven Fackert[1], Jan Conrad[2], and Toni Steimle[1]

[1] Ergosign GmbH, Saarbrücken, Germany
dieter.wallach@ergosign.de
[2] University of Applied Sciences Kaiserslautern, Kaiserslautern, Germany

Abstract. In this paper, we present an integrated approach to model-based evaluations of interactive prototypes. By combining a state-of-the-art cognitive architecture, ACT-R, with an elaborated prototyping tool, ANTETYPE, we enable UX designers without modeling experience to derive quantitative performance predictions for interactive tasks. Using ANTETYPE-PM, an interface designer creates an interactive prototype and demonstrates the action sequences to complete relevant application scenarios using the *monitoring* and/or *instruction mode* of ANTETYPE-PM. The system learns the interaction paths and predicts the interaction times over trials using ACT-R's symbolic and subsymbolic (i.e. statistical) learning mechanisms. To illustrate the working of ANTETYPE-PM, an example is provided and contrasted with empirical data.

Keywords: Model-based evaluation · Usability evaluation
Cognitive architectures · Quantitative performance prediction

1 Introduction

In this paper we report our progress on an integrated approach to model-based evaluation combining an advanced prototyping tool with a state-of-the-art cognitive architecture to derive quantitative performance predictions for interactive tasks. After first introducing the prototyping tool ANTETYPE, we discuss some limitations of empirical usability testing to motivate the use of model-based evaluation. Given that our approach is based on the cognitive architecture ACT-R, we provide a brief overview of this framework and compare related models to model-based evaluation. Finally, we illustrate the use of our integrated tool, ANTETYPE-PM, and present an example for its application in a real-world task.

1.1 User Interface Prototyping with ANTETYPE

ANTETYPE is a sophisticated design tool to create interactive prototypes for desktop, mobile and web-based applications. ANTETYPE was designed to support a seamless transition from early wireframes defining the layout of an interface, over the creation of visual design alternatives to the creation of complex interactive prototypes. Its layout

A. Marcus and W. Wang (Eds.): DUXU 2018, LNCS 10918, pp. 560–572, 2018.
https://doi.org/10.1007/978-3-319-91797-9_40

engine, advanced features for visual design and elaborated functionality for defining interactions, transitions and animations make ANTETYPE a highly flexible and powerful tool for developing sophisticated interactive prototypes.

Requirements for designing interactive prototypes have become increasingly complex: ANTETYPE's layout engine supports the creation of *responsive designs* that fluidly adapt to different screen dimensions and provide helpful layout declarations for development. Complementing the layout engine, elaborated *visual design functionality* enables designers to work with ANTETYPE from early conceptual wireframes to comprehensive high-fidelity prototypes. Enhancing ANTETYPE prototypes with *transitions and animations* guide users in understanding the consequences of their actions without requiring designers to switch to another tool.

Iteratively modifying and refining a prototype can become a very time-consuming task. Differentiating local and global elements, ANTETYPE's *widget concept* allows the automatic propagation of changes to the visual appearance and behavior of an interface widget—relieving designers from the tedious manual updating of all instances of the widget spread over potentially numerous screens of a prototype. The definition of *state-dependant, hierarchically nested widgets* finally enables the creation of powerful, reusable interface elements to meet the challenges of complex prototypes.

Over the course of a project, ANTETYPE is typically used to iteratively explore and evaluate ongoing design work with stakeholders. A particularly important application of ANTETYPE prototypes is their use in empirical usability tests involving representative participants.

2 Limitations of Empirical Usability Tests

Empirical usability testing is regarded as a hallmark of human-centered design approaches. While usability testing of interactive prototypes in a usability lab allows for the elicitation of valuable feedback from prospective users, several restrictions need to be considered—especially when being interested in *quantitative* performance indicators of an interface.

Predicting the time for the *routine performance* of users dealing with relevant key scenarios, for example, is severely limited when a novel interface is presented in a usability test: participants first need to learn how to operate the new interface. Depending on its complexity, a level of routine interaction will only be reached after significant amounts of practice with the user interface. The time requirements to arrive at such skilled performance might well be outside the practical scope of most usability tests of complex applications. It is, to give an obvious example, not trivial to isolate quantitative measures of *efficiency* from *learnability* metrics of an interface in empirical usability tests—at least for more complex applications.

Instructions in most usability tests typically do not explicitly require participants to complete tasks *as fast as they can.* Given that participants know that their behavior is recorded in a usability test, it is reasonable to assume that their efforts will at least be influenced by the goal to show an *effective, error-free* behavior—which stands in obvious conflict with a focus on time-efficient interaction.

Referring to *quantitative* performance predictions in general, the typical setting and instruction in an empirical usability test—that is often mainly targeted at the elicitation of *qualitative* insights—poses additional complications. To name just one: in most empirical usability tests, participants are asked to *think aloud* while working on test scenarios. Empirical studies (Steimle and Wallach 2018) have shown that concurrent thinking aloud results in significant increases in the time requirements for completing a task, contradicting the goal of valid performance measurement. For what it's worth, the instruction to think aloud can hardly be regarded as a representative characterization of typical working situations.

While we agree that empirical usability testing is an irreplaceable evaluation method, the aforementioned challenges need to be carefully considered when following the goal of deriving quantitative performance predictions for the routine work with an interactive system. Extended practice times with an interface to overcome learning effects in usability tests help to arrive at valid performance predictions for routine interactions. The additional effort, however, inevitably results in even higher costs for conducting empirical usability evaluations—worsening the cost-benefit ratio that inspired Nielsen (1989) to coin empirical usability tests a "deluxe method".

3 Model-Based Evaluation Using Act-R

Model-based evaluations, i.e. using artificial *cognitive crash-test dummies* to simulate user behavior with interactive systems provide a promising and cost-effective approach to overcome at least some of the challenges discussed in the previous section. In the following we will spell out the advantages of combining a powerful prototyping tool like Antetype with model-based evaluation on the grounds of a so-called *cognitive architecture* like Act-R (see Anderson et al. 2004; Laird et al. 2017).

Integrating a model-based evaluation module into Antetype allows quantitative performance predictions for interaction scenarios by simply running "simulated users" with an interface prototype: UX designers can explore the quantitative implications of their design decisions in the very same tool that they are using for the design of user interfaces. In the resulting tool, that we denote Antetype-Pm, UX designers can analyze and compare the performance of different design options for relevant interaction paths through key scenarios. Quantitative performance evaluations do not need to be postponed until the design of a user interface is largely completed—the temporal implications of an interaction path can be explored whenever the part of the user interface to support the respective path is designed, allowing a continuous inspection of performance parameters during the iterative course of interface design activities. Before illustrating how to work with Antetype-Pm, we introduce the Act-R cognitive architecture that forms the theoretical framework underlying the proposed model. After that, we will contrast it with alternative modeling approaches.

The Act-R Cognitive Architecture. A cognitive architecture embodies a comprehensive scientific hypothesis about the structures and mechanisms of the human cognitive system that can be regarded as (relatively) invariant over time. Act-R provides an integrative theoretical framework for explaining and predicting human behavior and

is instantiated as a theoretically justified, implemented software system allowing for the computational modeling of a wide range of phenomena. Authors have been discussing the potential benefits of applying cognitive architectures in the domain of Human-Computer Interaction for more than twenty years (Kieras 2007). Their broader use in HCI, however, has been fostered by the extension of architectures with perceptual-motor modules that allow the modeling of perception and interaction with an external world.

ACT-R comprises *symbolic* learning mechanisms to acquire new if-then-rules (so-called *productions)* in procedural memory and declarative facts (so-called *chunks)* in declarative memory, as well as *subsymbolic* mechanism for the statistical tuning of their appropriate application. ACT-R can *perceive* and *manipulate* the external world that is defined, in the case of ANTETYPE-PM, by an interface prototype. Figure 1 depicts a structural overview of ACT-R and its interplay within ANTETYPE-PM.

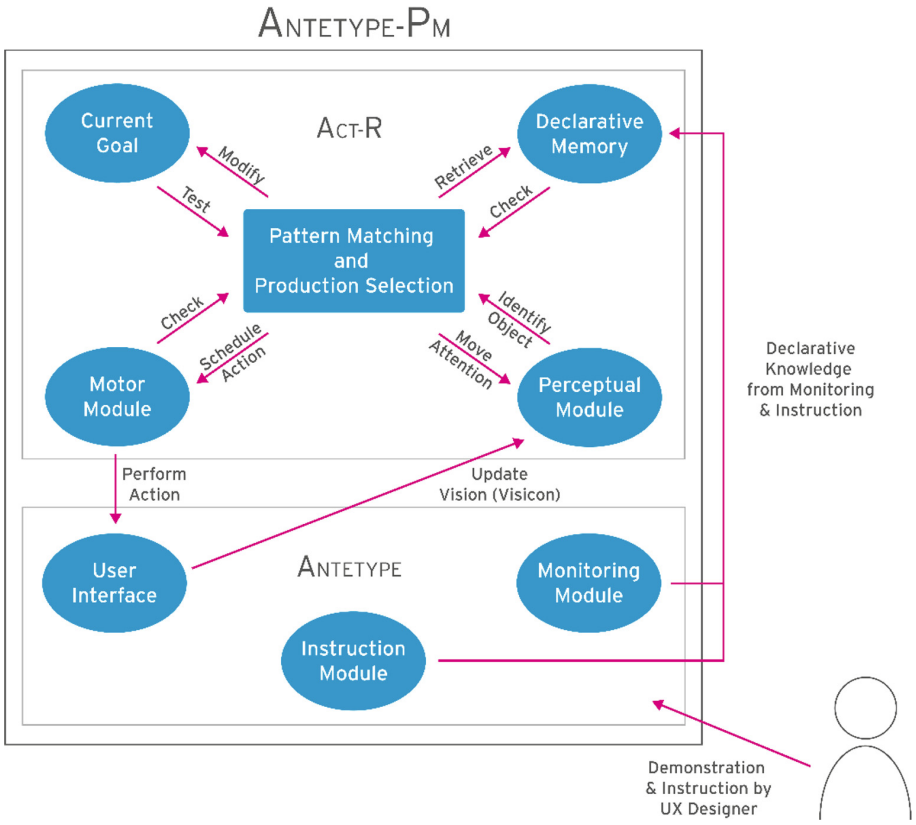

Fig. 1. Interplay of ANTETYPE and the cognitive architecture ACT-R in ANTETYPE-PM

ACT-R provides a theoretical framework that comprises empirically motivated constraints (see Kieras 2007) on the details of models to be written within this framework.

In ANTETYPE-PM we have instantiated a core ACT-R model that provides a set of operators:

- to *perceive* interface objects like buttons or labels on ANTETYPE prototypes,
- to *interact* with ANTETYPE's interface controls,
- to encode declarative and procedural knowledge for *monitoring* and *learning* actions that the model observes,
- to represent basic user-provided *instructions* on the conditional application of actions.

Before illustrating how to work with ANTETYPE-PM, we briefly describe related work in the field.

4 Creating Cognitive Models

Historically, different paths have been taken to arrive at cognitive models for predicting performance in Human-Computer Interaction scenarios. These paths differ, among other criteria, in the required effort for model development, necessary knowledge about the theoretical Cognitive Science underpinnings of a model and prerequisite modeling skills, as well as in the technical support to create models. A brief glance at cognitive modeling in the field of HCI unveils roughly the following approaches:

- *building cognitive models by hand*, which technically requires nothing more than just a pen and piece of paper for establishing an interaction sequence (typically derived from task analysis) and calculating time parameters of hypothetical human cognitive, perceptual and motor operators (see Sect. 4.1 of this paper);
- *using software tools for (more or less) automating* the aforementioned process of calculating the temporal effects of assumed operator sequences underlying the interaction path to be predicted (see Sect. 4.1);
- *utilizing comprehensive cognitive modeling tools* to assist a modeler in both the development and the simulation of a performance model (see Sect. 4.2).

The first two approaches presuppose a basic understanding of Cognitive Science theory to achieve valid results and typically require a serious amount of time for model development. The final approach may seem most promising because of its support in model creation and automating the generation of its predictions. However, without getting deep into the theoretical and syntactic details of the respective modeling tool or cognitive architecture, model development will be limited to very simple tasks. In the next section, we will look at the approaches in more detail.

4.1 Hand-Crafted and Semi-automatic Models

Initially introduced by Card et al. (1980) the *Keystroke-Level Model* (KLM) provides a framework for developing models to predict execution times for expert users completing routine tasks with a given interactive system. The KLM is an example of a (simplified) architecture that forms a foundation for model-based evaluations of design suggestions. Using this approach, a modeler bases his work on a task analysis before then going through the task step by step to translate each identified step into operators declared in the KLM. Each of these operators describes a subpart of the task at the keystroke-level and is associated with a time value. The execution time for an entire task is then calculated as the sum of the individual operators (in the simple case of sequential actions). While modeling is quite straightforward as long as operators only represent motor actions, the situation becomes complicated with cognitive processes. KLM, focusing on the keystroke-level, takes a rather abstract stance on cognitive processes and proposes the insertion of *mental operators*. Mental operators do not have fixed associated time values, but require a modeler to estimate the duration of steps to represent a user's *thinking processes*.

Apart from the time-consuming process of manually analyzing a task and having to write down each step when calculating execution times by hand, the application of KLM is best suited to what its name suggests: modeling at the observable level of key strokes. While it may be necessary to make decisions about underlying cognitive processes at various stages of modeling more complex interactions, KLM fails to provide much guidance on their duration.

Initial steps toward automating the development and simulation of cognitive models have been taken: Computer programs like GLEAN (Kieras et al. 1995) or Cogulator ("Cogulator: A Cognitive Calculator", 2018) provide a graphical user interface for writing cognitive models. Using them, modelers no longer need to manually calculate individual time predictions but are supported by software to compute and output modeling results. While such systems improve the efficiency of developing models and the determination of their temporal consequences, they still lack appropriate support at the cognitive level.

With regard to deriving quantitative predictions, running an ACT-R model can also be classified as a semi-automated way of model-based evaluation. In contrast to the aforementioned approaches, however, the proposed structures and processes of the Act-R cognitive architecture provide a detailed theoretical framework at the cognitive level. This advantage does not come without costs: cognitive architectures have notoriously been known to be hard to learn. Model development in a cognitive architecture is outside the scope for most non-specialists and it is hard to imagine UX designers to include cognitive modeling with ACT-R in their daily toolbox. Automated generation of evaluation models, discussed in the next section, offer a promising step to reduce modeling complexity, thus making cognitive modeling accessible to a larger range of users.

4.2 Automated Generation of Models

The obvious next challenge after automating the simulation of cognitive models is to automate model creation *per se*. A well-known endeavor in this direction is the so-called CogTool – proposed by Bonnie E. John and colleagues. CogTool consists of tools that "allow a UI designer to mock up an interface as an HTML storyboard, demonstrate a task on that storyboard, and automatically produce a consistent, correct Klm of that task that runs in the Act-R cognitive architecture (Anderson and Lebiere 1998) to produce predictions of skilled performance time" (John et al. 2004, p. 462). The advantages of such an approach are obvious: a designer requires no cognitive modeling skills because the model is created automatically while the designer demonstrates the task.

While the reduction in effort and the cost savings using CogTool compared to plain Act-R are without doubt impressive, this approach is still quite limited to the boundaries of Klm. I.e. a model-based evaluation using CogTool still focuses mainly on predicting the total execution *times* for tasks performed by *skilled users* under the assumption that *no errors* occur. In addition, initial versions of CogTool failed to make use of the full potential of the underlying Act-R architecture: cognitive processes were still reduced to the insertion of *mental operators* that only pause model execution for estimated amounts of time (John et al. 2004).

Although a more recent development version of CogTool, called CogTool Explorer (Teo et al. 2012), paved the way for modeling goal-directed user exploration on the cognitive level, the following reasons motivated the need for a tool like Antetype-Pm:

- development of CogTool went into quiescence: at the time of writing this paper, the latest stable release of CogTool dates back to December 2013 using deprecated versions of ACT-R and Java;
- in order to use CogTool, design prototypes need to either be rebuilt within CogTool or imported as HTML files, doubling the amount of work for designers;
- advanced modelers cannot easily elaborate or extend an Act-R model generated by CogTool, giving away options for its refinement.

While the last argument is not without importance, a core objective for developing Antetype-Pm was to create a valuable tool for UX designers without requiring users to have a comprehensive cognitive modeling background. In the next section we present an example for using Antetype-Pm.

5 Using Antetype-Pm: An Example

In order to explain how to apply Antetype-Pm in real world tasks, we present a first example that was targeted at predicting user performance with a new interface for a software-controlled riveting machine for airplanes. Before going into the details of using Antetype-Pm, the following list provides an overview of the necessary working steps, with the final step being optional:

1. Create an interactive prototype using ANTETYPE-PM.
2. Explain the task to ANTETYPE-PM using demonstration and/or instruction.
3. Let ANTETYPE-PM generate the cognitive model for the task.
4. Run an arbitrary number of simulated users on the task.
5. Inspect the predictions and analyze their implications.
6. (Revise your design solution and run the simulation again.)

5.1 Creating Interactive ANTETYPE-PM Prototypes

ANTETYPE-PM interacts directly with design prototypes and thus requires a design proposal to exist that takes the form of an interactive ANTETYPE prototype. As discussed at the beginning of this paper, ANTETYPE assists UX designers throughout the entire design process from wireframes to high-fidelity prototypes. A model-based evaluation using ANTETYPE-PM starts off with the creation of an interactive, low-fidelity prototype (i.e. linked wireframes) of the intended system. This starting point lays the foundation for an iterative design process where the early detection of potential usability shortcomings may prevent more expensive alterations of visually elaborated, high-fidelity solutions. Creating the prototype for ANTETYPE-PM does not differ from the usual practice of using ANTETYPE—except that a designer needs to denote which interface objects the underlying ACT-R model can *perceive*. Figure 2 depicts an ANTETYPE prototype of the software-based riveting machine. The interface was designed by an external company to assist users in operating large riveting machines in a complex context of use. Supporting an *efficient* and *error-free* interaction is a mandatory requirement for the machine's user interface. The next section focuses on how to predict the performance of the interface using ANTETYPE-PM.

5.2 Explain the Task Using Demonstration or Instruction

In addition to an interactive prototype that is to be evaluated, a cognitive model in ANTETYPE-PM needs relevant knowledge about the task and its workflow, as well as *generic, task-independent* knowledge about the operation of user interfaces in general. ANTETYPE-PM thus comes with a default set of generic productions that allow interaction with user interfaces, as well as knowledge to encode observed actions. *Task knowledge* and the sequence of involved interaction steps, however, need to be provided to ANTETYPE-PM by the user.

As discussed in Sect. 4.2, ANTETYPE-PM belongs to the category of tools that automate the development of cognitive models. To provide the necessary task knowledge, a user demonstrates the interaction path for a task scenario while ANTETYPE-PM is *observing* and *learning* the respective interaction sequence. There are, of course, dependencies between task steps and decisions that cannot just be derived from observing a designer completing the task with a prototype. An interaction with a check box, for example, may depend on its current state (is it checked or not?), or the next interaction step may depend on the outcome of a comparison of values that are provided by the interface. To resolve the rationale behind such decisions, relevant instructions can be entered into ANTETYPE-PM's instruction module, resulting in the creation of task-specific knowledge in ACT-R's declarative memory.

5.3 Generation of the Cognitive Model

Based on the information collected from demonstration and/or instruction, the cognitive model is generated. Declarative facts are encoded as ACT-R chunks and stored in the declarative memory of the model, whereas procedural knowledge is represented as productions in ACT-R.

The following code shows an example of a chunk type declaration used to store task-specific information about a simple button click interaction. It contains a name for the chunk type, an identifier, the text of the button that should be clicked and the kind of button on the mouse that should be pressed. The name of the chunk type does not have any impact on the model execution. Its main purpose is to improve the readability and clarity of the model. The interaction identifier is used to keep track of the position within an interaction sequence. The last information of this chunk type determines the kind of button that should be clicked. This may be the left or right button of the mouse.

```
(chunk-type button-click-interaction
  interaction-id
  button-text
  mouse-button)
```

This declaration is used to instantiate chunks containing the aforementioned information and is only one example for how task-specific information is delivered to the model. As soon as the model has been generated it is ready for simulating user performance, which is described in the next section.

5.4 Run a Simulation

Running a simulation in ANTETYPE-PM is a straightforward process. A designer just needs to select a task scenario and determine the number of simulation trials that ANTETYPE-PM should work on this scenario. A simulation can be run in *real time* or in *speed mode*. The former will visualize the simulation by highlighting the visual attention and eye movements of a simulated user, as well as showing the respective mouse movements and clicks. A user's visual attention (yellow circle in Fig. 2) and eye movements (blue circle in Fig. 2) are simulated using an ACT-R module called EMMA, developed by Salvucci (2000). Running the simulation in *speed mode* will provide faster results without visualization. The results of the respective modes do not differ, of course.

At this point we need to differentiate between different kinds of potential ANTETYPE-PM users. A typical UX designer without a comprehensive background in Cognitive Science will be able to perform a model-based evaluation by simply following the aforementioned steps. Users with an advanced understanding of ACT-R, however, can inspect and enhance a model's symbolic (declarative and procedural) knowledge or fine-tune its subsymbolic parameters in order to explore the model in an extended scope.

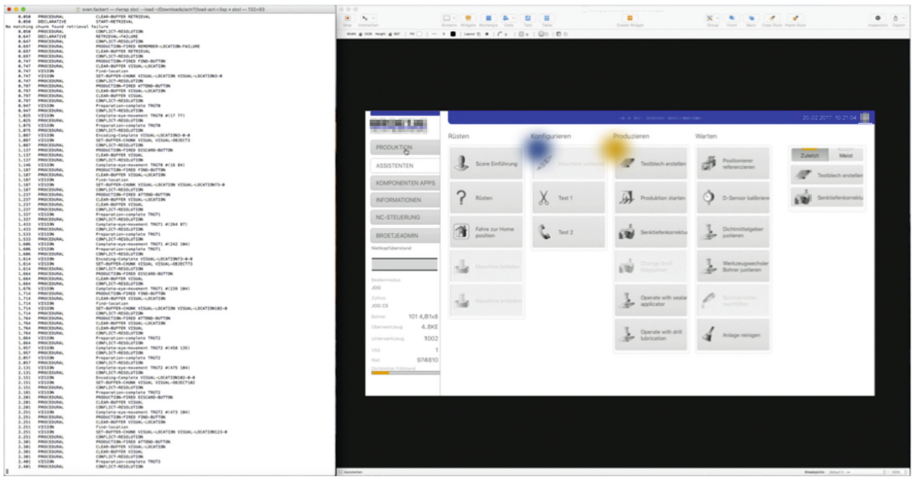

Fig. 2. ANTETYPE-PM showing the ACT-R trace (left) and interaction with a prototype (right) (Color figure online)

5.5 Inspect and Interpret Predictions

After the specified number of simulation runs is completed, results are presented to the user of ANTETYPE-PM. Let us assume that a user is interested in a prediction of the time that it takes for a novice user to achieve a specific task performance with a prototype. Running the simulation for multiple trials will output resulting learning curves that illustrate the transition from novice to more advanced users in terms of their respective time requirements for completing the task. A designer can then utilize this data along with the observations from watching the visualization of the simulation to understanding the time spent in different subparts of the task. Based on an interpretation of these findings, a design solution can then be revised, evaluated again and finally compared to previous solutions.

Repeating this process until the design solution meets a level of desired performance provides designers with constant feedback to derive informed design decisions when iteratively improving an interface. Revising a prototype is straightforward with ANTETYPE-PM and usually does not even require the modeler to configure the task, i.e. to demonstrate/instruct the interaction scenario, again. I.e., altering button positions, sizes, colors or the arrangements of elements will not require the cognitive model to be changed — the implications of the interface changes, however, will be predicted by the model. This enables designers to rapidly try out and compare design variations and to discover potential deficiencies of an interface at a very early stage.

6 Validating the ANTETYPE-PM Model

While it is tempting in the context of writing a paper to present a fine-tuned model that provides impressive fits to available data, we deliberately decided to refrain from such an endeavor. Instead, we report the bare performance of the model without adjusting

any subsymbolic ACT-R parameter (e.g. leaving all ACT-R parameters at their default value), not making use of ANTETYPE-PM's instruction module, as well as not adjusting or handcrafting the generated ACT-R model. Formally, this comes down to a *zero parameter model* — and that is exactly what is required given the goal of providing a tool for UX designers without cognitive modelling, or, more specifically, ACT-R experience.

In order to collect data for an interface proposal for the riveting machine modeled in ANTETYPE-PM, we ran a study with 15 participants (8 male, 7 female; mean age: 28.5, SD: 3.8). All participants were very experienced users working professionally with computers in their daily work. After introducing them to the steps of a task scenario with the user interface for the riveting machine, each participant was given a practice trial to clarify questions. After that, all participants were asked to repeat the task scenario for a total of 14 trials.

ANTETPYE-PM was trained for the task scenario by a UX designer who demonstrated the interaction steps while the model observed the respective actions. Then the model was run for a single trial to check its ability to reproduce the interaction path, before simulating a total of 14 runs through the task scenario. I.e., we did not only maintain all ACT-R parameters at their standard values, but also used the very same number of trials that the participants were required to do. Figure 3 shows the mean time trajectories of the participants and the runs of ACT-R over the 14 trials of the task scenario. While the model is clearly significantly slower than the participants in this task, a comparison of the learning trajectories shows an impressive fit with an r^2 of .97 (see Fig. 4).

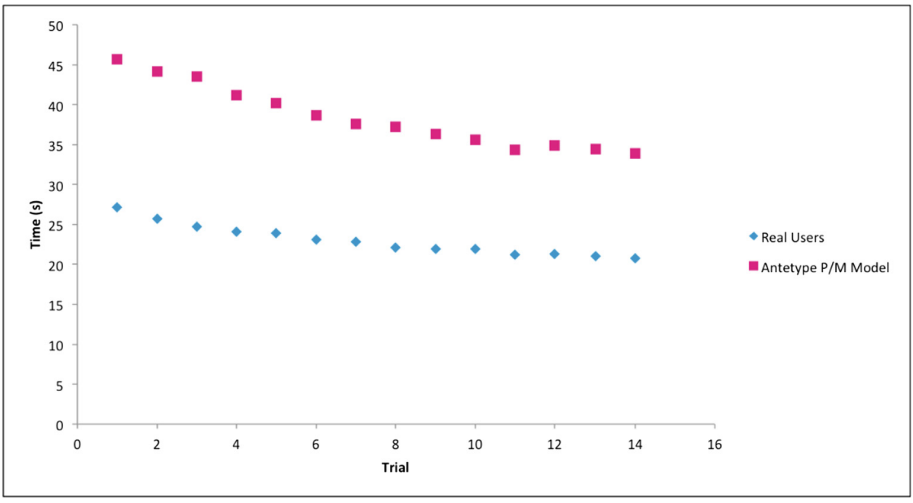

Fig. 3. Performance of the ANTETYPE-PM model in comparison to empirically obtained user performance

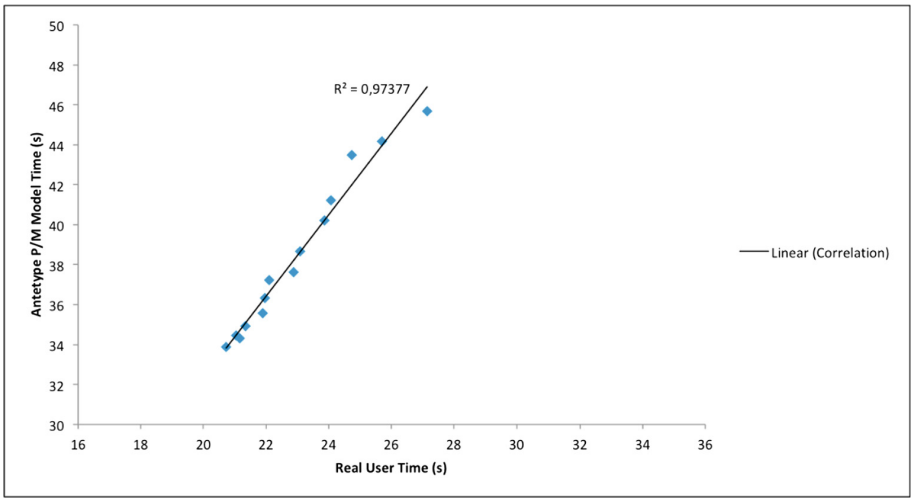

Fig. 4. Correlation between the ANTETYPE-PM model and participant data

While we of course need to explore the model in a broad range of tasks to appropriately judge its practical value, we regard the obtained initial results to be encouraging. Adjusting ACT-R parameters would of course result in a much better correspondence of the absolute values of the model and the empirical data. This, however, would not bring us closer to attaining our goal of developing a tool that is of practical significance. Post-hoc data fitting, although potentially exiting for cognitive modelers, is not of practical interest for a UX designer. Instead, receiving results to derive informed design decisions are needed to speed up iterative development — and such design decisions are to be taken *before* data with a resulting user interface can be collected. This of course does not exclude the exploration of ACT-R parameters that turn out to be more appropriate than the default values in Human-Computer Interaction scenarios.

7 Summary

In this paper we summarized our initial progress on developing a model-based evaluation approach that is combined with an advanced prototyping tool. While an interface created with the prototyping tool provides the external system environment for a human user, a generated model allows UX designers to simulate *artificial users* interacting with this interface. Focusing mainly on UX designers with no or little modeling experience, the resulting system, ANTETYPE-PM, comprises the ACT-R cognitive architecture and allows its use to predict quantitative usability performance parameters. Next research steps will include more detailed analyses to characterize the set of interactive tasks for which the approach is appropriate. While we are currently focusing on quantitatively predicting task times and learning curves with ANTETYPE-PM, initial steps towards the prediction of errors have been undertaken.

Acknowledgements. The authors are thankful to the German Federal Ministry of Education and Research for funding the research reported in the paper in the SiBed project (support code 01IS16037A).

References

Steimle, T., Wallach, D.: Collaborative UX Design. dPunkt, Heidelberg (2018)

Nielsen, J.: Usability engineering at a discount. In: Salvendy, G., Smith, M.J. (eds.) Proceedings of the Third International Conference on Human-Computer Interaction on Designing and Using Human-Computer Interfaces and Knowledge Based Systems, 2nd edn, pp. 394–401. Elsevier, Boston (1989)

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

Laird, J.E., Lebiere, C., Rosenbloom, P.S.: A standard model of the mind: toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. AI Mag. **38**(4), 13–26 (2017)

Kieras, D.E.: Model-based evaluation. In: Jacko, J., Sears, A. (eds.) The Human-Computer Interaction Handbook, 2nd edn, pp. 1139–1151. Lawrence Erlbaum Associates, Mahwah (2007)

Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Commun. ACM **23**(7), 396–410 (1980)

Kieras, D.E., Wood, S.D., Abotel, K., Hornof, A.: GLEAN: a computer-based tool for rapid GOMS model usability evaluation of user interface designs. In: UIST 1995 Proceedings of the 8th annual ACM Symposium on User Interface and Software Technology, pp. 91–100. ACM, New York (1995)

Cogulator: A Cognitive Calculator. http://cogulator.io/. Accessed 08 Feb 2018

Anderson, J.R., Lebiere, C.: The Atomic Components of Thought. Erlbaum, Mahwah (1998)

John, B.E., Prevas, K., Salvucci, D.D., Koedinger, K.: Predictive human performance modeling made easy. In: CHI 2004 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 455–462. ACM, New York (2004)

Teo, L., John, B.E., Blackmon, M.H.: CogTool-Explorer: a model of goal-directed user exploration that considers information layout. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2479–2488. ACM, New York (2012)

Salvucci, D.D.: A model of eye movements and visual attention. In: Proceedings of the International Conference on Cognitive Modeling, pp. 252–259. Universal Press, Veenendaal (2000)