Memory for the Random: A Simulation of Computer Program Recall

Fernand Gobet (fgobet@liv.ac.uk)
Department of Psychological Sciences, University of Liverpool,
Bedford Street South, Liverpool, L69 3BX, UK
Iain Oliver (iso@psyc.nott.ac.uk)
School of Psychology
University Park, University of Nottingham
Nottingham NG7 2RD, UK

Abstract

Contrary to a widely held belief, experts recall random material better than non-experts. This phenomenon, predicted by the CHREST computational model, was first established with chess players. Recently, it has been shown through a meta-analysis that it generalises to nearly all domains where the effect has been tested. In this paper, we carry out computer simulations to test whether the mechanism postulated with chess experts – the acquisition and use of a large number of chunks – also applies to computer programming experts. The results show that a simplified version of CHREST (without the learning and use of high-level schemata known as templates) broadly captures the skill effect with scrambled programs. However, it fails to account for the differences found in humans between different types of randomisation. To account for these differences, additional mechanisms are necessary that use semantic processing.

Keywords: chunk; computer programming; expertise; memory recall; random material

Introduction

Computer programming involves a variety of skills: the ability to understand programs written by others, to design, write and debug one's own programs, and to use problem-solving strategies to turn a set of constraints and desiderata into a correct and running program. Several general theories have been proposed to account for these abilities. Some authors (e.g., Adelson, 1981; McKeithen, Reitman, Rueter & Hirtle, 1981; Ye & Salvendy, 1994) have proposed that semantic knowledge plays an essential role. Others have proposed that expertise in programming, like in other domains, stems from the acquisition of a large number of perceptual chunks, which are the building blocks on which later semantic and procedural knowledge is constructed (e.g., Chase & Simon, 1973; Simon & Gobet, 2000).

Using a variety of chess-related tasks including a recall task, Chase and Simon (1973) gathered good evidence for the psychological reality of perceptual chunks. In addition, they found that there was a massive skill effect for the recall of positions taken from Masters' games, but that this effect disappeared with random positions. It was later shown that chess Masters keep a small, but reliable, superiority with random positions (Gobet & Simon, 1996). This result was actually predicted by CHREST (Chunk Hierarchy and REtrieval STructures), a computer model based on the idea of chunking (De Groot & Gobet, 1996; Gobet, 1993). The reason is simple: an expert, who has acquired more chunks than a weak player, is more likely to recognise a few chunks in a given position fortuitously, and thus obtains a better recall. Crucially, Sala and Gobet (2016) have recently demonstrated in a meta-analysis that this effect is present in nearly every domain of expertise reviewed. The overall correlation between expertise and recall of random material was moderate but statistically significant (r = .42, p < .001).

Chase and Simon's (1973) chunking theory spawned a large number of experimental studies. Several of these studies have been carried out in the domain of computer programming, and the importance of chunking in programming is generally accepted (e.g., Adelson, 1981; Barfield, 1986; McKeithen et al., 1981). In addition, Schmidt (1986) found that recall of computer programs correlates with their comprehension. Since high comprehension is a distinguishing feature of expertise in programming, this correlation suggests that chunks, as measured by the recall task, may play a causal role.

To our knowledge, no computational model has been developed so far to simulate the empirical data about memory for computer programs. The goal of the present paper is to fill in this gap, using the CHREST architecture as a modelling environment. Given Sala and Gobet's (2016) recent finding that the skill effect with random material generalises to many domains of expertise, the focus will be on the recall of randomised programs and the role played by perceptual chunks.

The paper is organised as follows. First, we briefly review research on memory for computer programs. Second, we describe a computer simulation using CHREST. Third, we compare the results of the simulations with those obtained with humans. Finally, we reflect on the impact of our results upon research into expertise.

Memory for Programs

Several studies have been carried out to investigate memory for computer code by individuals of different levels of expertise. While some of the studies were also interested in cognitive processing differences, this review will focus upon the studies where the recall task has been used—that is, a brief presentation of material taken from the domain of expertise, and a subsequent test of memory. In selecting the studies, we have also used the criterion that the experiment should compare memory recall ability between expert and novice programmers, and that some measure of performance (e.g., percentage of lines correctly recalled) was provided. These criteria resulted in the selection of four studies: Adelson (1981), Barfield (1986), Bateson, Alexander and Murphy (1987), and Guerin and Matthews (1990). Important features in these experiments include the participants' skill level and the type of stimulus given to them.

Assessing Programming Ability

Unlike similar endeavours into chess, there is no standard rating scale measuring a computer programmer's level of expertise. For the four experiments to be reviewed, each participant's level of expertise with a particular language was determined by their experience with it.[1] In general, the participants in the novice group had some experience with the programming language used in the experiment. The expert group usually consisted of programmers that had completed, or lectured on, courses in the language. Some programmers were rated as experts because they had experience in more languages than the target one in the experiment.

Experimental Material

For all experiments, the materials were examples of real computer code. Each experiment only used one programming language to draw its examples from, even if some of the participants knew more than one language. But unfortunately, there are no two experiments using the same language so as to allow direct comparison of results. Indeed, differences in the languages, experimental designs, and scoring methods make detailed comparisons awkward. Some authors (e.g., Guerin & Matthews, 1990) even criticised other experimenters for their choice of target programming language.

Summaries of Experiments

Adelson (1981). This experiment addressed the question of how experts represent and use programming concepts. It tried to show that experts use a hierarchy to organise their information and base its structure upon functional aspects of programming. This is opposed to novices who organise information based on the program syntax.

The experiment used the Polymorphic Programming Language (PPL). PPL is a variant of PL/I, which is a combination of FORTRAN, ALGOL and COBOL (see Schmidt, 1986). The novices were five undergraduates who had completed a course in PPL, and the experts were five lecturers in that language. Sixteen lines of PPL code taken from three separate, complete programs were used as stimuli. Each line of code was presented separately on a screen. Lines were presented in a random order and each line was visible for 20 seconds. After all lines had been presented, the participants had 8 minutes to recall the code. This procedure was repeated for nine trials.

Experts recalled more than the novices (see Figure 1). Adelson suggests that the discrepancy between Chase and Simon's (1973) chess data and her data comes from the fact that the code consists of lines taken from three complete programs and not of lines randomly selected from 16 different programs.

Adelson also looked at the size of chunks used in recall, defining a chunk as a sequence of items recalled in succession with less than a 10-second pause between them. Experts' chunk size was greater than novices' (on average 3.5 and 2.4 items, respectively). Based on these and additional results, Adelson concluded that experts organise information using functional principles, while novices categorise on a more syntactic (surface) basis.

Barfield (1986). Barfield was interested in being able to distinguish novice from expert problem solving behaviour and knowledge acquisition, and concentrated on chunking as the main process that discriminates individuals of different skill levels. He suggested that programmers take in the complex stimuli as meaningful chunks before they are processed.

Four levels of expertise were used. Naïve participants (n = 42) had not completed any programming courses. Novices (n = 80) had completed just one course in BASIC. Intermediates (n = 73) had completed a minimum of one BASIC course plus two or three courses in other languages. Experts (n = 26) were graduates in computer science as well as having at least one course in BASIC.

The material consisted of one 25-line program written in BASIC. The experimenters identified likely modules within the program, but no visible boundaries were marked (i.e. no spaces between lines). The experiment had three conditions: the stimulus could be presented either (a) in executable order, (b) with the order of lines randomised, or (c) with the order of modules randomised (in that case, the lines within a module preserved their order). The participants were allowed three minutes to study the stimulus and four minutes to recall it.

The results are summarised in Figure 1. Naïve and novice participants obtained the same level of performance regardless of stimulus type, indicating that little, if any, of the chunk knowledge possessed by experts is present with novices. According to the results, intermediates can chunk together lines of code as long as they are in executable order. As expected, randomising the lines did negatively affect the performance of the experts, although they still did better than Novices and Naïve participants. The randomising of modules did not affect the performance of experts and this was taken as support for Barfield's chunking explanation. However, Guerin and Matthews (1990) argue that Barfield is measuring recall and not comprehension, so even though the semantic structure of the program is tampered with, it will not affect the results because BASIC programmers are not as sensitive to the semantic complexity as programmers using other languages. They also criticise

[1]This is far from being a foolproof method. From research into other expertise domains, it is known that experience correlates only imperfectly with expertise (Gobet, 2016).

Barfield for not randomising lines within modules to complete his experimental design.

Bateson, Alexander and Murphy (1987). This paper aimed to expose expert-novice differences in syntactic and semantic memory, along with tactical and strategic skill. The gist of Bateson's experiment is to demonstrate the importance of semantics in gauging programmers' differences over tasks that use measures of memory and chunk size in syntactic recall. We discuss only the first of Bateson et al.'s battery of tasks, the syntactic memory task.

Two groups were used for this task: novices (n = 20), who had completed no more than three programming courses, and experts (n = 30), who had completed more than three courses. All participants had completed 12 weeks in an introductory FORTRAN class. The material used for the experiment was four short programs of equal length written in FORTRAN. Two of the four programs had lines randomised.

Participants were given only one normal and one random program, and were given three minutes to study a program and then allowed four minutes of free recall to write down what they remembered. The means for the proportion of total program recall are shown in Figure 1. As with Adelson (1981), there is a skill effect even when the order of the lines is randomised.

Guerin and Matthews (1990). This study aims to demonstrate the role of semantic knowledge in expert programmer ability. Only the first of their three experiments is described here. Guerin and Matthews used COBOL as their target language, and a genuine 116-line COBOL program was used as material. Two groups were used; the novices (n = 52) had an average of 0.5 years of programming experience and the experts (n = 52) had an average of 4.7 years. The participants were given 10 minutes to study the stimulus and then 8 minutes of free recall. There were four conditions: (a) Normal program; (b) Random lines within program modules; (c) Random modules; and (d) Random lines within program modules and random modules. A module is described by Guerin and Matthews as being a chunk of a program; however, they do not go into detail as to whether they are describing a functional section of a program or an amount of information thought capable of being memorised in one go. Nor do they specify how large these modules are.

Guerin and Matthews used a unique method to score recall trials. Instead of counting correctly recalled items, a system of points based on positional and lexical accuracy was devised in order to better describe recall performance. For each line, one point was earned if more than half of the components of the line were recalled correctly; an additional point was added if it was recalled in the correct sequence in the program, and a final point was added if it was recalled exactly as the original. Thus, a maximum of three points could be earned for each line.

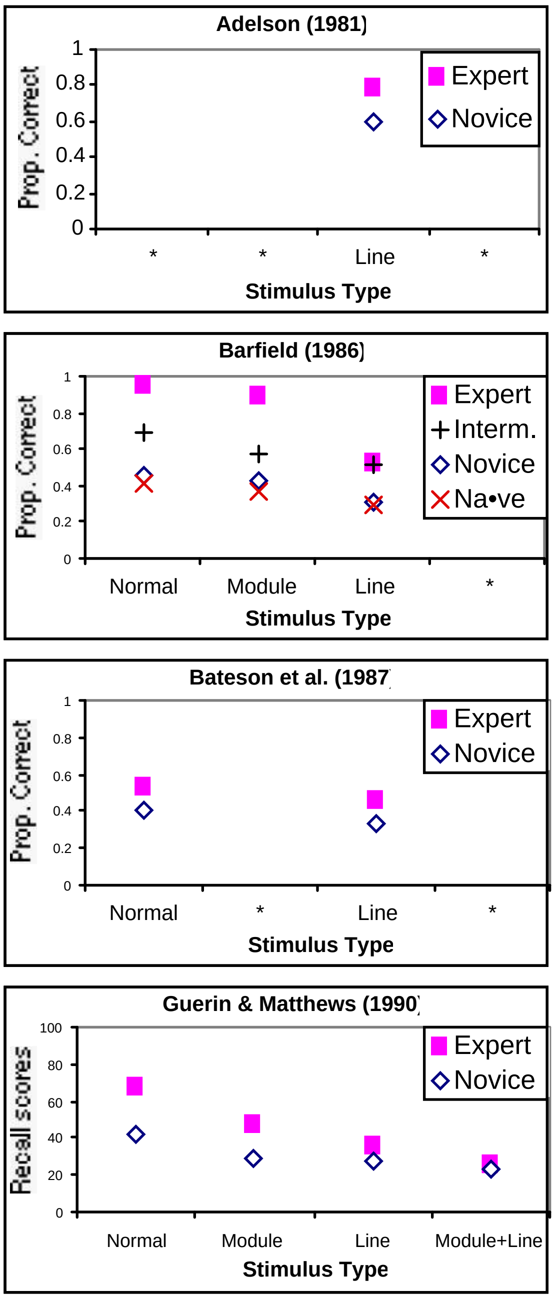

Figure 1. Memory for computer code as a function of level of expertise and type of randomisation. Adelson (1981) and Barfield (1986) used the number of lines correct, and Bateson et al. (1987) used the number of items correct (to facilitate comparison, we have converted these absolute numbers into proportion correct). Guerin and Matthews (1990) devised their own scoring method.

In all four conditions, experts were better than novices. The ordering of the conditions was as follows: Normal > Random modules > Random lines within program modules > Random Lines and Modules (see Figure 1). Finally, Guerin and Matthews found that recall correlated highly with comprehension, which was measured by having participants write a summary of the purpose of the program and how this purpose was achieved. The authors conclude that the superior semantic knowledge and comprehension skills of experts that allowed them to obtain better performance than novices could not be used when the lines of the program were randomised.

Summary. Several phenomena clearly stand out from these experiments. First, as expected, experts always do better than non-experts with executable code. Second, there is a robust correlation between recall and comprehension. Third, some evidence has been uncovered that experts have larger chunks than novices. Finally, the expertise effect is also present when the code has been scrambled in various ways. In the remainder of this article, we describe simulations with CHREST showing that this cognitive architecture captures the skill effect with randomised computer programs.

The CHREST Architecture

As mentioned above, the chunking theory has often been proposed, in its informal form, as an explanation of the skill effect found in memory tasks for computer programs. In the remainder of the paper, we wish to explore to what extent a computational implementation of the theory, which emphasises perceptual chunking, can account for the empirical data we have just reviewed.

CHREST (Chunk Hierarchy and REtrieval STructures; De Groot & Gobet, 1996; Gobet, 1993; Gobet & Lane, 2005; Gobet & Simon, 2000) is an expansion of the EPAM (Elementary Perceiver And Memorizer) cognitive architecture (Feigenbaum & Simon, 1984; Richman, Staszewski & Simon 1995). At the core of EPAM and CHREST lie mechanisms for encoding chunks into long-term memory (LTM) through the construction of a discrimination net and mechanisms for handling information in short-term memory (STM). Together, EPAM and CHREST have been used to account for domains such as verbal behaviour, chess memory, expert digit-span memory, use of multiple representations in physics, letter perception, spelling and acquisition of language (see Gobet et al., 2001, Gobet & Lane, 2005, for reviews).

CHREST consists of the following components: discrimination network, semantic LTM, and STM. STM, which consists of at most four chunks, is mostly a queue (first-in, first out). However, the largest chunk met at any point in time (the *hypothesis)*, is kept in STM until a larger chunk is met or constructed (see Gobet & Simon, 2000).

The net is grown by two EPAM-like learning mechanisms, *familiarisation* and *discrimination*. When a new object is presented to the model, it is sorted through the discrimination net. When a node is reached, the object is compared with the *image* of the node, which is its internal representation. If the image under-represents the object, new features are added to the image (familiarisation). If the information in the image and the object differ on some feature or some sub-element, a new node is created (discrimination).

Table 1 shows the key time parameters used with CHREST. These parameters are taken from previous work (Feigenbaum & Simon, 1984; De Groot & Gobet, 1996; Gobet & Simon, 2000) and are important in that they impose stringent constraints on how much information processing can be performed both during the training phase and during the presentation of the stimulus in the test phase. Note that creating an LTM chunk, adding a new link to a chunk, or familiarising a chunk occurs in parallel with the other operations.

CHREST incorporates mechanisms for incrementally creating schemas (known as templates in the theory), allowing information to be rapidly encoded in slots (Gobet & Simon, 2000). In this paper, we are primarily interested in how far perceptual chunks can account for skill effect in the recall of scrambled programs. Therefore, we did not use templates in the simulations.

Table 1: Main time parameters used in CHREST

Cognitive operation	Duration
creating an LTM chunk	8 s
familiarising an LTM chunk	2 s
placing a chunk into STM	50 ms
comparing two chunks	50 ms
carrying out a test in the discrimination net	10 ms

Simulations

For training and testing the model, a large collection of data was gathered from a variety of sources. Using the internet and some reference books, a corpus of about one hundred different FORTRAN programs was built.

Training Phase

During training, CHREST is given programs in FORTRAN—a naturalistic material—as input so that the vocabulary of the language as well as some sequences of items can be learnt. The lines of code had a mean length of 7 words. Elements (e.g. numbers, punctuation and other special characters) are recognised as distinct, individual items by the model. This type of input allows the model to build a discrimination net that encodes both the primitive items and legal strings from the computer language.

The same basic model is used to simulate different levels of ability; that is, only the amount of input is varied, and no other mechanisms or parameters are altered. This study will focus on the difference between *novices* and *experts,* who are simulated by passing CHREST either one program or a corpus of eighty-eight programs during training (these numbers were chosen arbitrarily).

As each item is passed to CHREST, the model constructs its discrimination net. The net initially starts with an empty root node. Primitive items are usually the first to be added to the net. Then, after a period of learning, the images at the nodes will come to represent sequences of items.

Test phase

Once the appropriate training had been undertaken (study of one program for novices and eighty-eight for experts), the novice and expert level models were tested using twelve new FORTRAN programs that were not included in the training set. Test programs were selected that did not include too many "print" statements and so that they were all of roughly the same number of lines and words. The simulations were run like an experiment with human participants, with a presentation time of five seconds per line. Various levels of randomisation were applied to the test programs before they were passed to the models. In addition to the conditions used in the studies reviewed above, we also thought it interesting to use a condition in which all elements of a program were randomised. There were therefore five conditions in total:

1. **Normal.** The sequence of the program is unaltered.
2. **Random Modules.** Segments of the program are randomised, but the line order within a segment is retained.
3. **Random Lines**. The lines of the program are randomised. Information within a line is unaltered.
4. **Modules and Lines**. Both modules and lines within modules are randomised.
5. **All Random**. All elements within a program are randomised, yielding a total randomisation.

To create the Random Modules, lines of code that acted as a meaningful unit of instruction were grouped. For example, lines belonging to declaration statements would be retained together as a module. For the All Random condition, the maximum and minimum line lengths of the original program were first noted; then, all the elements in the program were randomised and lines of random length were constructed, with the condition that the values fell between the original lengths.

Each model received two "practice" problems which were always the same. The first practice program was a Normal type and the second was an All Random type. The test programs were then presented. In addition to the practice problems, each model received two examples of each condition, thus making a total of 12 test programs. To control for random variation due to the order of programs in the learning set, CHREST was run with 40 simulations per skill level. The random order in which the programs appeared, the random order of the conditions and the randomisation of programs were all reset for each simulation.

For each program, CHREST read the program line by line, storing recognised chunks into STM, and, when applicable, using the following learning mechanisms. First, as described before, CHREST can add a chunk as a test to another chunk. It takes 8 seconds to carry out this discrimination operation. Typically, a new test is added to the hypothesis. Second, for chunks that have been in STM for at least 8 seconds, a new branch is added to access them by a novel path; this essentially means that episodic cues that permit access to this node are added to the discrimination net (Gobet & Simon, 2000). Such nodes can be recalled during the reconstruction phase even if they are no longer in STM.

During the recall phase, CHREST could output the information held in STM and in the nodes that had been created or for which new access links had been created. Recall was scored in the following way. A list of items that CHREST had recalled, in the order they were retrieved, was collected from the model. This recall list was matched alongside the original stimulus that was presented to CHREST. The first line in the stimulus was compared to the recall list and if the first items matched, then the lines were compared to find out how many of the items were recalled correctly before a mismatch occurred; both lines were then discarded and the next stimulus line compared to the recall. If the stimulus line did not match the recall line, then the line was discarded and the next one matched against the recall list, until all the stimulus lines were used. Not only does this method show how many items were recalled correctly, but it shows how many errors of commission the model made.

Results

Figure 2 illustrates the results of the simulations. We can see that for all cases, CHREST predicts a skill effect. These predictions are borne out by the data, with the exception of Guerin and Matthews' (1990) Module+Line condition, where no effect was found with the human participants. None of the studies reviewed incorporated the All condition, where the order of all elements of the code is randomised. Although the difference between the Novice and Expert models is small for the All condition, it is statistically reliable ($t(78) = 3.71$, $p < .001$). To test this counter-intuitive prediction of the model, we collected data with C programmers (n = 9) and novices (n = 9); the results have supported the prediction. Given that we found a skill effect with full randomisation, it is unclear why no such effect was found with Guerin and Matthews' Module+Line condition, which destroys less structure than our method.

While the simulations show a differential effect for the type of randomisation, this effect is limited to the All conditions vs. the other conditions. The model fails to capture the differential recall shown by humans from the Normal condition to the Module+Line conditions. It is likely that humans pick up semantic information from the modules, as proposed for example by Adelson (1981), which are beyond the essentially perceptual knowledge that this simplified version of CHREST can store.

Conclusion

The experimental studies reviewed in this paper clearly show that randomisation of aspects of computer code affects recall, while still preserving a skill effect in most cases. In

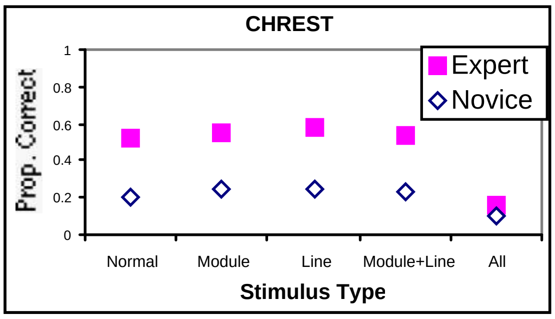

Figure 2. CHREST's memory for computer code as a function of level of expertise and type of randomisation.

order to account for these data, we have described a simplified version of the CHREST simulation model, where emphasis was given to perceptual chunking. We have shown that the model accounts for the skill differences with scrambled programs, which supports Simon and Gobet's (2000) contention that theories based on chunking mechanisms can account for skill effects in memory for computer programs. However, the model did not show differences between the randomisation conditions, as humans did. In this respect, the results differ to those obtained with chess, where it has been shown that CHREST is able to successfully capture recall differences with chess positions that were randomised in different ways (Gobet & Waters, 2003). A difference between the simulations in the two domains is that CHREST used templates with chess, but not with computer programs.

The presence of a skill effect even with randomised material has been demonstrated not only in chess and computer programming, but also in nearly all domains of expertise where this has been studied (Sala & Gobet, 2016). This finding strongly suggests that theories of expertise cannot only propose high-level and holistic mechanisms, but must also include some low-level mechanisms such as chunking to account for the empirical data. In this respect, CHREST is obviously on the right track. Further work will establish whether the presence of templates (Gobet & Simon, 2000) can help the model capture the differential recall of different types of scrambled programs, which is often claimed to tap into differences in high-level, semantic knowledge (e.g., Adelson, 1981).

References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition*, *9*, 422-433.

Barfield, W. (1986). Expert-novice differences for software: Implications for problem-solving and knowledge acquisition. *Behaviour and Information Technology*, *5*, 15-29.

Bateson, A. G., Alexander, R. A., & Murphy, M. D. (1987). Cognitive processing differences between novice and expert computer programmers. *International Journal of Man-Machine Studies*, *26*, 649-660.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, *4*, 55-81.

de Groot, A. D., & Gobet, F. (1996). *Perception and memory in chess.* Assen: Van Gorcum.

Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science*, *8*, 305-336.

Gobet, F. (1993). A computer model of chess memory. *Proceedings of 15th Annual Meeting of the Cognitive Science Society*, (pp. 463-468). Hillsdale, NJ: Erlbaum.

Gobet, F. (2016). *Understanding expertise: A multi-disciplinary approach*. London: Palgrave.

Gobet, F., & Lane, P. C. R. (2005). The CHREST architecture of cognition: Listening to empirical data. In D. Davis (Ed.), *Visions of mind* (pp. 204-224). Hershey, PA: IPS.

Gobet, F., Lane, P.C.R., Croker, S., Cheng, P.C-H., Jones, G., Oliver, I., & Pine, J. (2001). Chunking mechanisms in human learning. *TRENDS in Cognitive Sciences*, *5*, 236-243

Gobet, F., & Simon, H. A. (1996). Templates in chess memory: A mechanism for recalling several boards. *Cognitive Psychology*, *31*, 1-40.

Gobet, F. & Simon, H. A. (2000). Five seconds or sixty? Presentation time in expert memory. *Cognitive Science*, *24*, 651-682.

Gobet, F., & Waters, A. J. (2003). The role of constraints in expert memory. *Journal of Experimental Psychology: Learning, Memory & Cognition*, *29*, 1082-1094.

Guerin, B., & Matthews, A. (1990). The effects of semantic complexity on expert and novice computer program recall and comprehension. *The Journal of General Psychology*, *117*, 379-389.

McKeithen, K. B., Reitman, J. S., Rueter, H. H., & Hirtle, S. C. (1981). Knowledge organisation and skill differences in computer programs. *Cognitive Psychology*, *13*, 307-325.

Richman, H. B., Staszewski, J., & Simon, H. A. (1995). Simulation of expert memory with EPAM IV. *Psychological Review*, *102*, 305-330.

Sala, G., & Gobet, F. (2016). Experts' memory superiority for domain-specific random material generalises across fields of expertise: A meta-analysis. Manuscript submitted for publication.

Schmidt, A. L. (1986). Effects of experience and compre-hension on reading time and memory for computer pro-grams. *International Journal of Man-Machine Studies*, *25*, 399-409.

Simon, H. A. & Gobet, F. (2000). Expertise effects in memory recall: A reply to Vicente and Wang. *Psychological Review*, *107*, 593-600.

Ye, N., & Salvendy, G. (1994). Quantitative and qualitative differences between experts and novices in chunking computer software knowledge. *International Journal of Human-Computer Interaction*, *6*, 105-118.