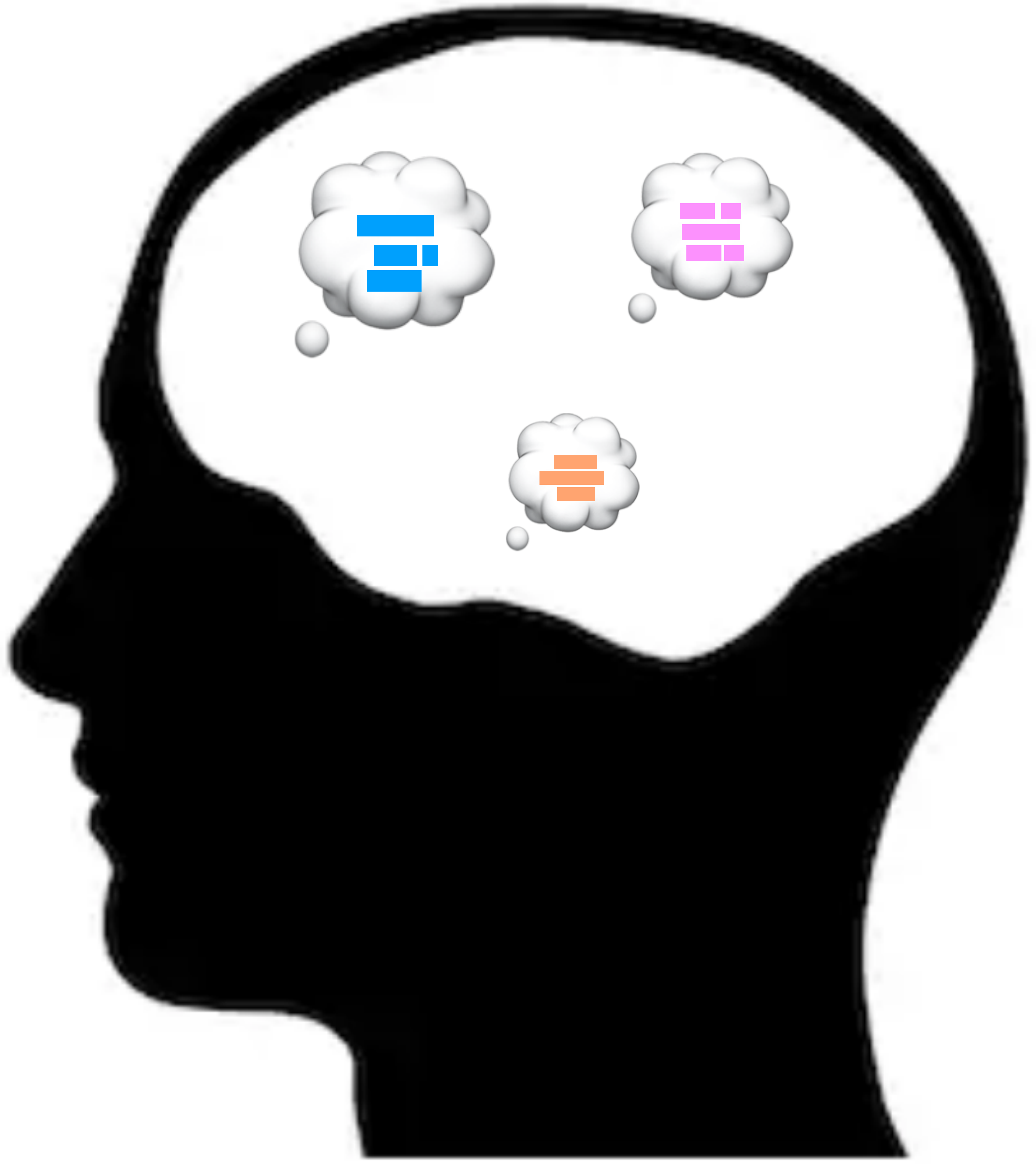
The Brain

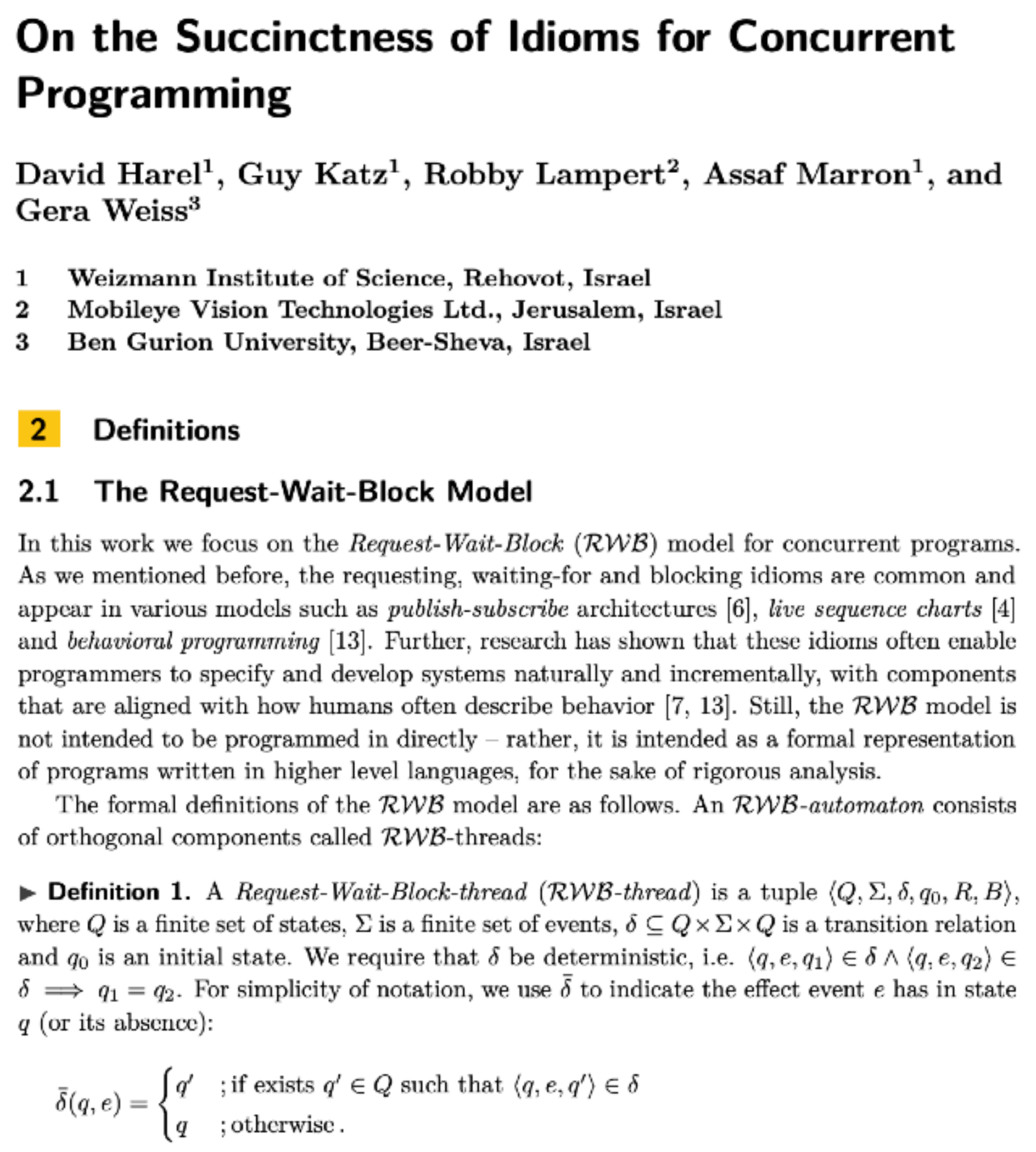


There is apparently no modification of existing initial memories -- no insertions, no cut-and-paste -- only more and more experiences.

Images seen, sentences heard, pain felt, are **all amassed as new memories and connected to existing memories in more ways than we can imagine today**.

Some of these, of course, explain, refine, correct, reorganize, or completely replace things that were previously experienced (or seen or heard or read) in how they affect future behavior.

🎓 Research



On the Succinctness of Idioms for Concurrent Programming

David Harel[1], Guy Katz[1], Robby Lampert[2], Assaf Marron[1], and Gera Weiss[3]

1 Weizmann Institute of Science, Rehovot, Israel
2 Mobileye Vision Technologies Ltd., Jerusalem, Israel
3 Ben Gurion University, Beer-Sheva, Israel

2 Definitions

2.1 The Request-Wait-Block Model

In this work we focus on the *Request-Wait-Block* ($\mathcal{RWB}$) model for concurrent programs. As we mentioned before, the requesting, waiting-for and blocking idioms are common and appear in various models such as *publish-subscribe* architectures [6], *live sequence charts* [4] and *behavioral programming* [13]. Further, research has shown that these idioms often enable programmers to specify and develop systems naturally and incrementally, with components that are aligned with how humans often describe behavior [7, 13]. Still, the $\mathcal{RWB}$ model is not intended to be programmed in directly – rather, it is intended as a formal representation of programs written in higher level languages, for the sake of rigorous analysis.

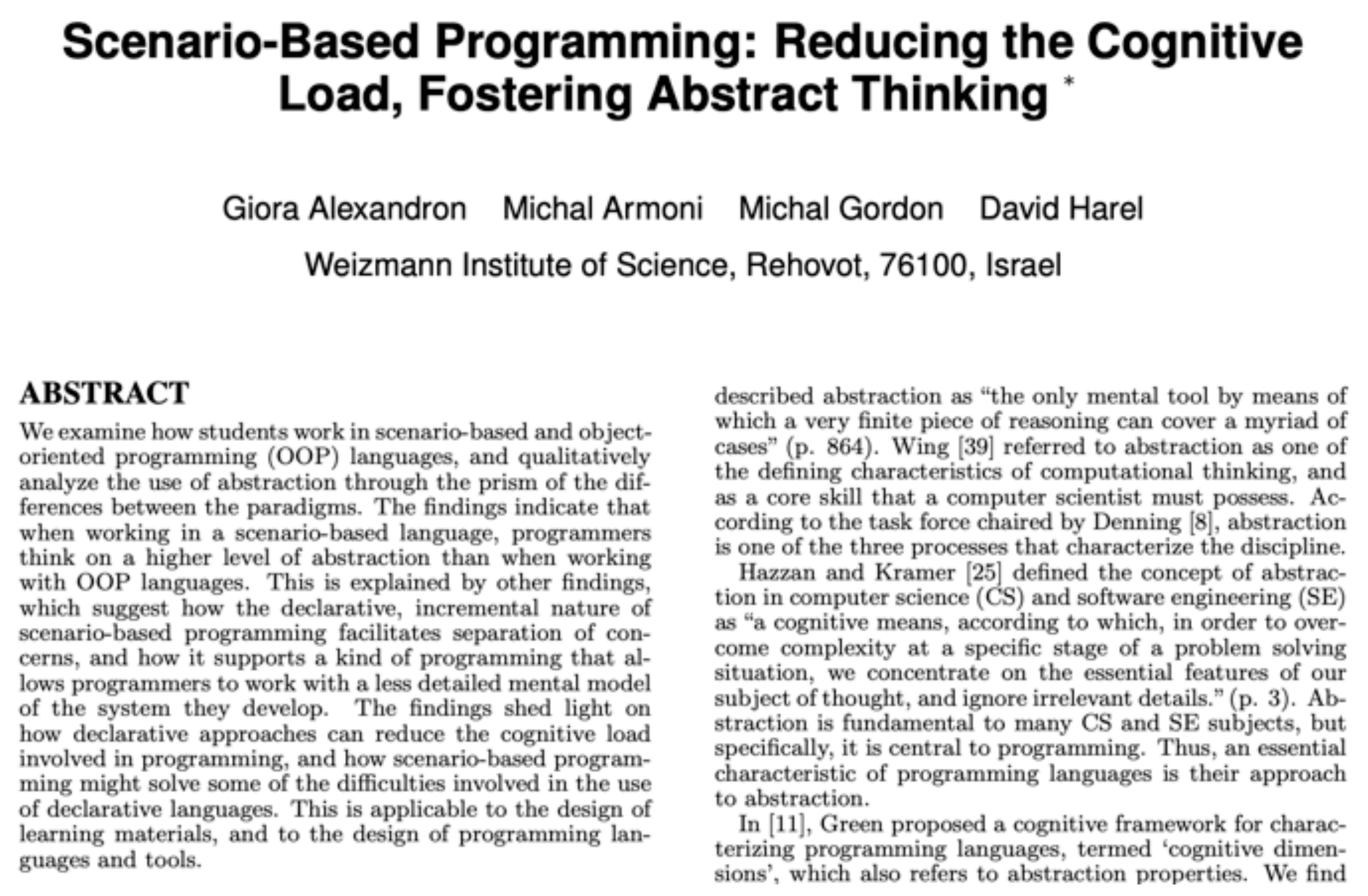
The formal definitions of the $\mathcal{RWB}$ model are as follows. An *$\mathcal{RWB}$-automaton* consists of orthogonal components called $\mathcal{RWB}$-threads:

▶ **Definition 1.** A *Request-Wait-Block-thread* (*$\mathcal{RWB}$-thread*) is a tuple $\langle Q, \Sigma, \delta, q_0, R, B\rangle$, where Q is a finite set of states, Σ is a finite set of events, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation and q_0 is an initial state. We require that δ be deterministic, i.e. $\langle q, e, q_1\rangle \in \delta \wedge \langle q, e, q_2\rangle \in \delta \implies q_1 = q_2$. For simplicity of notation, we use $\bar{\delta}$ to indicate the effect event e has in state q (or its absence):

$$\bar{\delta}(q,e) = \begin{cases} q' & ; \text{if exists } q' \in Q \text{ such that } \langle q, e, q'\rangle \in \delta \\ q & ; \text{otherwise}. \end{cases}$$

“behavioral programming harel”

http://www.wisdom.weizmann.ac.il/~harel/papers.html



Scenario-Based Programming: Reducing the Cognitive Load, Fostering Abstract Thinking *

Giora Alexandron Michal Armoni Michal Gordon David Harel

Weizmann Institute of Science, Rehovot, 76100, Israel

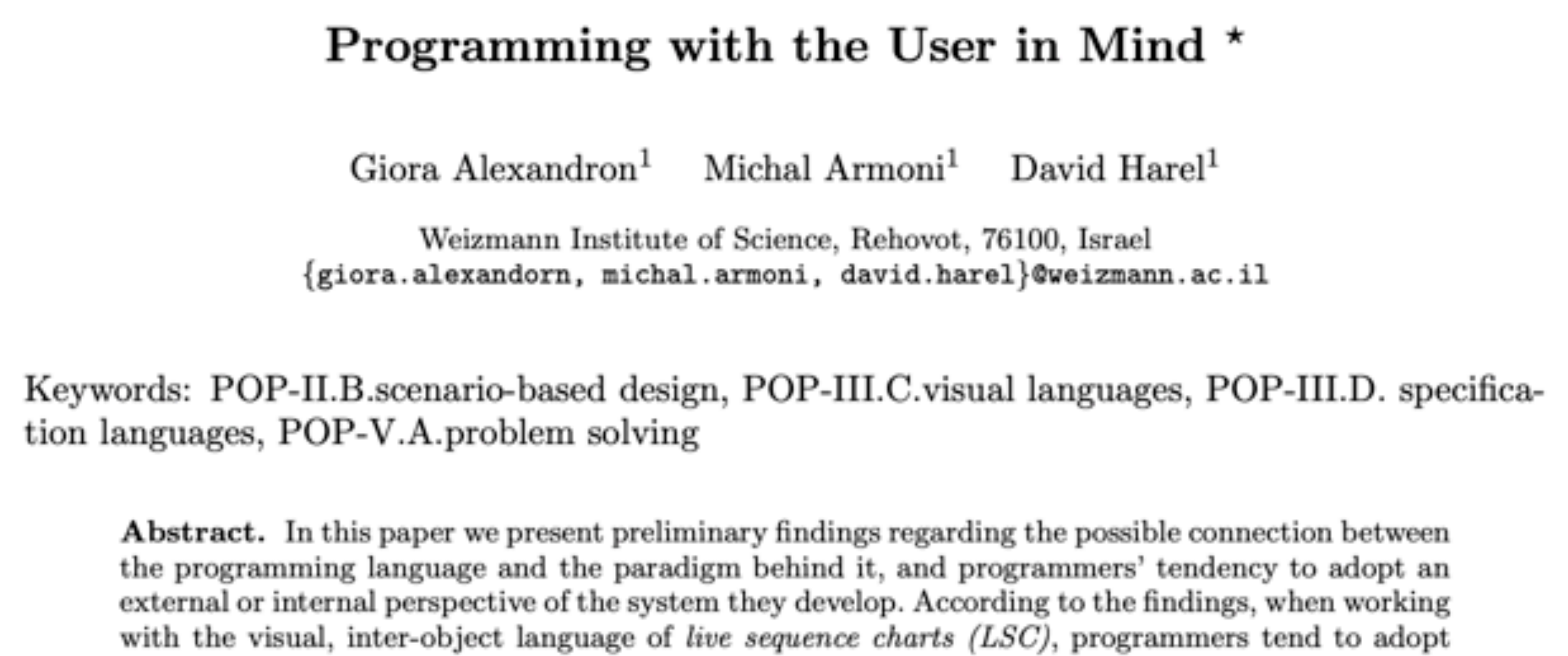
ABSTRACT

We examine how students work in scenario-based and object-oriented programming (OOP) languages, and qualitatively analyze the use of abstraction through the prism of the differences between the paradigms. The findings indicate that when working in a scenario-based language, programmers think on a higher level of abstraction than when working with OOP languages. This is explained by other findings, which suggest how the declarative, incremental nature of scenario-based programming facilitates separation of concerns, and how it supports a kind of programming that allows programmers to work with a less detailed mental model of the system they develop. The findings shed light on how declarative approaches can reduce the cognitive load involved in programming, and how scenario-based programming might solve some of the difficulties involved in the use of declarative languages. This is applicable to the design of learning materials, and to the design of programming languages and tools.

described abstraction as “the only mental tool by means of which a very finite piece of reasoning can cover a myriad of cases” (p. 864). Wing [39] referred to abstraction as one of the defining characteristics of computational thinking, and as a core skill that a computer scientist must possess. According to the task force chaired by Denning [8], abstraction is one of the three processes that characterize the discipline.

Hazzan and Kramer [25] defined the concept of abstraction in computer science (CS) and software engineering (SE) as “a cognitive means, according to which, in order to overcome complexity at a specific stage of a problem solving situation, we concentrate on the essential features of our subject of thought, and ignore irrelevant details.” (p. 3). Abstraction is fundamental to many CS and SE subjects, but specifically, it is central to programming. Thus, an essential characteristic of programming languages is their approach to abstraction.

In [11], Green proposed a cognitive framework for characterizing programming languages, termed ‘cognitive dimensions’, which also refers to abstraction properties. We find



Programming with the User in Mind *

Giora Alexandron[1] Michal Armoni[1] David Harel[1]

Weizmann Institute of Science, Rehovot, 76100, Israel
{giora.alexandorn, michal.armoni, david.harel}@weizmann.ac.il

Keywords: POP-II.B.scenario-based design, POP-III.C.visual languages, POP-III.D. specification languages, POP-V.A.problem solving

Abstract. In this paper we present preliminary findings regarding the possible connection between the programming language and the paradigm behind it, and programmers' tendency to adopt an external or internal perspective of the system they develop. According to the findings, when working with the visual, inter-object language of *live sequence charts (LSC)*, programmers tend to adopt