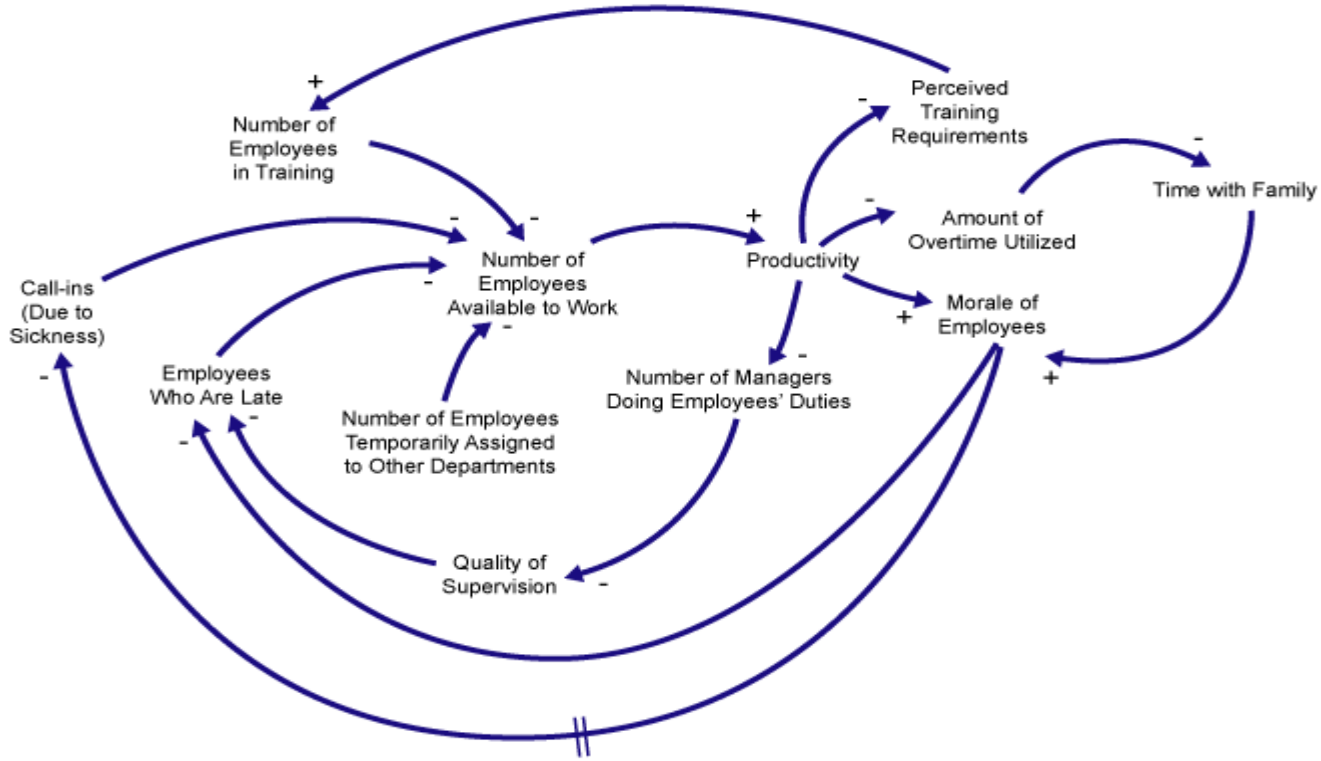
The Dynabook as a tool for Research and Development

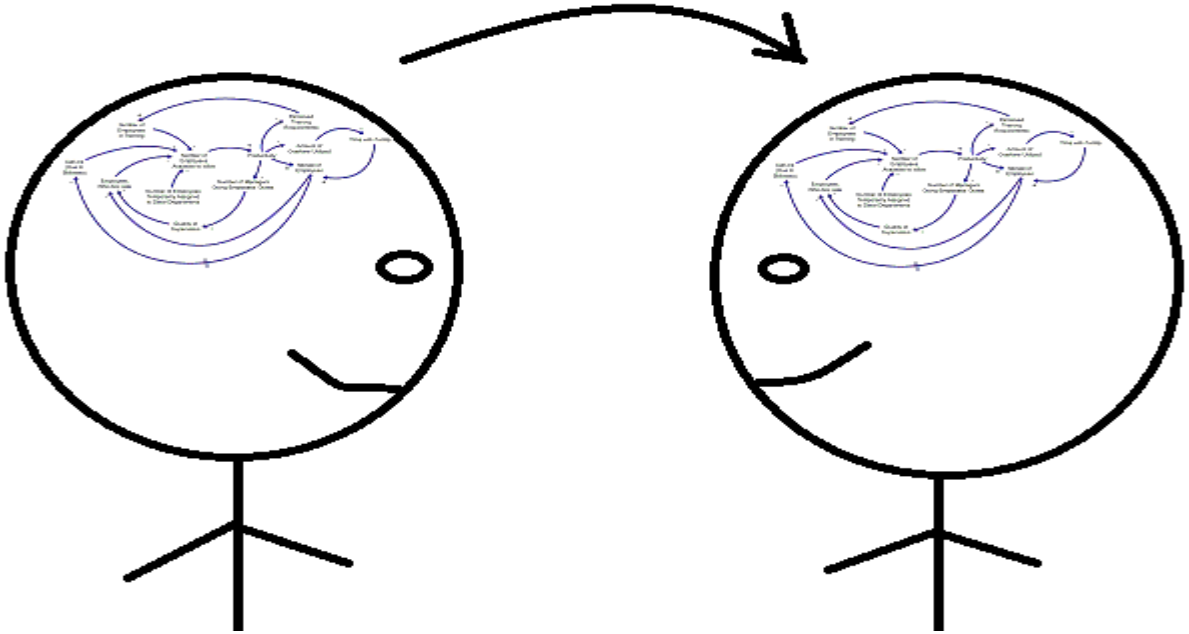
Juan Vuletich
www.cuis-smalltalk.org
@JuanVuletich

Knowledge



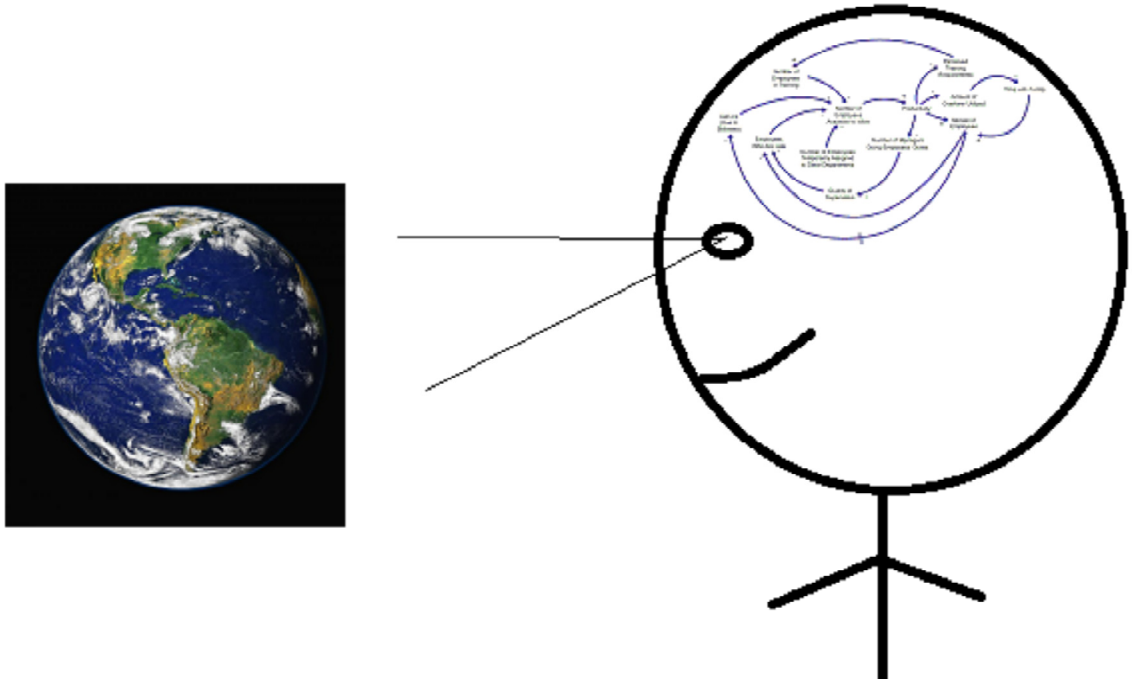
Mental models of some reality

Teaching and Learning



Duplicating those models in another mind

Research and Discovery



Bulding new knowledge by ourselves

Representing Knowledge: Books and Journals

- Not models. Just descriptions
- Limited media: Just words and pictures
- Static (can not show dynamic phenomena)
- Inert (don’t react to our queries)

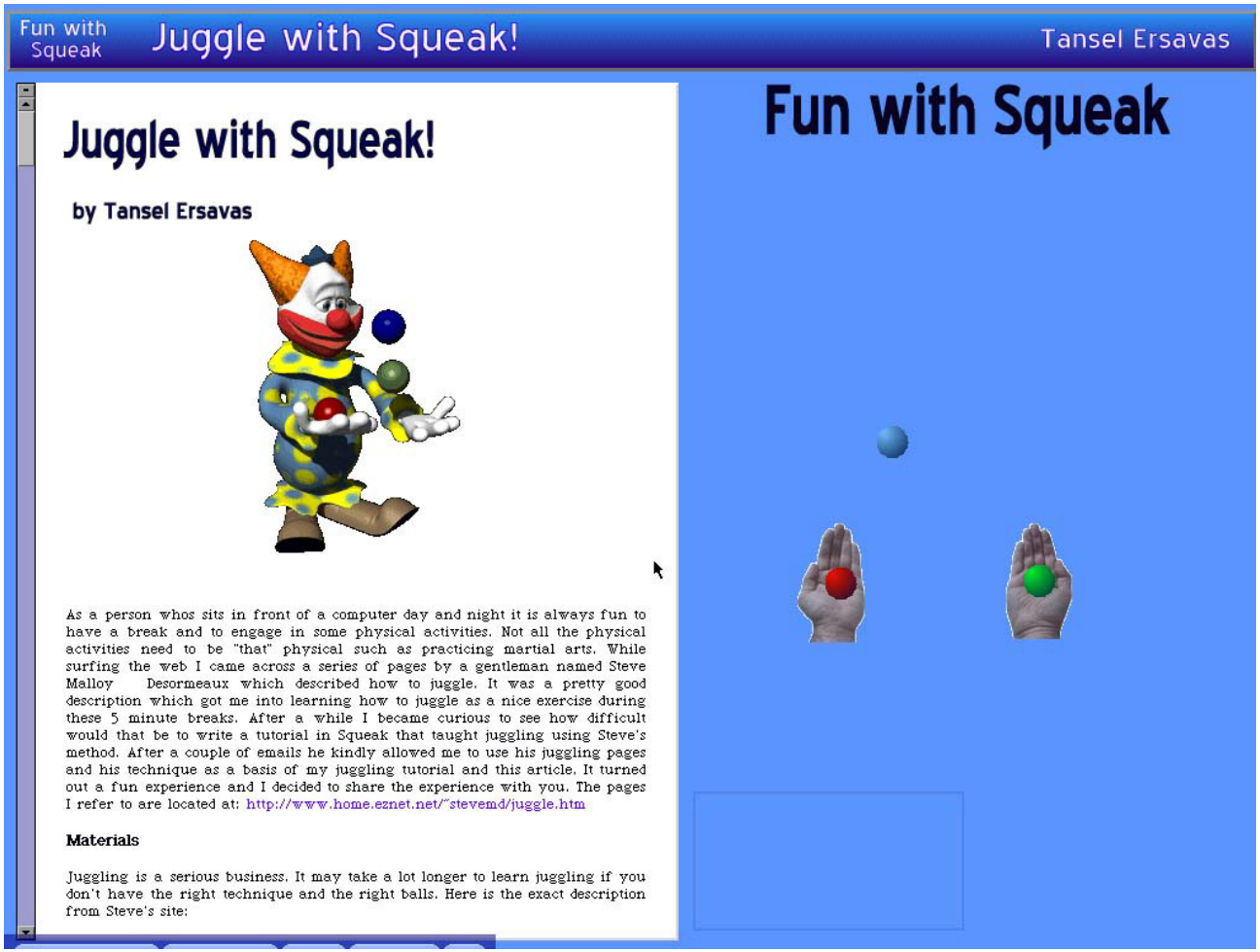
Representing Knowledge: Video

- Not models. Just descriptions
- More powerful media
- Dynamic. Can show evolution and change
- Inert (don’t react to our queries)

Representing Knowledge: Dynabook

- Explanations + Models
- Dynamic Media
- Interactive

A great example: SqueakNews



A Dynabook can be
Not just for Teaching and Learning

An Experimental Lab

A Dynabook can be

A Lab Notebook

A Dynabook can be

International Science Index
International Journal of Social, Behavioral, Educational, Economic and Management Engineering Vol:9 No: 2, 2015

Wasting Human and Computer Resources

Mária Csernoch, Piroska Biró

***Abstract*—**The legends about "user-friendly" and "easy-to-use" birotical tools (computer-related office tools) have been spreading and misleading end-users. This approach has led us to the extremely high number of incorrect documents, causing serious financial losses in the creating, modifying, and retrieving processes. Our research proved that there are at least two sources of this underachievement: (1) The lack of the definition of the correctly edited, formatted documents. Consequently, end-users do not know whether their methods and results are correct or not. They are not aware of their ignorance. They are so ignorant that their ignorance does not allow them to realize their lack of knowledge. (2) The end-users' problem solving methods. We have found that in non-traditional programming environments end-users apply, almost exclusively, surface approach metacognitive methods to carry out their computer related activities, which are proved less effective than deep approach methods.

Based on these findings we have developed deep approach methods which are based on and adapted from traditional programming languages. In this study, we focus on the most popular type of birotical documents, the text based documents. We have provided the definition of the correctly edited text, and based on this definition, adapted the debugging method known in programming. According to the method, before the realization of text editing, a thorough debugging of already existing texts and the categorization of errors are carried out. With this method in advance to real text editing users learn the requirements of text based documents and also of the correctly formatted text.

The method has been proved much more effective than the previously applied surface approach methods. The advantages of the method are that the real text handling requires much less human and computer sources than clicking aimlessly in the GUI (Graphical User Interface), and the data retrieval is much more effective than from error-prone documents.

***Keywords*—**Deep approach metacognitive methods, error-prone birotical documents, financial losses, human and computer resources.

I. Introduction

IN the Graphical User Interface (GUI), end users almost exclusively apply surface approach methods for computer-related problem-solving. As a result they focus on providing outputs, instead of solving problems [1], [3], [4], [13], [15]. This approach has led to a high percentage of error prone administrative (birotical) documents, which are highly demanding of human and computer resources, causing serious financial losses both in the productive and retrieval processes [8], [12]–[16], [18]. It is not common knowledge that non-traditional software environments are also algorithmically driven, and to solve problems in these programs the same approach should be applied as with "real" programming [2], [5], [17], [19].

M. Cs. is with University of Debrecen Faculty of Informatics (phone: +36-52-512-900/75128; e-mail: csernoch.maria@inf.unideb.hu).
P. Bi. is with University of Debrecen Faculty of Informatics (e-mail: biro.piroska@inf.unideb.hu).

Faced with these problems, we have developed deep approach metacognitive methods for computer-related problem-solving in "birotical" environments, focusing on text management in the present paper. The methods are adapted from high level programming languages, and follow problem-solving and debugging methods which have so far proved effective and efficient.

II. Theoretical Background

A. Metacognitive Approaches to Computer-Related Problem-Solving

To cover all computer related activities, both in traditional and non-traditional programming environments, Case and Gunstone's [6] well accepted system of metacognitive problem solving approaches had to be extended (Fig. 1) with one deep (CAAD, Computer Algorithmic and Debugging-based) and one surface approach (TAEW, Trial-and-Error Wizard-based) category [7].

B. Levels of Mastery

Generally speaking, non-traditional software environments are not considered programming environments. The GUI and the support from the software companies suggest that user-friendly environment does not require any algorithmic skills, and there is no need for any computational thinking [20]. However, this is not so; to work effectively in "birotical" environments, similar to programming, the order of the three levels of mastery should also be followed [7]:

- *Familiarity:* The student understands what a concept is or what it means.
- *Usage:* The student is able to use or apply a concept in a concrete way.
- *Assessment:* The student is able to consider a concept from multiple viewpoints and/or justify the selection of a particular approach to solve a problem.

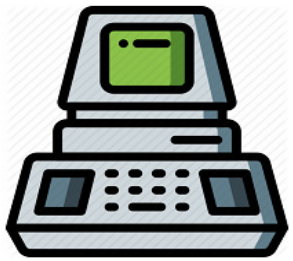
The GUI-support approaches leave out the first level of mastery, and the focus is on the usage. However, with this approach there are at least two fundamental problems:

- The one mentioned earlier; the high number of error prone documents and the wasted human and computer resources,
- Even more series the other consequence; leaving out the first level of mastery, we would never have a chance to reach the third level, the assessment.

We have to note here that there are sources which claim that our failure in teaching Informatics and Computer Sciences are rooted in Word and Excel [21]. However, we are convinced that Word and Excel should not be blamed. They are harmless, algorithmic based pieces of software. The popular and widely accepted surface approach methods and those who teach and

A valid Scientific Publication Format

Integrating



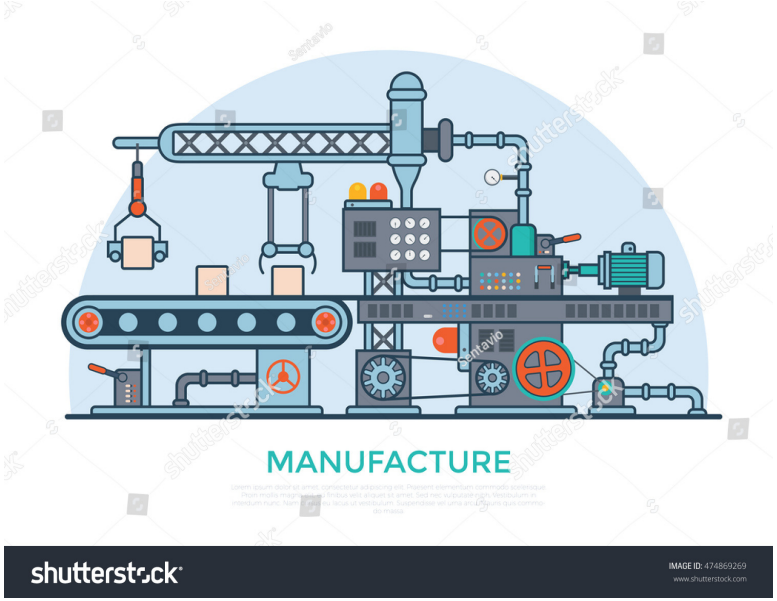
Computable Models of Scientific Theories

Explanations as text with static and dynamic graphs and data explorers

Experiments (Tests)

R & D

Enough about Research. What abut development?



Computable Models of Theories are in fact Frameworks.
Just use them to build Applications
And use them to keep learning and improving theories

My experience at Satellogic

Built computable theories (knowledge) about

- Telescopes and Sensors
- Orbits and Satellite maneuvers
- Sun light spectra
- Reflectance of various kinds of Grounds
- Atmospheric effect (why the sky is blue)
- Cartography
- 3D models of terrain (Digital Elevation Models)

These also resulted in practical applications of Satellite Image Processing

How does this differ from Python + Jupyter Notebooks?

Python provides good libraries for several domains, but they are usually installed without source code, are maintained by separate groups, and are not meant to be modified and extended by the user. Library evolution is slow, and libraries usually don't fit well with the problem you are solving. In Smalltalk, all the code is on equal footing. Modifying your code, domain specific libraries, core libraries, or the environment itself is equally possible.

In Python, common practice is to have generic models in imported libraries, and no models in application code: just ad hoc mashups of library calls. This doesn't lead to building theories, just working code. Smalltalk encouragse the development of your own models, making theories and computable knowledge real.

How does this differ from Python + Jupyter Notebooks?

Python has 35 reserved words, and a complex syntax that takes a while to learn. Smalltalk has 4 reserved words, and almost no syntax.

In Jupyter notebooks, text, code, and graphs are three distinct kinds of content, each built with specific editors. Documents are a sequence of “cells,” each cell being one kind. In Smalltalk, integration can be much tighter

How to do next

Start building and publishing Dynabooks
Improve Cuis support for Dynabooks
Do not ask the reader to know Smalltalk

This could be a “Killer App” for Smalltalk