The Role of Abstraction in Software Engineering

Conference Paper *in* ACM SIGSOFT Software Engineering Notes · November 2006

DOI: 10.1145/1370175.1370239 · Source: DBLP

2 authors:

Jeff Kramer
Imperial College London
379 PUBLICATIONS **18,855** CITATIONS
SEE PROFILE

Orit Hazzan
Technion - Israel Institute of Technology
259 PUBLICATIONS **3,280** CITATIONS
SEE PROFILE

The Role of Abstraction in Software Engineering

Jeff Kramer
Department of Computing
Imperial College
London SW7 2AZ, U.K.
Tel: +44-20 7594 8271
j.kramer@imperial.ac.uk

Orit Hazzan
Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa 32000, Israel
Tel: (9724) 8293 107
oritha@techunix.technion.ac.il

ABSTRACT
This workshop explores the concept of abstraction in software engineering at the individual, team and organization level. The aim is to explore the role of abstraction in dealing with complexity in the software engineering process, to discuss how the use of different levels of abstraction may facilitate performance of different activities, and to examine whether or not abstraction skills can be taught.

Categories and Subject Descriptors
D.2. [**Software Engineering**]: General.

General Terms
Management, Human Factors.

Keywords
Abstraction, software engineering processes, cognition, management, organizational learning.

1. INTRODUCTION
Why is it that some software engineers are able to produce clear, elegant designs and programs, while others cannot? One hypothesis is that the answer lies in *abstraction*: the ability to perform abstract thinking and to exhibit abstraction skills [3]. Abstraction is a cognitive means by which engineers, mathematicians and others deal with complexity. It covers both aspects of removing detail as well as the identification of generalisations or common features, and has been recognized as a crucial skill for software engineering professionals. There are many explanations for the importance of the notion of abstraction. Among others, the intangibility of software systems, the need to deal with complexity, and the ability to examine many topics in software engineering at different levels of detail according to the purpose, are presented as strong justification for the central role of abstraction. Keith Devlin [1] has stated the case clearly and concisely, "Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner."

ICSE'06, May 20-28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

In the workshop we explore the potential benefits, as well as the barriers, of using abstraction from cognitive, management and organizational perspectives and at the individual, team and organization levels.

2. THE CONCEPT OF ABSTRACTION IN COMPUTER SCIENCE AND SOFTWARE ENGINEERING
Abstraction is a central topic both in Computer Science (CS) and in Software Engineering (SE). It is 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. Abstraction is especially essential in solving complex problems as it enables the problem solver to think in terms of conceptual ideas rather than in terms of their details. Abstraction can be expressed in different ways. Liskov and Guttag [4], for example, describe abstraction in program development as "a way to do decomposition productively by changing the level of detail to be considered. ... The process of abstraction can be seen as an application of many-to-one mapping. It allows us to forget information and consequently to treat things that are different as if they were the same. We do this in the hope of simplifying our analysis by separating attributes that are relevant from those that are not". The decision as to what is essential and what is irrelevant depends on the *purpose* of the abstraction.

In addition to the use of abstraction, CS and SE practitioners should be aware of the need to think in terms of *different abstraction levels* and capable of moving between abstraction levels. For example, software developers should move from a global view of the system (high level of abstraction) to a local, detailed view of the system (low level of abstraction), and vise versa. More specifically, when trying to understand customers' requirements, developers have a global view of the application (high level of abstraction); whereas when coding a specific class, a local perspective (on a lower abstraction level) is adopted. Obviously, there are many intermediate abstraction levels in between these two levels, which programmers consider during the process of software development. The knowledge of how and when to move between different levels of abstraction does not, however, always come naturally, and requires some degree of awareness and experience. For example, a software developer may remain in too high (or low) a level of abstraction for too long, while the problem he or she faces could have been solved more readily had it been viewed at a different and more appropriate level of abstraction. The shift to that abstraction level might not

be a natural move unless one is aware that this might offer a step towards a solution [2].

3. WORKSHOP DESCRIPTION

The one-day workshop consists of theory- and practice-based presentations, group work and discussions. It examines the concept of abstraction from organizational, managerial and cognitive perspectives. It is envisaged that the workshop may contribute to the state of software engineering research, practice and teaching.

Workshop activities:

- Experience sharing with respect to case studies in which abstraction plays a central role: Participants discuss situations in software development in which thinking in terms of different abstraction *levels* may improve and enhance software development processes;
- Examination of different software engineering topics from the perspective of abstraction: Participants explore the relevance and *role* of abstraction and its contribution to software development processes;
- Exploration of how abstraction may foster organizational processes: Participants discuss heuristics, such as abstraction, that can guide the *performance* of different activities throughout the process of software development at the individual, team and organization levels;
- Evaluation of different ways to *teach* abstraction in software organizations and in academia: Participants explore how abstraction can be integrated as a core concept in different teaching frameworks.

4. WORKSHOP STRUCTURE

The workshop consists of three parts:

a. Abstraction in practice

b. Teaching abstraction

c. Understanding abstraction: How can we categorize abstraction for different purposes?

5. SUBMITTED PAPERS

The following papers will be presented in the workshop in order to facilitate the discussions:

a. *Abstraction-based Requirements Management* by Leah Goldin and Anthony Finkelstein

The paper outlines a method for impact analysis of requirements change and other requirements management activities using abstraction identification.

b. *Emerging Design: New Roles and Uses for Abstraction* by Christopher Van der Westhuizen, Ping H. Chen and André van der Hoek

The paper introduces the concept of dynamic abstraction (emerging design) that is able to fulfill new roles in the development process, such as coordination, detecting design decay, and project management.

c. *Increasing Quality of Conceptual Models: Is Object Oriented Analysis That Simple?* by Davor Svetinovic, Daniel M. Berry and Michael W. Godfrey

Based on teaching experience, this paper presents some insights into the construction of conceptual models, where the quality of the models is related to incompleteness and varying levels of abstraction.

d. *Cataloging Abstractions* by Spencer Rugaber

This paper explores the possibility of mounting a comprehensive effort to catalog abstractions. Related efforts, such as the design of textual and electronic dictionaries, are surveyed and a set of derivative questions is presented to explore the problem space.

6. WORKSHOP COMMITTEE

Daniel M. Berry, University of Waterloo, Canada

Shing Chi Cheung, HKST, Hong Kong

Yael Dubinsky, Technion, Israel

Anthony Finkelstein, University College London, UK

Carlo Ghezzi, Politecnico di Milano, Italia

Leah Goldin, Afeka - Tel Aviv Academic College of Eng, Israel

Shimon Schocken, Interdisciplinary Center (IDC), Israel

Tetsuo Tamai, The University of Tokyo, Japan

Jim Tomayko, Carnegie Mellon University, USA

Sebastian Uchitel, Imperial College London, UK

7. REFERENCES

[1] Devlin, K . Why universities require computer science students to take math, *Communications of the ACM*, **46** (9), Sept 2003, 37-39.

[2] Hazzan, O. The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software* **63**(3), 2002, 161-171.

[3] Kramer, J. Abstraction – the key to Computing?, *Communications of the ACM*, to appear.

[4] Liskov, B., Guttag, J. *Abstraction and Specification in Program Development*, The MIT Press, 1986.