Report Number: WUCS-86-18

1986-11-01

Data Engineering in Software Development Environments

Gruia-Catalin Roman

The design of a Software Development Environment (SDE) represents a very interesting point of contact between data engineering and software engineering. In this context data engineering becomes the cornerstone for successful software engineering practices. This paper attempts to bring about a better understanding of the difficulties associated with this task by considering sources of complexity in SDE design.

DATA ENGINEERING IN SOFTWARE DEVELOPMENT ENVIRONMENTS

Gruia-Catalin Roman

WUCS-86-18

November 1986

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

As appeared in Proceedings of the 3rd International Conference on Data Engineering, February 1987, pp. 85-86.

DATA ENGINEERING IN SOFTWARE DEVELOPMENT ENVIRONMENTS

Gruia-Catalin Roman

Department of Computer Science
WASHINGTON UNIVERSITY
Saint Louis, Missouri 63130

ABSTRACT

The design of a Software Development Environment (SDE) represents a very interesting point of contact between data engineering and software engineering. In this context data engineering becomes the cornerstone for successful software engineering practice. This paper attempts to bring about a better understanding of the difficulties associated with this task by considering sources of complexity in SDE design.

1. Introduction

A Software Development Environment (SDE) is a coordinated collection of computers and software tools dedicated to supporting the development and maintenance of software systems. The SDE, as we know it today, is essentially a highly sophisticated information system subjected to continual evolutionary pressures. For these and other reasons the design of a SDE is a major data engineering undertaking executed in the context of a very challenging software engineering exercise.

The SDE represents a very interesting point of contact between data engineering and software engineering. In this context data engineering becomes the cornerstone for successful software engineering practice. The challenges facing the data engineer are enormous. This paper attempts to bring about a better understanding of the difficulties associated with this task by considering some of the sources of complexity present in the design of a SDE.

Since there is no prototypical SDE, one way to approach this topic is to consider several different dimensions along which SDEs may be classified [1]. In the remainder of this paper we discuss three key dimensions and we examine the data engineering requirements originating with each of them.

2. SDE Dimensions

The dimensions we have selected are: organizational principle, mode of interaction, and accessibility to information.

Organizational principle. In our work we have identified several different orientations or organizing principles for software development environments:

Incongruous. An incongruous development environment does not have a single organizing principle.

Language based. A language-based environment concentrates its support on software written in a specific language, e.g., Ada® or Lisp. In its purest form, this type of SDE is merely a collection of tools that allow one to configure, edit, compile, run, and debug programs. Data engineering concerns are either peripheral or subordinate to the greater programming language issues.

Operating system based. A development environment may be based on the concepts and features of a particular operating system, e.g., UNIX®. In such SDEs tool kit integration via file system and interface design is the dominant data engineering concern.

Tool based. A tool-based environment is built around the concepts and capabilities of one or more key, highly versatile tools. An editor, a parser, or a database package may be the tool in question. The nature of the tool determines the role to be played by data engineering. A parser will enforce specification standards and check syntactic relationships while a database package will place the emphasis on modelling, representation, semantic consistency, query processing, etc. It is generally accepted, however, that database support is essential to a modern SDE.

Process model based. The central concept of this class of environments is a model of the series of *activities* required to create or modify a software system. Using the model, the environment can guide the software developer through the necessary steps, checking that the actions taken at each step are correct. A project management system is representative of this category of SDEs. (We know of no SDE in which this paradigm is dominant, but methodologies based on it do exist, e.g., rapid prototyping.) By necessity, a process based model involves plans, strategies, and historical records. As such, software life-cycle models must be integrated into the traditional project data management system. The latter must also be augmented by a sophisticated, evolving knowledge base.

Specification model based. This type of software development environment is organized around a formal semantic model of the *objects* created during the development or maintenance of a software system and the relationships among these objects. The environment can syntactically and semantically validate each object and cross-check it with related objects. The choice of an appropriate conceptual model is the difficult task here—few question its desirability.

Our own effort in this area has been directed toward the use of formal logic for the specification of Geographic Data Processing (GDP) requirements [2]. The emphasis was placed on modelling data and knowledge requirements rather than processing needs. A subset of first order logic was proposed as the principal means for constructing formalizations of the GDP requirements in a manner that is independent of the data representation. Requirements executability was achieved by selecting a subset of logic compatible with the inference mechanisms available in Prolog. Concepts that are important within the context of GDP (e.g., time, space, and accuracy) have been added to the formalization without losing Prolog implementability or separation of concerns.

Most SDEs do not fit exactly into one or the other of the categories listed above. What is important to note, however, is the fact that increased sophistication of the SDE organizational principle is accompanied by a commensurable increase in the importance and complexity of the data engineering concerns. This trend is reenforced by our analysis of the other two SDE dimensions.

Mode of interaction. The mode of interaction describes the level of user involvement with a software tool. We divide modes of interaction into four categories:

Static. In this mode of interaction, users are not involved during processing. They merely examine the tool output.

Dynamic. The dynamic interaction mode enables users to trace the internal processing actions while they occur as well as to examine the results of an activity. The static and dynamic modes do not place any special demands on data organization.

Interactive. An interactive mode enables users to guide a process by exchanging information with the process while it is active. The user and tool actions are viewed as separate. The data organization becomes visible via the constraints it imposes over the user/SDE interactions.

Synergistic. Synergistic interfaces transcend the capabilities of interactive interfaces to merge the user's actions with those of the computer. By merging the capabilities of tools with those of the human, the effectiveness of each is magnified. Increased data visibility and rapid access to large data volumes are required to support this mode of interaction. This places significantly greater demands on the data engineering tasks.

Accessibility to information. Although the level of sophistication built into the mode of interaction generally correlates with increased accessibility to data, different degrees of data accessibility may be considered for a given mode:

Localized. With localized access users have access only to information inside of their current contexts, e.g., to information contained in the file that they are currently processing. There is no formal organization or cross-referencing of related information from different contexts.

Structured. Structured access organizes the different information environments in some well-defined way (perhaps hierarchically). Although users may not have direct access to related non-local information, they can follow a well-defined path of context changes to access the information.

Total. Total access means that all related information is immediately available to the user regardless of the context in which it is specified. This includes both access to basic data stored by the SDE and logical inferences that could be reached starting with the basic data. The current trend is toward total access and is reenforced by attempts to merge data and knowledge bases.

3. Conclusions

The idea that software engineering and data engineering have a very close relationship is self-evident. This paper points out that the success of modern SDEs rests heavily on data engineering and that this reliance increases with the the sophistication level of the SDE.

4. References

[1] Roman, G.-C. et al. "Long-Range Technological Impact on Computer-Aided Product Development at DMA," *Final Report*, RADC/DMA Contract F30602-83-K-0065, 1986.

[2] Roman, G.-C. "Formal Specification of Geographic Data Processing Requirements," *Proceedings of the 2nd International Conference on Data Engineering, (Outstanding Paper Award)*, pp. 434-446, February 1986.