Science of Computer Programming ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure

John Hutchinson[*], Jon Whittle, Mark Rouncefield

School of Computing and Communications, Lancaster University, UK

HIGHLIGHTS

- We present extensive results from a survey of MDE practices in industry.
- We present case studies of the adoption of model driven engineering (MDE) by four companies.
- We identify important factors that can affect the success or failure of MDE use from both the survey and case studies.
- MDE provides genuine benefits to those companies who use its appropriate contexts.
- Success/failure appears to be more dependent on organizational factors than technical.

ARTICLE INFO

Article history:
Received 7 March 2012
Received in revised form 26 March 2013
Accepted 27 March 2013
Available online xxxx

Keywords:
Model driven engineering
Empirical software engineering
Industry practice

ABSTRACT

In this article, we attempt to address the relative absence of empirical studies of model driven engineering (MDE) in two different but complementary ways. First, we present an analysis of a large online survey of MDE deployment and experience that provides some rough quantitative measures of MDE practices in industry. Second, we supplement these figures with qualitative data obtained from some semi-structured, in-depth interviews with MDE practitioners, and, in particular, through describing the practices of four commercial organizations as they adopted a model driven engineering approach to their software development practices. Using in-depth semi-structured interviewing, we invited practitioners to reflect on their experiences and selected four to use as exemplars or case studies. In documenting some details of their attempts to deploy model driven practices, we identify a number of factors, in particular the importance of complex organizational, managerial and social factors – as opposed to simple technical factors – that appear to influence the relative success, or failure, of the endeavor. Three of the case study companies describe genuine success in their use of model driven development, but explain that as examples of organizational change management, the successful deployment of model driven engineering appears to require: a progressive and iterative approach; transparent organizational commitment and motivation; integration with existing organizational processes and a clear business focus.

1. Introduction

The complexity and pervasiveness of software in society is growing exponentially [13]. It is generally agreed that the only realistic way to manage this complexity, and to continue to provide software benefits to the public at large, is to develop

[*] Corresponding author. Tel.: +44 1524 510311; fax: +44 1524 510492.
E-mail addresses: johnhutchinson.uk@gmail.com, j.hutchinson@lancaster.ac.uk (J. Hutchinson), j.n.whittle@lancaster.ac.uk (J. Whittle), m.rouncefield@lancaster.ac.uk (M. Rouncefield).

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.03.017

software using appropriate methods of abstraction [20]. Today, the state-of-the-art in software abstraction is model-driven engineering (MDE) – that is, the systematic use of models as primary artifacts during a software engineering process[1]. MDE has recently become popular in both academia and industry as a way to handle the increasing complexity of modern software and, by many, is seen as the next step in increasing the level of abstraction at which we build, maintain and reason about software. MDE includes various model-driven approaches to software development, including model-driven architecture, domain-specific modeling and model-integrated computing. This article is concerned with describing and understanding the industrial experience of MDE and identifying any best practice or lessons learned.

Whether or not the current brand of MDE tools succeeds, the notion of abstract models is crucial to the future of software. But empirical evaluations are needed to ensure that future software tools will match the way that software developers think. Although MDE claims many potential benefits – chiefly, gains in productivity, portability, maintainability and interoperability – it has been developed largely without the support of empirical data to test or support these claims [4]. As a result, decisions whether or not to use MDE are based mainly on expert opinion rather than hard empirical data; and these opinions often diverge [30,31] as companies tend to adopt MDE based not on an analysis of how it will affect their business but on perception or the advice of evangelists. The lack of empirical results on MDE is a problem since industry invests millions in the development and application of MDE tools [12]. Without empirical evidence of the efficacy of these tools, there is a danger that resources are being wasted.

The benefits of MDE are frequently cited and are often considered to be obvious. However, there are also reasons why MDE may in fact have a detrimental effect on system development. Firstly, it is by no means guaranteed that higher abstraction levels lead to better software. In fact, results from psychology generally [19,5] and psychology of programming specifically [29] show that abstraction can have a negative effect because thinking in abstract terms is hard, with a tendency for individuals to prefer concrete instantiations (e.g., exemplars, simulations) over abstract conceptualizations.

Secondly, MDE involves dependent activities that have both a positive and a negative effect. For example, code generation in MDE appears, at first glance, to have a positive effect on productivity. But the extra effort required to develop the models that make code generation possible, along with the possible need to make manual modifications, would appear to have a negative effect on productivity. How the balance between these two effects is related to context, and what might lead to one outweighing the other, is simply not known. Thirdly, there are many different flavors of MDE and so companies have a difficult choice to select the right variant for their business. The benefits and drawbacks of MDE are not obvious and, in fact, may be highly dependent on context. As a result of these factors, there is as yet no clear decision-making framework that can tell whether MDE will succeed or not in a given context. Our research is intended to provide a better foundation for MDE adoption by trying to understand empirically which factors lead to successful adoption of MDE and cataloging exactly what works in MDE projects [17,18].

2. Previous work

There has not yet been a systematic and multidisciplinary programme of work to study the effectiveness of MDE in broad terms. For example, there are currently no widespread, systematic, surveys of industrial use of MDE. These can be important because they often reveal common misperceptions [13]. A 2005 survey looked at the penetration of UML and UML tools into the marketplace [10] but focused on UML not MDE. Forward and Lethbridge [10] conducted an online survey of software practitioners' opinions and attitudes towards modeling. Afonso et al. [1] documented a case study to migrate from code-centric to model-centric practices. But there has been very little research that examines the social and organizational factors related to MDE adoption and use since most empirical studies have concentrated on technical aspects of MDE. The Middleware Company conducted two studies, commissioned by Compuware in 2003 and 2004, that measured development and maintenance times of an online pet store using both MDA and a state-of-the art IDE [24,25]. The studies compared only one MDE tool (Compuware's OptimalJ) but showed a 35% increase in productivity and a 37% increase in maintainability. Anda et al. [2] reported anecdotal advantages of modeling such as improved traceability but also pointed to potential negatives, such as increased time to integrate legacy code with models and organizational changes needed to accommodate modeling.

There has been more research on evaluating the language UML [3,6,21,26]; studies attempting to empirically measure the comprehensibility of UML models [7–10,14,22–24,28]; and a range of experiments that assess software engineering techniques incorporating UML. But MDE is more than just UML. UML is just one example of a modeling language but MDE additionally incorporates the notion of multiple modeling languages (each for a different domain), the idea of automating model transformations between modeling languages, and the principle of maintaining multiple perspectives of a project (e.g., platform-specific vs platform-independent). Whilst most existing work has concentrated on evaluating the appropriateness of the UML language, almost nothing has been done to examine the wider issues of MDE: such as whether code generation works in practice; what are the trade-offs between domain-specific languages and general purpose languages; is MDE appropriately aligned with existing organizational structures?

[1] This description of MDE as "the systematic use of models as primary artifacts during a software engineering process" is essentially the definition of MDE that was used for the work described here. Although others may attempt a more formal definition, and then use that to determine if the practices observed constituted MDE, the goal here was to be as inclusive as possible in our attempt to discover what practices were actually occurring in industry. Thus our notion of MDE arises from what is being practiced by those who took part in our study.