Abstract

We would like to understand the interests of our target audience. Please register at **www.softmetaware.com/whitepapers.html** to provide us with some information about yourself, and to obtain access to the full content of all SoftMetaWare white papers.

Model-Driven Architecture Implementation & Metrics

Version 1.1
28 August 2003

Jorn Bettin
jorn.bettin@softmetaware.com
http://www.softmetaware.com

Copyright © 2002, 2003 SoftMetaWare Ltd.
SoftMetaWare is a trademark of SoftMetaWare Ltd.
All other trademarks are the property of their respective owners.

Abstract

A small-scale example application is used to highlight the potential of Model-Driven Architecture. This paper compares traditional software development (no abstract modelling), standard Unified Modelling Language (UML)-based software development, and MDA-based software development. To allow a comparison, Lines of Code (LOC) serve as a measure of development effort, and Atomic Model Elements (AME) are introduced to measure the modelling effort.

1. Introduction

In the 21st century software is pervasive, it is part of mobile phones, PDAs and other devices. Software is also embedded in a less noticeable way in cars, heating systems, household appliances such as washing machines, and so on. The software industry has become one of the largest industries on the planet, and many of today's most successful companies are organisations built around the production of software and related services.

Looking at the history of other industries, such as the automobile industry or the aviation industry, and their respective level of maturity half a century after their inception, one could assume that the production of software must surely be a highly standardised process. However this is not the case. Instead, most software that is developed today is produced with tools and processes that are not too different from what was used three decades ago. Of course, numerous technology waves have rolled over the industry in the past 30 years, but the last major step forward was the event of general purpose programming languages and compiler technology. Similarly the flavour of software development methodologies has changed about every five years or so, without having an impact on the foundations of software development. The primary work product of a programmer is still textual source code, and textual source code is still the ultimate specification that is translated into operations that a computer or a virtual machine can execute.

Is the production of software an art form or an engineering discipline? That there is no universally accepted answer to this question points to some of the deepest problems in the industry. Many individuals and whole companies have resigned to the facts of life: developing software is expensive, unpredictable, and stressful for those involved.

If software development were an engineering discipline, there would be industry-wide accepted, thoroughly tested methodologies (and the metrics to prove it) and robust end-to-end development tools to support these methodologies. We claim that already yesterday's tools were sufficient to transform the production of software into an engineering discipline. Most organisations have been side-tracked by keeping pace with the constantly changing set of implementation technologies. The generic skills and knowledge that are required for building large volumes of high quality software were not part of the equation. Instead of hoping for the next product release form the large vendors that provide the infrastructure plumbing (operating systems, integrated development environments, databases etc.) to solve productivity problems, organisations should rather nurture and grow their specialised domain knowledge and apply this knowledge to streamline their software production process.

There are many examples of organisations that actually do produce software economically and to predictable schedules, only these organisations currently still constitute a minority. Given the economic pressure to produce more software, in shorter timeframes, with fewer people, this situation is about to change. Abstract modelling techniques and template-language based code generators allow an organisation to leverage its specific domain knowledge and to speed up software development in that domain by an order of magnitude.