MODEL DRIVEN DEVELOPMENT

Strategic advantage in software development

INTRODUCTION

Model driven development (MDD) is a software development approach that aims at creating software from domain-specific models. An MDD strategy involves tools, but it is more than just using tools. In MDD strategy, tools are more than clever, nice-to-have side efforts to the development process. Rather, tools are a conscious strategic effort that is integral to the software development process. MDD strategy embraces tools and meta-programming to achieve strategic objectives and strategic advantages in software development.

Elements of MDD

There are many terms similar to *model driven development*, to include: domain analysis, meta-modeling, model-driven generation, template languages and domain-driven framework design. Some of these terms are a synonymous while other terms indicate a focus on a particular aspect of MDD.

MDD begins with a domain model, a conceptual description of the components of a category or domain of systems. A domain can be any sphere of knowledge or expertise, such as life insurance, equity trading, aircraft manufacturing or hospital patient management. A model is a description of something to be made. Domain models use the language, the terms, phrases, nouns and verbs, specific to a sphere of expertise. The domain model can also be called a meta-model, because it is the model of models. That is, the domain model is the basis of all the subsequent models that developers create.

MDD must have modeling tools. Without tools to increase productivity, MDD would be a burdensome effort rather than a strategic advantage. These tools are based on the domain model. Three most common types of tools are:

1. Graphical modeling tools
2. Domain specific languages (DSL)
3. Code generators

Graphical modeling tools enable developers to draw diagrams, boxes and arrows usually, to visually express models of systems. A domain specific language (DSL) enables developers to express models of systems in text that is easily understandable by humans. Depending on the application, MDD may use diagrams or DSL or both together. Code generators work from the graphical and DSL models to create functioning software code. With MDD, developers are writing code that writes code.

The Eclipse open source community

The Eclipse open source community is not just another integrated development environment (IDE). Prior to Eclipse, developers might use an IDE or a command line text editor. Now the IDE is no longer optional. Eclipse has become an integral part of the software development process. Because Eclipse is so easily customized, development teams are customize Eclipse to integrate with their development process. They are also unconsciously customizing their development process to integrate with Eclipse. As such, it is changing the software development process.

MDD is quickly moving from a novel concept to a pragmatic business necessity in large corporations. The change is due in no small part to the advent of the open-source Eclipse project. Prior to Eclipse, creating tools was an expensive proposition. One needed either to build the entire tool set from scratch or to commit to a particular vendor's proprietary solution. Neither choice was strategically sound. Now, with Eclipse, the best tool platform available on the market is free and open source. More importantly, it is designed with the specific intent of extending and customizing. With Eclipse, creating tools is relatively inexpensive.

STRATEGIC OBJECTIVES

There are several common strategic objectives for using MDD. The seven most common objectives are as follows:

1. Lower the overall cost of building large internal applications
2. Speed time to build large applications
3. Lower the risk of large applications
4. Simplify development
5. Lower the required skill level needed to work on large applications
6. Expand the pool of resources that can work on large applications
7. Leverage open source

The first three objectives should sound exceedingly familiar. What organization does not claim to want lower costs, faster delivery and lower risk? The other four objectives may sound new, even controversial. Some software professionals take exception to the objective of lowering the skill set necessary by contributors. Nevertheless, these objectives are inherently interdependent.

Figure 1 shows a conceptual graph of the interdependencies of these seven strategic objectives. The business strategist Michael Porter used these sorts of diagrams to show how certain business processes gained strategic advantage over others. In order to replicate the strategic results, a competitor must replicate the entire interdependency of processes. The winning company becomes as strong as its *strongest* link.

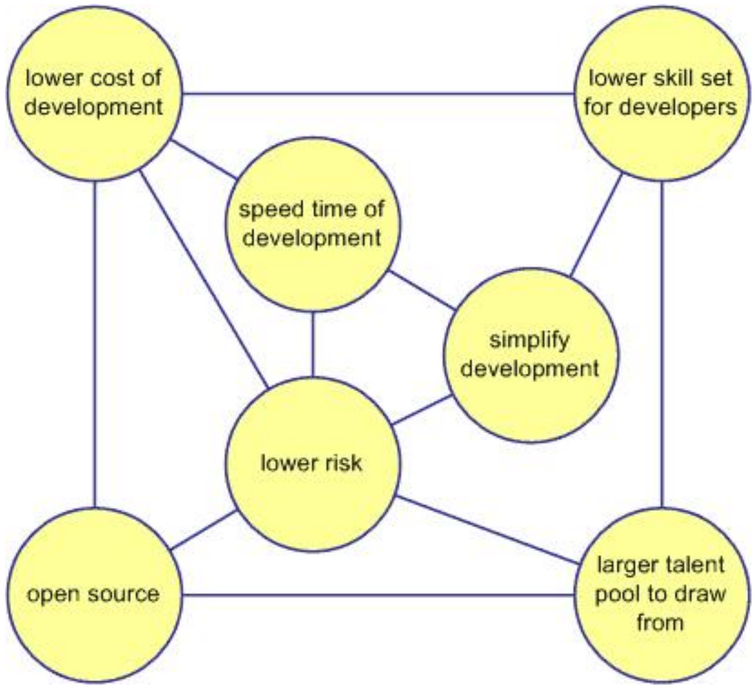
lower cost of development
lower skill set for developers
speed time of development
simplify development
lower risk
open source
larger talent pool to draw from

Figure 1 Seven objectives of model driven development

Using open source lowers the risk of development because the code is peer reviewed and developers can change critical bugs themselves if necessary. Using open source provides a large talent pool to draw from because many developers have experience with the open source or they can quickly learn it. Open source is free, so there is a direct cost savings.

Having a larger talent pool to draw from lowers risk because the company can replace critical developers if necessary. Lowering the skill of developers directly increases the pool of talent to draw from. The simpler something is to learn, the more people there will be that have learned it or can learn it. In a free market, supply and demand dictate that the greater the supply the lower the cost. Thus, lowering the required skill set of developers lowers the cost of the developers and thus lowers the overall cost of the project.

Tools should simplifying development. If development is simpler, then development requires a simpler or lower skill set. Simplifying makes work more predictable and thus lowers risk. Simple processes are quicker to execute, so simplifying development speeds the time of development.

Speeding the time of development lowers the cost because cost is directly attributed to work hours. Speeding development also lowers risk because the team has more time available to react to problems and changes.

STRATEGIC ADVANTAGES

In pursuing the seven strategic objectives of MDD, organizations achieve six identifiable advantages. The implementation of MDD:

1. Enables scripters to contribute to enterprise development
2. Reduces both direct and indirect development efforts
3. Increases the flexibility of reassigning team members
4. Enables task-oriented management of development
5. Enables advanced developers to focus on the hard stuff
6. Manages complexity by displacing it to appropriate points.

These six advantages are concomitant with pursuing the seven objectives. Like the strategic objectives, the strategic advantages of MDD are interdependent. Figure 3 shows conceptually how these six advantages reinforce one another.

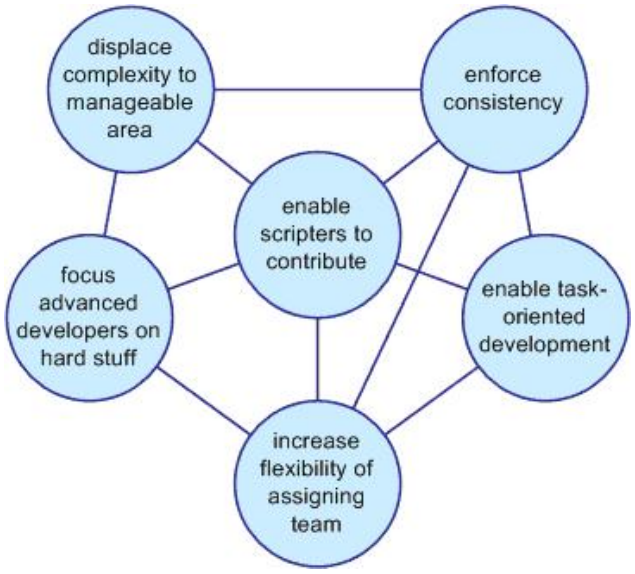
displace complexity to manageable area
enforce consistency
enable scripters to contribute
focus advanced developers on hard stuff
enable task-oriented development
increase flexibility of assigning team

Figure 2 Linkages between tool advantages

We will discuss each of these advantages in sequence.

Expand the talent pool

Consider the available software talent pool as an inverted pyramid like the one shown in Figure 3. The higher tiers are dependent upon the contributions of the lower tiers. The higher one goes up the inverted pyramid, the more common and less expensive the skill level. The lower one goes down the pyramid, the less common and more expensive the skills. At the bottom are hard core scientists who build compilers and operating systems. Next up are engineers who understand how to build enterprise strength software. Higher up are developers who are competent programmers, but do not necessarily understand or need to understand advance concepts of concurrency, transactions and such. At the top of the inverted pyramid are scripters who can write XML, JavaScript, HTML and Java methods.

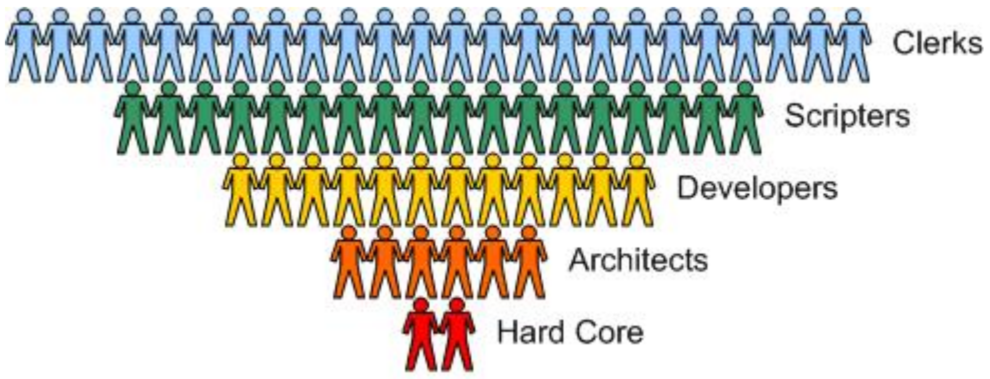

Figure 3 The inverted pyramid talent pool

The supply of software talent becomes tighter with each year. Companies demand more and more automation. The talent pool is growing, but not growing as fast as demand. We have to import talent into the United States. We have nearly consumed the capacity of India to the point where it is no longer the cost advantage it once was. We have expanded off shoring into Eastern Europe, Southeast Asia and China.

MDD enables us to move development effort higher on the inverted pyramid. Through MDD, scripters can contribute to enterprise software development.

Reduce coding efforts

MDD generates hundreds, sometimes thousands of lines of code through the automated application of templates to models. There are immediately obvious savings to generating redundant code rather than typing it out by hand and debugging mistakes.

What is not immediately obvious is how much is saved in indirect efforts. The templates define much of what would otherwise be arbitrary decisions that would have to be made by individual developers. MDD eliminates a great deal of meetings between developers to discuss how an interface is implemented, how it will be named, and how it will integrate with other interfaces. As one senior engineer stated,

> “The game is not the tool; it’s the consistency.”

MDD enforces consistency. Arbitrary decisions that have to be made are made once and then replicated.

Reassign team members

The code consistency provided by MDD enables developers to cooperate more easily and readily.

- First, much of the significant information is stored in the models, which provide an abstract representation that is easier to process and understand.

- Second, one developer can easily navigate another developer's code, because it is organized in the exact same manner as his own.
- Third, exceptional code such as critical business logic is placed in predefined locations that are easily located by the developers.

All these factors lead to the flexibility of reassigning team members. One developer can pick up where another left off due to vacation or sick leave. Extra developers can be moved to sections of code that are falling behind or have expanded requirements.

Agile methodology

Agile development methodology relies on sound tasking of developers. The development team must be able to break down work into discrete tasks with specific estimates of effort. If you do not break down the tasks well, then you cannot manage well.

MDD helps with task break down in three ways.

- First, MDD facilitates working top-down. One model leads to sub-models which lead to artifacts.
- Second, the model components represent a predictable set of artifacts that are generated or customized.
- Third, MDD enforces greater consistency that enables direct comparison of time and effort across tasks.

In one project I saw, the MDD tools were wizard based. The wizards walked the developer through the questions and then generated a dozen or more artifacts. The wizard also generated a task list of things that the developer had to complete by hand. How hard would it be to have that task list feed into an agile software management tool?

Focus on the hard stuff

The initial reaction of many advanced developers to MDD is negative. Some see it as a threat to their job. Some see it as "dumbing-down" their work. Some see it as tying their hands and limiting what they can do. Negative initial reactions soon dispel as the advanced developers begin to use the tools.

MDD tools accelerate development and increase productivity for advanced developers as well. In MDD, advance developers do far less repetitive typing of code. They focus instead on the creative aspects of their work. Most advanced developers end up serving one or more of the following roles in an MDD effort:

- Building the tools themselves
- Modifying the framework to implement new patterns
- Mentoring junior developers
- Serving as a SWAT force, taking on the particularly hard tasks

Far from a negative experience, MDD makes the work of advanced developers even more rewarding. It also makes their work more valuable to the organization.

Manage complexity

Hurst's Law states:

"Complexity can neither be created nor destroyed; it can only be displaced."

One might also call Hurst's Law the Law of Conservation of Complexity. A corollary to Hurst's Law would be, "Pay attention to where you are displacing the complexity."

Hurst's Law is very important in information technology where new technologies claim to "greatly simplify" a complex business problem. These technologies are not making the problem simpler; they are instead displacing the complexity from one place to another. The new technology is beneficial if it displaces complexity to a point where it is more manageable than it was before.

If an organization does not pay attention to Hurst's Law, then its information technology efforts can devolve into a game of "whack-a-mole". The team pushes down on a problem here only to have a new problem pop up over there.

MDD works because it inherently embraces Hurst's Law. The pattern developers solve the complex part once and place it in the shared framework. The tool developers identify the best practice once and incorporate it into the tools. The rest of the developers gain access to these solutions automatically through the generated code.

CONCLUSION

MDD strategy embraces tools and meta-programming to achieve strategic objectives and strategic advantages in software development. We have discussed seven common strategic objectives and six common strategic advantages that MDD achieves. We have also discussed how these objectives and advantages are interdependent and reinforce one another.

One should not ignore the importance of the open-source Eclipse project in achieving the advantages of MDD. Without Eclipse, large companies would have to either build the whole tool set themselves or tie themselves into a vendor's proprietary solution. Without Eclipse, creating the tools for MDD would either be cost prohibitive or strategically impractical.

The strategic objectives and advantages of MDD are very compelling. Many large companies have already begun to deploy applications built using MDD techniques. The knowledge of the success from these projects is spreading. With each new success, MDD proves itself to be not only a novel concept but also a pragmatic business necessity.