Pathfinder Solutions

Platform-Independent Model Driven Development

PI-MDD Executive Summary

Version 1.1

January 13, 2014

Pathfinder Solutions
www.pathfindersolns.com
+1 508-568-0068

Table Of Contents

Executive Summary

Platform-Independent Model Drive Development (PI-MDD) is an integrated agile, test-driven and model-driven approach to architecting, developing, testing and deploying complex, high-performance systems software. Two decades of constant refinement by hundreds of development teams and dozens of technology and expertise vendors spanning a range of challenging industries have optimized this method and its supporting automation to deliver the highest performance and best quality systems while boosting key business results. Customer-reported project metrics show the PI-MDD approach has yielded substantial, repeatable business-level gains including a doubling of productivity and a 10X quality gain within 14 months of adoption (as compared with common hand-crafted coding approaches).

Introduction

The development of complex and high-performance software systems is a challenging endeavor. Meeting both technical and business needs is increasingly more difficult, and teams applying classic hand-crafted software development approaches more frequently fail to deliver these required results that those that achieve even qualified success. Simplistic "just code it" approaches relying on a handful of talented and motivated artisans can work well for small, short duration efforts. But typical fruits of this success – team growth and product opportunity – quickly inflate software teams and projects beyond the point where long hours and good intentions can make up for ambiguous architecture and inconsistent engineering.

Model Driven Development (MDD) approaches use software modeling languages and tools to help organize software and improve team communication. However not all MDD approaches are able to meet the challenges faced in building complex and high-performance systems. The Platform-Independent MDD (PI-MDD) method is an approach incorporating key modeling techniques and automation specifically designed to meet the unique challenges of building these types of systems. In addition to its technical advantages, PI-MDD has been specifically tailored to boost business results including system quality and productivity. Leveraging best practices and industry standards evolved over decades, PI-MDD is a refined set of mutually-supporting architectural and detailed modeling techniques applying and enforcing key disciplines through automation. Delivering efficient and capable systems quickly, PI-MDD also delivers substantial practical organizational improvements, with newly adopting teams posting substantial gains in critical business results including time to market, productivity and overall product quality within a year of adoption.

For organizations already working towards more disciplined practice and applying MDD technologies, moving to PI-MDD may not be a far step. However for many organizations still producing most of their software through a less structured hand-coding culture, moving to PI-MDD can be more of a paradigm shift, requiring explicitly staged adoption steps facilitated by training and expert mentoring.

The Needs

Technical

Simply addressing the technical challenges in building complex, high-performance software systems is difficult. Feature complexity, the challenge of deploying to complex target environments, and the need to build flexible and adaptable systems that meet all feature and performance requirements require sound techniques to manage complexity and produce efficient implementations. Many teams face further challenges in extending the time horizon beyond a single project to consider longer term agility to in the face of changing requirements and volatile markets, and ultimately the need to deliver families of products.

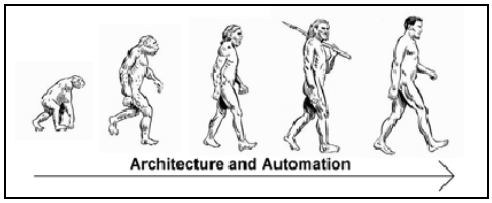
Business

The biggest change of the past few years has been a heightened need to dramatically boost business results delivered by software development organizations. Facing increasingly competitive markets, products must get to market much quicker with higher productivity and overall systems quality. Today it is not enough to simply deliver the system – it must be delivered on time, within budget and will full capability at high quality.

Methodology History

Compilers

Since the dawn of complex software systems, the industry has been continually working to manage complexity and simplify individual system elements. Two key principles were quickly proven effective: Separation, where the complex whole is broken down into simpler parts; and Abstraction, where focus is more on the problem being solved and less on the implementation details of that solution. Effective compilers for FORTRAN, C and other languages quickly showed how the right technology can make the most of these sound techniques.



Architectural Focus and Automation evolve with the rise in Level of Abstraction and Separation

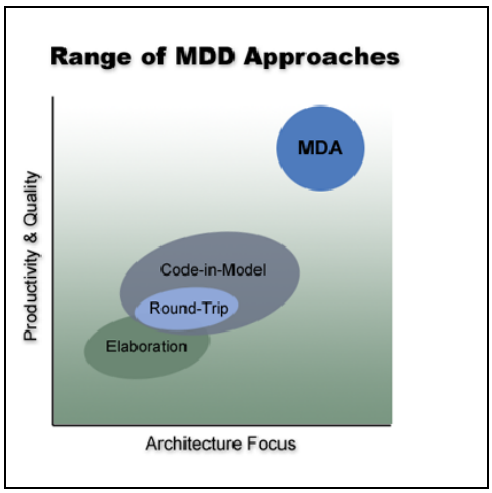
Early Modeling

While the use of more abstract textual programming languages made gains relative to machine and assembly programming, the pursuit of further gains pointed to graphical languages and more specific and refined techniques for building systems. By the late 1980's a number of methodologies based on graphical software modeling notations posted significant successes in challenging industry segments. Object Modeling Technique (OMT) and Shlaer-Mellor Object Oriented Analysis and Recursive Design (OOA/RD) were prominent among several competing approaches. Tools grew

up around the capture of these specific modeling languages, and more disciplined approaches - like OOA/RD – through their foundational emphasis on Separation and Abstraction, facilitated early technologies that automatically generated full implementations from models.

Modeling Language Standards and Automation

Increasing adoption of modeling quickly drove the development of an industry modeling language standard: the Unified Modeling Language (UML). Subsequently the Model Driven Architecture (MDA) set of standards were built to augment UML with standard ways to share model information and transform it to other forms, including code. Now a broad base of modeling tools and other technology grew up supporting the MDA standards, and advanced automation around model checking, report generation and code generation began to evolve.



PI-MDD is based on MDA Standards, boosting business results through higher Level of Abstraction and greater architectural focus.

Agile Methods and PI-MDD

Agile methods have been mitigating risk and shortening schedules for over a decade. With a focus on customer-centric sprint planning and short develop/test/integrate cycles, Agile approaches carefully and continually manage team focus, explicitly managing risk and maintain the pace of development – velocity. These disciplines combined with the continuous integration and testing of small, proven increments deliver better results than an old-school waterfall project organization. PI-MDD's executable models and high level of deployment automation provide a natural architectural and implementation base for an Agile culture.

A key element of PI-MDD that makes it particularly well suited for Agile development is the short path to testability with a PI-MDD model. With UML-level action language and platform independence, the PI-MDD model is complete and executable. Model development in rapid, automated develop/test cycles form an ideal nucleus capability for Agile sprints.

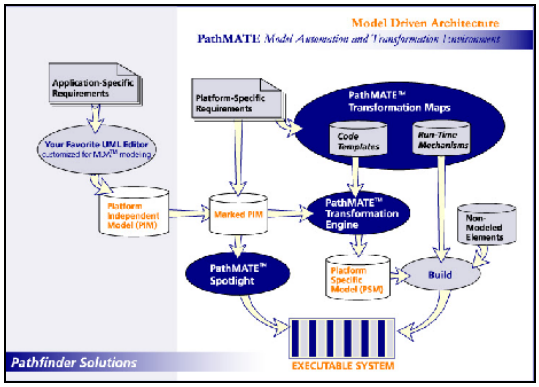
Test-Driven Development

Test Driven Development (TDD) is an Agile-based approach to building complex system where unit test (and in some cases inter-component integration tests) are built in advance of the product software and are used exercised upon component implementation. The immediate exercise of these tests provides a rapid, objective assessment of the readiness of the new capability, and dramatically improved component quality and reduces integration time and risk.

With PI-MDD code the openness of the underlying code generation is used to automatically generate test a framework and automation, making the creation and maintenance of these test far less expensive in terms of time and effort. This automation also promotes better test coverage and builds an automated regression test base, ultimately providing a quantum boost in overall system quality. PI-MDD's platform independence – facilitating component testability - and flexible code generation make TDD a natural progression of PI-MDD practice.

Standards-Based Industry Best Practice

With a standard language in UML and a wide range of modeling techniques evolving throughout the industry, coherent and complete methodologies matured and were applied across varying tools and in varying industry segments. Those producing highly complex and high-performance systems found greatest success when Separation and Abstraction could be facilitated through automation. Focusing on a key subset of the UML with an automation-based path to implementation, the Executable UML family of approaches (aka xUML and xtUML) emerged from the OOA/RD community, continuing the emphasis on Separation and Abstraction. The broadest and most flexible of these standards-based methodologies is Pathfinder Solutions' PI-MDD:



PI-MDD automation produces optimized code using open and standards-based technology.

Next evolutionary steps in the standards area involve Foundational UML (FUML), bringing together vendors and practitioners with tightened standards and a reference implementation. This continued refinement of technical standards best practices feeds steady improvement into PI-MDD and its supporting technologies.

With a history of thousands of successfully fielded systems in service for over two decades, the Platform Independent MDD approach has been proven to be effective in critical industries including medical devices, aerospace, defense, and automotive. It is the most effective way to build complex and high performance software today.

Steps to Adopt PI-MDD

How does an organization adopt PI-MDD in a way that manages the risks of change while still moving decisively toward the goal? They key is to establish a clear path with incremental steps, combining careful review and introspection, enlisting the support of proven experts and automation, and decisive steps.

Detail Business Context/Needs

The first step is simply to understand your business context. What are the technical and business challenges faced by your systems engineering and software development teams? What are their relative priorities? Often times critical business needs are those faced by most organizations:

- Time to market
- Overall cost to develop product
- Changing product requirements
- Shifting markets
- A drive for improved testing and test automation
- Productivity
- Overall systems quality

Assess Suitability of PI-MDD to Address Top Concerns

Once the organizational challenges/needs have been cataloged, match these with key strength areas of PI-MDD:

- Flexible and durable software architecture
- A consistent and well-documented development process
- Manage overall problem space complexity
- Ease deployment and manage target environment complexity
- Direct facilitation of development team communication and coordination
- Agile approach with short build/integration cycles
- Strong foundations for software development testing, system integration or and testing
- Self-optimizing implementation generation for high system performance in the target environment

If there are one or more priority challenges that can be addressed by PI-MDD, then move decisively to gain practical experience with the approach and its automation.

Explore PI-MDD with a Pilot Project

Scope out a limited PI-MDD project addressing a limited-scope but real development effort. Use experts and trained staff, and measure key metrics to build a base of

objective information to decide how to move next. Successful pilots share these characteristics:

- Modest and clear technical goals, set in a limited time frame
- Explicit and objective evaluation criteria
- Relevant to the overall product context, usually building a small piece of an actual system
- Staffed with a small team of motivated and capable engineers
- Relevant training and mentoring/guidance from experts
- Use of proper tools/automation

Apply on a Real Project

Quickly after the pilot effort, refine your approach to better address your unique needs, and apply PI-MDD on a full-scale project. Again mitigate cultural and technology risks with training and mentoring from industry experts, and apply proven automation technology.

Like the pilot, clear goals/requirements and success criteria within short, iterative and incremental builds provides essential focus. Proper training, mentoring/guidance and tooling are essential for maximizing team productivity and system quality.

Like the pilot, the first "real project" must also have a clear, limited timeframe, and a focus on harvesting key lessons learned so broader adoption can move quickly and effectively.

Pivot Your Organizational Culture

Once your first full-scale project effort establishes and adapts PI-MDD to the unique needs of your organization, leverage your newly earned expertise and new architectural foundations by rolling these forward via shared components and product line architecture.

Lessons learned from the pilot and first real project efforts are fed into essential plans for rolling out PI-MDD throughout the organization where business benefits can be derived. Making the paradigm shift includes careful planning and sequencing of:

- Product Line Architecture
- Selecting Product Development efforts for PI-MDD
- Team Enablement, including methodology training and mentoring
- Building, managing and deploying common components

Summary

Platform Independent Model Driven Development is a software development method with supporting automation that can deliver complex and high performance software systems effectively and with high quality. It also delivers substantial gains in business results, with newly adopting teams posting substantial gains in critical business results including time to market, productivity and overall product quality soon after adoption. For many organizations still producing most of their software

through a less structured hand-coding culture, moving to PI-MDD can be a substantial paradigm shift, requiring explicitly staged adoption steps facilitated by training and expert mentoring.

PI-MDD is the most effective way to build complex and high performance software today.

References

Lahman, HS. Model-Based Development: Applications. Addison-Wesley, 2011.

Raistrick, Chris et al. Model Driven Architecture With Executable UML. Cambridge University Press, 2004.

Douglass, Bruce Powel. Real-Time UML Third Edition: Developing Efficient Objects for Embedded Systems. Addison-Wesley, 2004.

Fowler, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition. Addison-Wesley, 2003.

Kleppe, Anneke, Jos Warmer, Wim Bast. MDA Explained The Model Driven Architecture: Practice and Promise. Addison-Wesley, 2003.

Mellor, Stephen J. and Marc J. Balcer. Executable UML: A Foundation for Model-Driven Architecture. Addison-Wesley, 2002.

Rumbaugh, James, Ivar Jacobson, and Grady Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

Starr, Leon. Executable UML How to Build Class Models. Prentice Hall, 2002.

OMG Systems Modeling Language (OMG SysMLTM) Specification, OMG document ptc/06-05-04, May 4, 2006

Pathfinder Solutions whitepapers (various) , http://www.pathfindersolns.com/resources/whitepapers