MDA Journal

January 2005

David S. Frankel
David Frankel Consulting

df@DavidFrankelConsulting.com

MDA Journal's origins lie in my desire to define the relationship between MDA on the one hand, and business process management and improvement approaches on the other. That is why I am happy that MDA Journal was able to make Business Process Trends its home. Since the Journal's inception, a number of articles have indeed focused on this relationship, but recently most of the articles have been more about MDA itself.

This month, Mike Guttman, a frequent contributor, returns MDA Journal to its roots by examining the relationship between MDA and Six Sigma, the business process improvement approach that has taken the corporate world by storm. Mike makes a compelling case that, in today's world where business processes rely on software, Six Sigma practitioners cannot afford to ignore the need for discipline and quality control in software development. He also argues that a model-driven software development lifecycle is a promising avenue for producing software whose quality satisfies Six Sigma's exacting requirements.

This article will probably spur some lively discussion about the relationship between MDA and Six Sigma. Let the conversation begin…

—David Frankel

MDA Journal

January 2005

Michael Guttman
Director
OMG's MDA FastStart Program

Six Sigma in a Nutshell
Six Sigma and Software Development
Six Sigma and the SDLC
Six Sigma and MDA
Transitioning to an MD-SDLC
Conclusion

Sidebar 1. Six Sigma's DMAIC
Sidebar 2. Assessing the Financial Benefits of an MD-SDLC
Sidebar 3. Integrating MDA and Six Sigma -- An Example

Six Sigma and MDA: *The Next Dynamic Duo?*

One of the main virtues of MDA is its ability to integrate with and support other technologies and methodologies. One example that is already very familiar to readers of this journal is how MDA can be used to enable BPM. MDA can be used to improve significantly the way in which those elements of business process models that require automation are transformed into software. MDA can thus reduce the risks and costs associated with actually implementing many BPM components.

In this article, we will be examining another important methodology—Six Sigma—and how MDA can be used to support it, albeit in a somewhat different way than BPM. Certainly anyone in corporate America not hiding under their desk for the last few years has heard more than an earful about Six Sigma. Today it ranks as one of the top methodologies for initiating and managing continuous process improvement, and has already been enthusiastically adopted by a wide range of companies and industries. It is pretty clear that MDA would generate a lot more market awareness if we could begin to show how it can be used to enable Six Sigma. Not surprisingly, that is just what this article aims to do.

Six Sigma in a Nutshell

What is Six Sigma and how did become so popular? Invented at Motorola in the mid-1980s, Six Sigma was first developed to help identify and quantify the defects generated by manufacturing processes, and then reduce those defects to an acceptable level. Motorola claimed that using Six Sigma saved it millions, and, as word spread rapidly, this approach soon became the darling of manufacturers worldwide. In particular, Jack Welch of GE took a shine to Six Sigma, where it soon became virtually enthroned as a company deity.

Over time, Six Sigma has been extended so that it now can be applied to reducing defects from nearly any kind of process, not just manufacturing. This has worked particularly well at GE, which nowadays makes most of its money from business activities other than manufacturing, such as financial services. As a result of the efforts of GE and other big companies to expand the use of Six Sigma, it has recently become the rage in a wide variety of industries, from health care to insurance to telecommunications.

One reason Six Sigma has rapidly gained a following is that its basic principles are easy for a typical manager to understand. Essentially, Six Sigma states that by systematically examining defects, analyzing their causes, and applying appropriate corrective measures, it should be possible to improve the underlying process to the point where further efforts in this direction would be statistically meaningless. Specifically, this means reducing errors to +/-3 standard deviations from the norm, or approximately 3.4 defects per million units of activity. Since the mathematical symbol for standard deviation is the Greek letter “sigma,” the approach was dubbed “Six Sigma.”

Here is how it works in a simple manufacturing process. Suppose we crank out widgets that should be "exactly" 2 meters long. In practice, let's say this means no acceptable widget is shorter than 1.99 meters, and no longer than 2.01 meters, a tolerance of +/-0.01 meters. Using Six Sigma, we should be able to continuously improve our widget manufacturing process until we get no more than 3.4 widgets per million that fall outside that tolerance.[1]

Of course, most meaningful business processes are much more complex than our widget example, and they can't usually be reduced to such simplistic measurements of quality control. Even a relatively simple widget has more than one physical dimension with an associated tolerance, and part of producing the widget is meeting other measurable goals, like so many widgets per hour. Other, more complex business processes might be measured in terms of customer satisfaction levels, mean time to failure, and so on.

However, without getting into details here, suffice it to say that there is now any number of gurus dedicated to working out a way to break down almost any process so that Six Sigma principles can be applied. For all the relevant tolerances that can be identified and measured for each process, a continuous improvement cycle can be applied until the magic number—"six sigma"—is achieved for every type of defect measurement.

So, one of the attractions of Six Sigma is that it provides a very discrete, measurable goal for process improvement. All process improvements are welcome, but only achieving the magic number— "six sigma"—for a given identified defect means total success. Conversely, achieving "six sigma" is enough for any given defect . Then it is time to move on to the next defect. With clear goals like this, it is much easier to fight the long-term process improvement war.

Of course, whatever the goal, process improvements do not happen by magic. Therefore, Six Sigma defines a method for making improvements to existing processes, called by its acronym, *DMAIC*:

- DEFINE the project goals and customer (internal and external) deliverables
- MEASURE the process to determine current performance
- ANALYZE and determine the root cause(s) of the defects
- IMPROVE the process by eliminating defects
- CONTROL future process performance

As one can imagine, there are an endless number of theories about the best way to execute DMAIC and a related method for developing brand new processes called *DMADV* (Define, Measure, Analyze, Design, Verify). And, yes, there is a Six Sigma priesthood as well. Those who work in DMAIC and DMADV teams are called Green Belts, and those who lead them are called Black Belts. Beyond that, a Six Sigma practitioner can aspire to become a Master Black Belt and, ultimately, a Champion, presumably enhancing his career options along the way. All this adds to the mystique of Six Sigma that has helped to make it so popular.

Six Sigma and Software Development

At first blush, one area that would seem to be a shoo-in for a Six Sigma-style process improvement methodology is software development. Everyone knows that the typical corporate software development life cycle (SDLC) is a veritable minefield of problems, from inaccurate and incomplete specifications of requirements to incessant failures and delays in deployment, testing, integration, and even seemingly routine upgrades.

In fact, well before the advent of Six Sigma, process improvement schemes, such as the Carnegie-Mellon Capability Maturity Model (CMM), have stressed the urgent need for IT organizations to take a more systematic and disciplined approach to managing the SDLC. The ultimate goal of approaches like CMM is to upgrade the SDLC into an integrated suite of well-documented, reliably repeatable processes.

Unfortunately, despite some lip service to CMM and other approaches, the typical corporate SDLC remains a fractured patchwork of poorly documented activities supported by a polyglot collection of tools. Applications are developed and maintained only through the continuous "heroic efforts" of IT personnel, typically on a project-by-project basis. In organizations with decentralized IT, this problem is further compounded by the fact that each business unit's IT group may well have its own customized version of an SDLC.

Given this chaotic situation, what is truly remarkable is not that most software systems encounter serious problems throughout their lifecycle, but that any are successfully deployed at all! Not surprisingly, whenever any system reaches something resembling a stable state, everyone is then loath to make any further changes for fear of causing it to break in some totally unpredictable way.

These endemic problems naturally increase the risk and cost of every IT project and, by extension, every business initiative that depends on any new or upgraded software. Obviously, this poses a rather daunting challenge to any Six Sigma initiative where software plays a significant role in the target business process. How can you ensure continuous process improvement of the target process, when you can not control—or depend on—the SDLC associated with developing the software required to support that process?

By this logic, every major corporation that endorses Six Sigma for improving their overall business processes should be rushing headlong to systematically upgrade their software development initiatives as well. To date, however, this has not happened. Why not?

The main reason is that Six Sigma initiatives generally come out of the business arena, and are targeted to core business processes, such as widget production, insurance claims, or health care delivery. The Six Sigma folks involved do not usually understand—or even want to understand—the vagaries of software development, so they tend to avoid it. It certainly doesn't help that the typical SDLC is so confusing—and poorly documented—that even the IT department itself doesn't fully understand it.

We should make clear that Six Sigma has indeed been used help improve some processes in the IT department. For example, Six Sigma can—and has—been used successfully to streamline various non-technical IT management processes, such as those associated with procurement of hardware and software, processing of help desk complaints, hiring and retention of staff, and so on. But, after saving a few million dollars by, say, improving the procurement process for network routers, the Six Sigma folks generally declare victory and move on. The result is that the really ugly problems associated with the SDLC largely remain untouched.

Why don't IT shops themselves adopt Six Sigma and use it to improve their SDLC? As we have already pointed out, most IT shops are "Balkanized," either by project or by business unit, or both. Central IT has typically had little control over how individual projects are run, so even honest intentions to define a better SDLC process are usually doomed to failure in practice. In the end, each "Balkanized" unit does only what its business client is willing to pay for, and rarely does that include making major investments specifically for the purpose of upgrading the overall SDLC process.

Six Sigma and the SDLC

Can anything end this impasse? In order to do so, the business must decide that having a truly efficient, reliable SDLC across the enterprise is absolutely mandatory to meeting its own overall Six Sigma goals. To better understand why they should do so, let's go back to our simple widget example.

Let's call our widget manufacturing process P1, and assume it is controlled by a business unit called B1. Now suppose that P1 has an unacceptable defect level problem. Suppose the Six Sigma gurus at B1, then come up with a process enhancement that is expected to reduce P1 defects to an acceptable level. Now suppose B1 needs to create or update some software to implement this enhancement, which we'll call S1. The relevant question now is: Does B1 care what kind of SDLC is used to develop and maintain S1?

Today, the most likely answer is probably 'no. To the Six Sigma folks at B1, S1 is just a plug-and-play solution that is needed to bring P1's outputs within the "six sigma" tolerance factor. It doesn't matter whether S1 is implemented and supported by a reliable SDLC, or whether it is implemented and supported by the "heroic efforts" of a team using a completely chaotic SDLC. To the Six Sigma team at B1, if S1 appears to help reduce P1's defect rate, their immediate problem is solved.

To date, this is how most Six Sigma people who are focused on improving core business processes have tended to view SDLC issues. In this view, the SDLC is at least one level removed from the core business processes they really care about, and therefore are outside of their purvey. So long as the IT department can somehow continue to crank out S1-style software enhancements to solve the core problems they uncover, why should the Six Sigma folks care about the SDLC?

Unfortunately, this is a very narrow, short-term, and ultimately unrealistic view. It assumes that P1 is a fairly static process, and that S1 is a discrete, one-time solution that will seldom if ever need any further changes or upgrades. This is rarely the case. To understand why, we need to look at this situation in a broader context.

So, let's now view P1 more holistically—that is, as a constantly changing business process, driven by frequent updates in customer requirements within the entire B1 business unit. Any given day at B1 could bring the need to make a change to P1 that has a potential to require a change in S1. In that case, how quickly and reliably can S1 be adapted to deal with that change to P1?

Now, let's add one more layer of reality—and complexity—to this situation. Suppose the S1 is not a stand-alone, discrete software component. That is, consider the far more likely scenario that S1 depends on other software—a database, a core application, a third party application, even a particular operating system, and so on. What happens to S1 when any of these subcomponents needs to undergo a change? And, in that case, what happens to reliability of P1?

Considering the problem in this way, it now becomes obvious that, for the business process gurus at B1 to reliably keep P1 within "six sigma," they need to be sure that all of IT has a highly predictable SDLC that can ensure S1 can be readily adapted to P1's changing requirements, and that any changes in S1 or its sub-components do not undermine the functioning of P1. Now it should be quite clear why the SDLC process for S1 ultimately falls within the scope of B1's Six Sigma initiative.

This is a subtle but very important shift in thinking. Under this logic, it is no longer possible for the business to sequester IT behind a "Chinese wall," throw project requirements from over that wall, and just expect quality software to somehow come out the other end. For the business to be able to meet the ongoing demands of applying Six Sigma to its business processes, it must know that there is a transparent, repeatable SDLC process capable of reliably producing any needed software on time and on budget, and in close coordination with those core Six Sigma initiatives.

Six Sigma and MDA

Of course, just knowing that the SDLC process needs to be improved and somehow integrated with Six Sigma is not enough. You have to have some clue how to create such a reliable SDLC process. So where do you start?

As we mentioned earlier, there are any number of generic SDLC improvement paradigms that can be used as starting points, including CMM and RUP. In many cases, IT organizations have already picked one or more of these approaches, so it seems reasonable to use them in an SDLC upgrade intended to work with Six Sigma.[2]

Unfortunately, in terms of supporting Six Sigma in the way we have described, there is a big problem with most of these SDLC approaches: They are way too

generic. In an attempt to remain non-prescriptive, they don't get into the details of exactly what kind of software platforms and tools are going to be used, or what deployable components are going to be produced.

To make this clearer, let's go back to our widget example. Remember that our business unit, B1, has determined that a core process, P1, has some defects, and that reducing those defects requires upgrading some associated software, S1. Also, remember that S1 may be related to any number of other software components, which could be affected by (or could, themselves, affect) the impending S1 upgrade.

At this point, what the Six Sigma gurus at B1 really want to know is: Exactly what deployable software components are interdependent with S1, and how do we upgrade S1 without upsetting the whole applecart? Unless they can get this answer quickly and reliably, they won't care whether the IT department uses RUP, CMM, or any other SDLC paradigm.

Fortunately, there is at least one approach that can provide this kind of information—*Model Driven Architecture* (*MDA*). MDA is a framework of standards, developed by the OMG standards group, aimed at defining how all the various aspects of a software system are modeled throughout the entire SDLC of that system. To be clear, MDA itself does not currently define an SDLC process (why it doesn't is perhaps a good subject for another article!). However, what MDA does define is a way to represent the key outputs of each stage of any SDLC as a set of formal models, related to each other through a set of formal transformations, all of which can be managed in a common repository based on the OMG's *Meta Object Facility* (*MOF*) standard.

So, MDA provides a convenient basis on which to define a "Model-Driven" SDLC process, which we will call an MD-SDLC. A typical MD-SDLC starts with formal MOF-compliant models of the business requirements on the one hand, and formal MOF-compliant models of the available computing platforms on the other. Then, through a set various formal model transformations, the MD-SDLC eventually ends up "manufacturing" a deployed system that provably meets the stated business requirements and operates reliably on the available computing platforms.[3]

For our purposes here, it is even more important for an MD-SDLC to be able define a flexible and reliable way to identify and update those deployed software components to support incremental changes in either the business requirements or the available platforms. This is particularly critical to the success of continuous process improvement approaches like Six Sigma.

Fortunately, this is where MDA really shines. MDA is particularly adept at expressing various levels of abstraction as separate models, while relating all those models through formal, traceable transformations. So, for example, a Six Sigma practitioner analyzing a core business process can readily identify the MDA model components related to that process and specifically include these as elements in a DMAIC process. At the same time, the IT department can run its own DMAIC process to upgrade its infrastructure, without stepping on anyone else's toes. We will be referring to a concrete example of the approach shortly.

Transitioning to an MD-SDLC

Sounds great, but is it just pie-in-the-sky? Has anyone ever developed such a Six-Sigma-friendly MD-SDLC and, more importantly, actually installed a working version in a real IT shop? Happily, the answer is "yes.".

Over the last few years, The MOD Group[4], a professional services organization active in the OMG's MDA FastStart Program, has developed a methodology for transitioning organizations to MDA that it calls "Model Process Framework" (MPF). The MOD Group has already successfully deployed MPF to large corporate clients in different vertical markets to create MD-SDLCs that incorporate MDA and fit into existing Six Sigma initiatives.

More specifically, The MOD Group claims its clients have been able to validate a 50-65% reduction in operational costs associated with the introduction of their new MDA-based SDLCs. In one major project, using an MD-SDLC has already saved one client $4.2M, representing more than a 50% reduction in system development and maintenance costs as compared to what would have been expected using their previous SDLC process. (See Sidebar 2 for a more detailed discussion of how these metrics were derived.)

For the moment, let's focus on how The MOD Group managed to transition their customer to an MD-SDLC using MPF. The most important point is that they did so riding on the coattails of the customer's commitment to Six Sigma. Rather than try to convince the customer of the overall virtues of MDA and an MD-SDLC, they simply positioned these as necessary enablers for Six Sigma. (From an MDA point of view, of course, it was Six Sigma that was the enabler—that is, it enabled The MOD Group to successfully transition the customer to MDA.)

Let's reiterate the steps The MOD Group used in getting the customer to adopt MDA, using Six Sigma as the "enabler":

1) The customer, like countless other firms, had already adopted Six Sigma as its primary methodology for overall corporate process improvement.

2) The customer came to realize that, to reach its Six Sigma targets for core business processes, it had to ensure that its IT groups followed a reliable SDLC process.

3) At this point, The MOD Group introduced its MPF product as the best way to transition to an MD-SDLC that could support Six Sigma.

4) After the transition, the client could control its new MD-SDLC using Six Sigma's DMAIC method.

How does this work in practice? First, let's examine how The MOD Group uses its MPF transition methodology to introduce MDA to a client.

Perhaps the easiest way to visualize MPF is as an extensive decision and activity tree that is used to both analyze and upgrade the client's current SDLC and computing infrastructure to an MD-SDLC. Traversing this tree, the client is ultimately led through the process of modeling their underlying computing frameworks and business processes using MDA principles, while simultaneously upgrading their SDLC.

The MOD Group works with its clients to provide a "rolling" transition plan. First, the MOD Group performs an initial assessment to determine the client's readiness to integrate MDA into the SDLC. Once the data is analyzed, the client is able to target the rollout to provide a high return on investment. For example, the first major initiative can be cost-justified by associating it with an important upgrade to some core business process/system. Later projects can leverage what was done initially, while adding additional pieces to the puzzle. Throughout the transition, MDA practitioners insure that the principles of MDA are communicated with IT and business unit executives.

MPF must take into account that each client brings certain constraints to the table. For example, they may have to conform to various mandated corporate methodologies, such as Six Sigma and/or other more IT-specific methodologies, like Capability Maturity Model (CMM) or Rational Unified Process (RUP). Then there are, of course, budgetary and organizational constraints of all kinds.

After working with a number of clients, the MOD Group discovered that the biggest single problem is usually mapping the client's complex mix of legacy platforms and applications into a coherent Platform Specific Model (PSM). That is largely because much of the information necessary to populate the PSM is not (or is poorly) documented. However, until this mapping is done, it is not possible to trace reliably which deployed components are going to be affected by a given business process within a Six Sigma initiative.

According to Stan Sewall, Managing Partner for The MOD Group, the secret to accommodating Six Sigma is to show clearly how the artifacts of the MD-SDLC are going to map to the various steps in the DMAIC and DMADV activities. This allows the SDLC process to be "driven" by a combination of MDA and Six Sigma practitioners, thereby integrating MDA into the company's overall Six Sigma initiative. (See Sidebar 3 for an example of how this works in practice.)

Conclusion

Clearly, there is no way in this relatively short article to provide anything like a full explanation of how to integrate Six Sigma and MDA. Hopefully, we have been able to show that organizations embracing Six Sigma must inevitably deal with SDLC issues. Also, we hope we have been able to explain why MDA is a very natural basis on which to create an SDLC that can stand up to the demands of a continuous improvement methodology like Six Sigma. Finally, we hope that the example we cited provides additional evidence that the alignment of Six Sigma and MDA is doable and can produce tangible benefits.

Sidebar 1 - Six Sigma's DMAIC

Define the customer, the issues critical to that customer, and the core business process involved.

Measure the performance of the core business process by collecting data to determine defects and defect levels.

Analyze the data collected to determine the root causes of defects and identify opportunities for improvement.

Improve the targeted core business process to reduce the defects.

Control the implementation of the improvements to insure that the change is complete and meets the expected reduction in defects.

Sidebar 2 – Assessing the Financial Benefits of an MD-SDLC

Most IT managers are well aware that the most expensive element of an SDLC is the cost of maintenance throughout a given system's lifecycle. The MOD Group estimates that, for every dollar spent on analysis, five dollars is spent on development and ten dollars on maintenance (1:5:10). For the incorporation of an Enterprise Resource Planning (ERP) system, the ratio is (1:8:14). In addition, the time required for maintenance is critical, since it can seriously affect the business savings associated with a given upgrade. For these reasons, The MOD Group and its clients have chosen to focus an initial MDA project on reducing the cost and time associated with system maintenance and consolidation of architecture frameworks.

So, just looking at maintenance costs, The MOD Group client expects that:

- The system it has developed will experience about 5000 changes over its lifetime.
- The new MD-SDLC will reduce the time required per change by about 10 person-hours.
- The typical cost per person-hour of change is $60.

Using these numbers, the company initially expected to be saving over $3 million. Actually, the client observed an additional savings of 4 hours per change, adding $1.2 million to the projected savings, for a total of $4.2 million.

As the MD-SDLC matures and Enterprise Architecture methodologies are applied, The MOD Group expects to reach a new cost ratio: For every dollar spent on analysis, three dollars will be spent on development and five dollars on maintenance (1:3:5). Assuming that analysis costs are static, this should mean that development and maintenance costs should drop to 60% compared to the previous SDLC. An example of how this breaks out is shown in Figure 1 [5]:

Optimize Six Sigma with MDA

Results:

- Synchronization of app tiers
- Documentation support for CMM
- Communicate with business units
- Compress design with standards.
- Incremental changes - Business Agility
- Institute and enforce quality control
- Rejuvenate legacy technology frameworks

Activities	Current DMAIC/hrs	MDA/DMAIC/hrs
Web Tier	5	1
App. Tier	3	.5
Business Tier	9	3.5
Analysis	4	3
Dev Integration	1	.25
Deployment	1	.25
Total Person Hours	23 Hours	8.5 Hours

Figure 1 – Optimize Six Sigma with MDA (courtesy of The Mod Group):

Sidebar 3 – Integrating MDA and Six Sigma – An Example

As Six Sigma practitioners propose changes to a given business process, it should be straightforward to identify those parts of the software system that require enhancements, and to make the necessary changes without causing other parts of the system to break. Similarly, it should be possible to make changes reliably in the underlying architecture, software components, computing platforms, and software development and deployment tools without creating problems in the operation of applications that are supporting other business processes.

Figure 2 and Figure 3[6] illustrate an example of how such changes might be managed using an MDA-based architecture and SDLC. In Figure 3, a team of business analysts led by a Six Sigma Black Belt has decided to address the defects associated with the deployment of a set of software components created with an Enterprise Java Beans (EJB) framework. The problem is that the current infrastructure and SDLC force business programmers to replicate the component in a number of applications each time a change is required.

The yellow shaded area shows the Six Sigma team working with a system architect to measure exactly how many systems are involved, and how many negative effects (defects) they generate throughout the SDLC of these components. In MDA terms, these deployed components are represented as being within the *Platform Specific Model* (*PSM*).

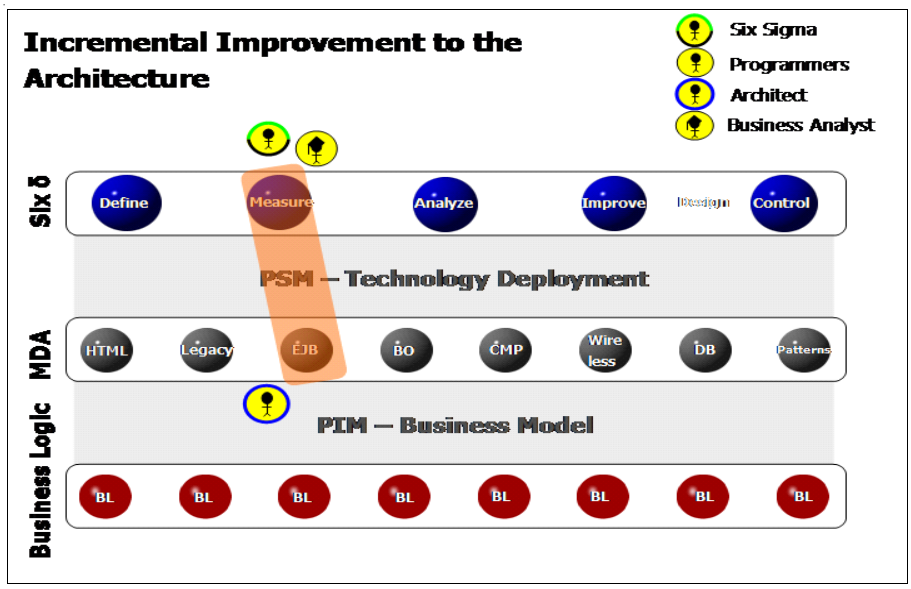

Figure 2 - Incremental Improvement in the Architecture

Figure 3 illustrates the further impact of altering the deployment of these components so that they can be shared (rather than copied) by multiple instances of business logic. In this case the initial cost of a one-time retooling to a shareable component will require some recoding and redeployment of the various business logic components by programmers. Subsequently, of course, enhancements to the shared component will not require further upgrades to the business components that share it.

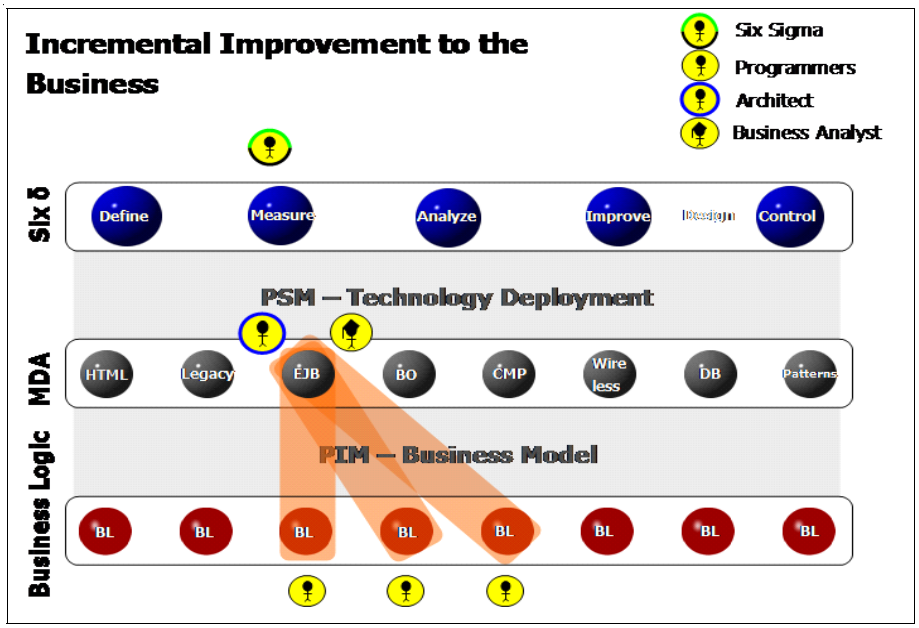

Figure 3 - Incremental Improvement to the Business

In MDA terms, the deployable components affected are in the PSM, but are clearly traceable to elements in the Platform Independent Model (PIM) which models the business logic of various business processes. Because MDA provides this traceability, the Six Sigma practitioners can readily identify the affected PIMs and their related business processes, so they can measure the likely impact of the proposed infrastructural upgrade.

Footnotes

[1] For a more complete mathematical explanation, see http://www.isixsigma.com/library/content/c010101a.asp

[2] One interesting attempt at integrating Six Sigma and RUP can be seen at: http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/nov03/m_rupsix_nm.pdf

[3] Since MDA is a relatively new initiative, many MDA tools available today fall short of implementing full provability, but still provide benefits in that they promote formal modeling and help to automate the alignment of models and code, thus bringing software development closer to being a true engineering discipline.

[4] http://www.themodgroup.com

[5] Figure courtesy of The Mod Group.

[6] Both figures courtesy of The Mod Group.