

LEAN INTEGRATION

John Schmidt

VP, Global Integration Services
Informatica, johnschmidt@informatica.com

Abstract

Lean manufacturing is a management system that emphasis creating value for end customers and eliminating activities that are not value added (waste). Lean Software Development is an agile approach that translates Lean manufacturing principles and practices to the software development domain. Lean Integration builds on these prior works by applying their principles to the process of integration.

Keywords

Lean Integration, Integration Competency Centre (ICC), System Integration, Data Integration, Enterprise Application Integration, Continuous Improvement, Deming

Introduction

Lean manufacturing is a management system that emphasis creating value for end customers and eliminating activities that are not value added (waste). Its principles were derived from the Toyota Production System (TPS) which was developed in the 1950's but in the last 15 years it is simply referred to as Lean. While Lean is rooted in product manufacturing, it is now widely regarded as a management approach that can be applied effectively to a wide range of product and service industries. Lean is closely related, and borrows from, other methodologies including Value Network, Theory of Constraints, Six Sigma and Statistical Process Control.

Lean Software Development is an agile approach that translates Lean manufacturing principles and practices to the software development domain. It was adapted from TPS and introduced by Mary Poppendieck and Tom Poppendieck in their book *Lean Software Development*¹ and expanded in *Implementing Lean Software Development*².

Lean Integration builds on these prior works by applying their principles to the process of integration. The definition of integration used in this paper is “the process of making independent information technology elements work together as a cohesive system”.

Lean Integration may be applied in various integration domains including:

- **System Integration** is the bringing together of component subsystems into one cohesive system and ensuring that they interoperate effectively.
- **Data integration** accesses data and functions from disparate systems to create a combined and consistent view of core information for use across the organization to improve business decisions and operations.
- **Enterprise Application Integration** enables information exchange and process automation across business applications that were independently developed, may use incompatible technology, are typically based on different data models, and remain independently managed.

Software systems are, by their very nature, flexible and will change over time (except for legacy systems). Integration in each of these domains therefore is not just a one-time activity and instead is an ongoing activity.

This paper addresses Lean Integration in three sections; 1) a discussion of **Lean Integration Principles**, 2) **Deming's 14 Points** and how they may be interpreted in the integration domain, and 3) a particularly relevant best practice called **Management by Fact**.

Lean Integration Principles

Lean Integration can be summarized as seven principles similar to Lean manufacturing and Lean software development:

1. Eliminate waste
2. Sustain knowledge
3. Plan for change
4. Deliver fast
5. Empower the team
6. Build quality in
7. Optimize the whole

1. Eliminate waste

The original seven wastes (*muda* in Japanese) that were targeted in manufacturing and production lines are:

- Transportation (unnecessary movement of materials)
- Inventory (excess inventory including work-in-progress)
- Motion (extra steps by people to perform the work)
- Waiting (periods of inactivity)
- Overproduction (production ahead of demand)
- Over Processing (rework and reprocessing)
- Defects (effort involved in inspecting for and fixing defects)

Developing software has many parallels with the production of physical goods and Poppendieck has done a terrific job in *Implementing Lean Software Development*² to translate these wastes into the world of software development. Many of these are fairly obvious of course such as wasted effort fixing defects rather than developing defect-free code in the first place. Overproduction in the software world is the equivalent of adding features to a software component that were not requested or are not immediately required (sometimes referred to as “gold plating”). As per Poppendieck², “*Wise software development organizations place top priority on keeping the code base simple, clean, and small.*” (p. 69)

Other parallels between the physical product world and the virtual software world are less obvious such as transportation waste being the equivalent to loss of information in hand-offs between designers, coders, testers and implementers or motion waste being the equivalent of task-switching in an interrupt-driven work environment. In any event, Poppendieck has done such an excellent job describing the wastes for software development, including systems integration to some degree, that it’s not necessary to repeat them here. Just read the book.

That said, *Implementing Lean Software Development* does not address all the integration sub-disciplines and there are several areas where we often see a tremendous amount of wasted time, effort and money.

First, building functional integration capabilities or extra features before they are needed is a waste. There is a strong desire among integration teams to anticipate organizational needs and build interfaces, data marts,

canonical messages, or SOA services with the needs of the entire enterprise in mind. This is an admirable objective indeed, but the integration teams typically don’t have the resources and funding to build the common capability and so they run into trouble when the first project that could use the capability is saddled with the full cost to build the enterprise-wide solution. This practice is referred to as “the first person on the bus pays for the bus” or similar metaphors. Business leaders whose projects budgets are impacted by this practice hate it. Let me repeat – they hate it. The business units are given a budget to optimize their function and they become frustrated, and even angry, when told that the implementation will take longer and cost more money because it must be built as a generic re-usable capability for other future users. This may indeed be the policy of the organization, but in the end it pits the integration team against the project sponsor rather than fostering alignment.

There is however another approach – build only the features/functions that the first project requires, and do it in such a way that it can be extended in the future when the second project comes along. And if the nature of change is such that the second project requires refactoring of the solution that was developed for the first project – then so be it. This approach is still by far more desirable because of the downsides of the “build it and they will come” approach:

- There is an imperfect understanding of requirements for future projects if it is built in advance so it is possible that the wrong thing will be developed
- The additional code or data will cost money to develop and maintain without any benefits until the next project comes along to use it
- The business benefits from the first project are delayed while building the supposedly “ideal” solution that will meet future needs
- The business sponsor for the first project will be dissatisfied with the implementation team (which is a good way to chase away your customer)

Organizations would be much better off adding the cost of refactoring the first project to the second project when (and if) it comes along. Under this scenario, the needs of the second project are clear as are the needs of the first project (since of course it is already in operation) so there is no ambiguity about what to build. Furthermore, the benefits of the first project will be realized sooner which can in essence help to fund the cost of the 2nd project (siloed accounting practices may still make this difficult in many organizations – but

nonetheless the advantage is clear). The bottom line message is “refactor integrations constantly so that they don’t become legacy”.

Second, applying middleware technology everywhere is a waste. This may sound like heresy to some integration specialists since after all, it is well known that the way to facilitate loose coupling between components is to add an abstraction layer. No debate on that front. But the reality is that while each layer adds a potential decoupling benefit, it also adds a cost. So this is really a cost/benefit tradeoff decision.

As integration professionals, we often deride point-to-point integrations as “evil” since they tightly couple components and, if applied without standards, over time will result in an integration hairball. True enough. But that doesn’t mean that point-to-point interfaces for specific high-volume data exchanges with stringent performance requirements isn’t the best solution. Each middleware layer bears a cost in terms of technology, people development, organizational change, and complexity on an ongoing basis, so they should only be added when the benefits of the abstraction layer outweigh the cost of sustaining it.

Another example of a middleware abstraction layer is canonical models or common data definitions. It is hard to argue against the principle of having a common definition of data across systems with incompatible data models, but nonetheless common data models are not static objects that are created at a point in time – they evolve constantly and need to be maintained. Unless you can justify the incremental staff to maintain the canonical model, don’t add this layer since it will surely become stale and irrelevant within just a few short years which is yet another example of waste.

Third, not taking advantage of economies of scale and instead re-inventing the wheel. The reality is that there are a relatively small number of integration patterns that can satisfy the integration needs of even the largest corporations. While the total number of integrations may be large (thousands or tens of thousands) the number of patterns is quite small – probably no more than a handful for 90% or more of the data exchanges. But if the integration development work is treated as a unique effort by each project team, then the result over time will be thousands of “works of art”. We know from years of experience in manufacturing lines, that cost savings of 15%-25% accrue every time volume doubles³. Our experience in software development aligns with these savings due to two factors – benefits of the learning curve (the more times you do something the better you get at it) and visibility into re-use

opportunities (the more integrations one does, the more obvious the patterns become).

Fourth, unnecessary variation in tools and standards is a waste. We have also learned from the world of manufacturing that costs increase 20%-35% every time variation doubles³. Variation in the integration arena arises from different middleware platforms, development tools, interchange protocols, data formats, interface specifications and metadata standards to name a few. In the absence of governance around these and other sources of variation, the variety of integrations could be huge.

If we combine the third and fourth examples, lack of scale and unnecessary variation, the cost and quality implications are an order of magnitude difference. To take a simple example, consider an organization with eight divisions where each develops one integration per year with their own preferred set of tools, techniques and standards. It is fairly intuitive to see that if they were to combine efforts and have one team build multiple integrations, there would be costs savings from a reduced learning curve, re-use of tools and common components. The basic idea is that a team that builds eight integrations per year rather than one will see more opportunities for re-use and will get much faster simply by doing similar work with similar tools on a repeated basis. A centralized team that is producing higher volumes sees the patterns and can take advantage of them.

Using this simple example, if the cost of building an integration for each of the eight divisions was \$10,000, the cost for a central team would be \$1,159-\$3,144 - a 70-90% reduction! This result comes from doubling volumes three times with savings of 15-25% each time and cutting variation in half three times with a 20-35% saving each time. Real world case studies have shown that this kind of dramatic improvement in integration development savings is in fact achievable.

2. Sustain knowledge

A core principle of Lean and the Toyota Production System is continuous improvement through experimentation and learning. The general notion is that there isn’t a “perfect” way to do something. Rather, that there are always opportunities for improvement that can be uncovered using scientific disciplines. As described by Poppendieck², the scientific method in general follows these steps:

1. Observe and describe a phenomenon or group of phenomena.

2. Formulate a hypothesis to explain the phenomena.
3. Use the hypothesis to predict something—the existence of other phenomena or the results of new observations.
4. Perform experiments to see if the predictions hold up.
5. If the experiments bear out the hypothesis it may be regarded as a theory or rule.
6. If the experiments do not bear out the hypothesis, it must be rejected or modified.

This breaks down to the more simplified 4-step method prescribed by Deming; Plan, Do, Check and Act. Most Lean practices focus on building knowledge but I prefer to put the emphasis on sustaining knowledge which incorporates the idea of ongoing maintenance and nourishment. There a number of ways this principle can be applied to integration activities.

First, standards should be challenged and improved. A corollary to this principle is one of the five EAI Laws⁴ “There are no universal standards”. Standards are in fact essential for a Lean integration, but it is a mistake to consider them to be fixed, static, un-changeable and applicable in all situations. Industry standards such as COBOL, TCP/IP and HTTP (to name just a few) have evolved significantly over the years. Enterprise integration standards also need to be sustained and evolve over time.

Second, Lean practices put a strong emphasis on scientific methods. One of the reasons this is so important in the integration arena is that we are often faced with the challenge of achieving alignment across multiple independent teams or organizational functions. Gaining agreement across groups that don’t usually work together is hard. In the absence of data, all you have is opinions, and gaining agreement between people with different opinions that are filtered by paradigm-colored glasses is virtually impossible without objective data. A particularly useful, and practical, technique for data driven cross-team problem solving is Management by Fact (MBF) as described later in this paper.

Third, integration dependencies between components should be maintained in a structured, and searchable, repository rather than in static unstructured formats. Your first reaction might be that this is unnecessary since your organization already has a mandate that each project develop detailed documentation about all information exchanges between components. In which case ask yourself these questions:

- 1) Does the documentation reflect what was deployed to production including changes made after the design was completed?
- 2) Would a typical business analyst, designer, or developer be able to find the documentation 2 years later? 5 years later? If the original author of the documents is no longer with the company?
- 3) If they do find the documentation, would it be understandable? In other words, are the graphical notation conventions the same for all documentation and are they based on a common glossary and taxonomy for data objects?
- 4) If maintenance changes were made to production after the project was deployed, was the documentation updated to reflect the changes?

If you answered yes to all four questions, then you already have the necessary discipline for this particular aspect of sustaining integration knowledge. If you answered no to one or more questions, then you are wasting time every time you need to change something and need to re-create yet another version of a custom integration document.

3. Plan for change

Most Lean practices refer to this principle as “defer commitment” or “decide as late as possible”. The general concept is to wait as long as possible until you have the maximum information before making a decision that would be hard (or expensive) to reverse. Agile software development makes effective use of this principle. A concept that is closely tied to this idea is mass customization which includes practices such as a) designing products that are based on standard components and are customized, assembled or configured for specific users, b) reducing setup times to enable small batch sizes (even batches of one), and c) leveraging the supply chain to achieve just-in-time inventory management. In the realm of integration, it is more relevant to think of this principle as planning for change and there are several key techniques which can be leveraged to enable it.

First, break dependencies between components to enable each component (or system) to change without impacting others. Loose coupling is one of the most highly valued architectural properties – especially at the enterprise level. Despite the qualifiers in the previous waste section about applying middleware blindly, middleware technologies are one of the most

effective ways to achieve loose coupling, and thereby help to break dependencies between applications. One of best examples of loosely coupled systems can be seen in the typical supply chain. Business-to-business (B2B) interactions are made possible by industry standard data definitions, common protocols, and tools which handle process orchestration as a separate layer from the business applications. In other words, B2B interactions often include at least three types of middleware abstractions; 1) canonical data models, 2) common transport, and 3) orchestration layer. Achieving loose coupling within an enterprise between vendor-purchased applications can be achieved with the same kind of middleware infrastructures.

Second, make it easy to rebuild integrations at the “push of a button”. The reality is that you never build an integration just once. You build it many times – and in fact it is common practice to rebuild an integration a number of times even before the first production deployment since it isn’t until system integration testing that the “real” data definitions and quality issues appear. So if changing an integration is the norm, why not automate as much as possible to the point where you can quickly regenerate a production integration at the push-of-a-button.

Third, make decisions reversible. For example, let’s say you made a design decision to use an EII (Enterprise Information Integration) approach to create composite views of customer account data in real-time, only to discover through volume testing that the response time will not meet requirements. The alternative approach is to use an ETL solution with some of the customer account data replicated in order to achieve the performance requirements. Normally this would be a significant design change that would impact the overall project time-line if made late in the implementation cycle. However, if all the systems of record and data mapping rules are maintained in a metadata repository and if the data transformation routines are re-useable middleware components, then it may be possible to quickly reverse the original design decision.

Fourth, use mass customization techniques such that a given integration becomes more of an assembly process rather than a custom development effort. This certainly is the core idea behind SOA and middleware platforms, but it still requires discipline to execute it effectively. For example, most large IT shops have a common routine for doing customer address cleansing, but it is surprising how many teams don’t use it and reinvent their own which defeats the purpose of a common routine if it is not used everywhere. A part of the solution is to establish an ICC or a central team that

is responsible for maintaining and enhancing the shared business objects and re-useable components on an ongoing basis.

4. Deliver fast

There is an old project dilemma that suggests you can’t have it fast, good and cheap – you can only pick two. That trade-off may be true for custom one-off solutions, but it’s not true for integration development if you approach it as a repeatable process. Some organizations have reduced their cost by a factor of 10 while also delivering integrations rapidly and with consistently high quality. The secret to achieving this amazing result is to focus on time.

A funny thing happens when you speed up the development of integrations – costs drop and quality improves. This may seem counter-intuitive since speed is often equated with throwing more people at a problem to get it done faster which drives up cost. Another common misperception is that speed means rushing and making mistakes which reduces quality and drives up cost due to errors and re-work. Time-based competition – or using time to differentiate your capabilities – can fight the old paradigm.

The best way to speed up development of integrations is to eliminate wasted activities, reduce delays, and re-use common components, all of which have already been discussed. Nonetheless, there are few additional techniques which can improve delivery times.

First, take integration development off the critical path for projects. Each project has a critical path or bottleneck that is the limiting factor for how quickly the end-to-end project can be completed. Integration activities often are on the critical path because of the complexities and uncertainties about how all the pieces work together. This doesn’t need to be the case if you clearly understand the root cause problems that tend to put integration activities on the critical path and tackle them head on. For example, data quality issues between disparate systems are a common cause of significant delays during integration testing. A solution for this problem is to include a mandatory pre-project data profiling assessment for all relevant projects. It is amazing how much time can be saved in development and testing when the data to be exchanged between systems is well understood at the beginning.

Second, implement a variable staffing model and develop processes to rapidly ramp up new staff. Supply and demand mismatch is another common root cause problem for delays. For example, if you have a fixed number of integration development staff that are

supporting new project development, some of them may be idle at times when the demand is low while at other times demand may peak at well above the planned staffing level. If you have 10 staff and the demand jumps to 15 for a period of several months, there are only a few options available:

1. Get all the work done with 10 people by cutting some corners and reducing quality. This is not sustainable since reduced quality will come back to haunt you as increased maintenance costs or reduced future flexibility.
2. Make the 10 people work overtime and weekends. This is not sustainable since you could end up “burning out” the staff.
3. Delay some of the projects until demand drops. This is not sustainable since you are in fact chasing away your customers (telling someone that you can help them in two months when they need help now is the same as saying “no”).
4. Bring on additional staff. This is the only sustainable model, but only if you can ramp-up the staff quickly and then ramp them down when the volume of work subsides. The keys to making this happen include having well-defined standard processes, good documentation, and a long-term relationship with consulting firms.

Third, focus on driving down end-to-end project cycle time and not on optimizing each activity. This may seem counterintuitive, but optimizing the whole requires sub-optimizing the parts. For example, the task of creating user documentation is most efficient when all the software is developed. However, this can add a significant delay to the overall project if the documentation is prepared in a sequential manner. The overall project time-line can be reduced if the documentation effort is started early (even in the requirements phase) despite the fact that some of it will need to be re-written to reflect the changes that take place in the design and development phases. But if every functional group that is support a project is motivated to reduce the total cycle-time rather than focusing on the efficiency of their team, the overall improvements can be very significant.

5. Empower the team

Empowering teams is primarily the responsibility of senior management, but front-line staff can also take power through their actions. The book *X-Teams*⁵ lays out the dilemma quite succinctly:

“Top managers have a vision, but how can they get the rest of the organization to implement the programs needed to realize it?”

This would not be a dilemma if personal and organizational goals were aligned, if teams have the support and tools they need, if all the information needed to get the work done is available and doesn’t change constantly, and if the work is not highly dependent on other activities inside and outside the organization. These conditions are quite rare, hence the advice from X-Teams that teams need to be externally focused and not internally isolated and that their power and effectiveness comes from a combination of top-management empowerment, treating people with respect, and team member initiative. In the world of integration, there are a number of specific principles which can help.

First, clearly define responsibilities of the integration team. Much of the problem with integration in organizations is that while the responsibilities of each functional area are well defined, there is a tremendous amount of ambiguity around who is accountable for the information exchanges between groups. This same problem about who is responsible for the “white space” is evident in the technology arena. More specifically, who is responsible for the integration systems.

An integration system is a collection of components that are managed as a unit for purposes of data integration. It is separate from, and provides services to, other application systems within an enterprise. The services may include data migration, data consolidation, data synchronization, data quality, or process orchestration, to name just a few. This contrasts with the traditional view, in which integration components are managed as part of a business application. In a worst case scenario, they are not managed at all (i.e., they are “orphan” integration components).

An integration system views the integration components from a holistic perspective. It defines clear boundaries around each business application and explicitly defines all the components that collectively represent the integration system and its functions, regardless of how distributed those components might be. This provides the ability to sustain data integration across applications after initial projects are completed.

Second, automate routine activities to give people jobs where they have to think. Many of the integration tasks are very repetitive which has two downsides; 1) using manual labor to perform repetitive tasks is a waste of labor, and 2) staff are lulled into complacency when

the work is too routine and they are not being challenged intellectually. It is important to note that automating manual activities is NEVER about replacing people with machines. Instead, it is about raising the level of integration that may be achieved. For example, if you take a team of 10 people that do nothing but data mapping and reduce the need for staff down to 5 people through improved automation, you can then use the other 5 people to focus on more value added capabilities such as end-to-end process flows, master data management or improved data quality all of which create much more value for the enterprise than low-level mapping work.

Third, treat the extended team, including offshore partners, as first-order team members. This relates to both formal measures such as metrics and incentives and informal techniques such as team celebrations and recognition. Most company human resource policies put limits around this since it is important for tax and legal reasons to not treat contract staff the same as employees, but that doesn't mean that contractors can't be treated as team members. For example, if you have an incentive bonus for employees, then make it a point to negotiate the terms of the contract with your partner such that staff assigned to your project are eligible for similar project bonuses from their employer. Informal techniques for team can be even more powerful. For example, ordering a celebration cake and having it delivered to the offices of your offshore partner in Mumbai when a project milestone is achieved is tremendously motivating for the remote staff.

Fourth, teach staff how to learn and solve problems, and teach managers how to teach staff. Note that we didn't say teach technical skills. Of course if someone is expected to develop Java programs they need to learn Java – but that is not where team empowerment comes from. Empowerment arises when staff are able to think “outside the box” (i.e. to not put artificial barriers around their job responsibilities) and are not afraid to tackle problems that are outside of their comfort zone. While some people have a natural tendency for this capability, there are a number of techniques and skills that can help everyone improve. In any event, the first step should be to ensure that first line managers and supervisors have the skills necessary to coach their staff.

6. Build quality in

There are three basic strategies for improving quality:

1. Inspections, review or audits to ensure quality
2. Reduce cost of recovery to quickly fix problems that are identified

3. Build quality in by “error proofing” the process

Most organizations put the emphasis in that order – but this is a mistake. You can't “inspect quality in”. By the time the defects are found the damage is done and the cost of fixing the defects can be quite significant. If it is relatively easy to reduce the cost of error recovery in comparison to error-proofing the process, then option 2 could be a viable strategy. However, the recommended approach is to not create the defects in the first place. Contrary to popular opinion, it is possible to create defect free code – even when an outsourcing partner is involved. Poppendieck provides a robust treatment of this topic in Chapter 8 of *Implementing Lean Software Development*². In addition, there are several techniques that integration teams can use.

First, mistake-proof the deployment process. Deployment can include the entire process of source code management and propagation of code from development to test to production. The best way to ensure that mistakes don't happen (i.e. moving the wrong version, not including a dependent component, not synchronizing the changes with other dependent elements, etc.) is to automate the process. The general idea is to not allow human hands to touch the test or production environments. Instead, describe the changes you want to make in a repository that is integrated with a deployment tool, and when the deployment manifest is complete and the approvals are in place, then have the tool push out all the changes. Two very powerful features emerge when you have this capability; 1) you can make test or production releases much more frequently which allows development of code in small increments (small batches) which is one of the keys to improved quality and reduced cycle-time, and 2) it becomes much easier to reverse the change and roll-back to the prior production state if in fact a problem arises.

Second, stop building legacy code. Poppendieck² defines legacy systems as follows:

“There are two kinds of software—change tolerant software and legacy software. Some software systems are relatively easy to adapt to business and technology changes, and some software systems are difficult to change. These difficult-to-change systems have been labeled “legacy” systems in the software development world. Change tolerant software is characterized by limited dependencies and a comprehensive test harness that flags unintended consequences of change. So we can define legacy systems as systems that are not protected with a suite of tests.

A corollary to this is that you are building legacy code every time you build software without associated tests.” (p. 166)

The lesson to take from this for middleware systems or components is to make them modular and testable. A corollary of this principle is that you should mandate integration requirements for new applications. That is, create explicit requirements for the ability of application systems to expose their internal functions through standardized and supported interfaces. The bottom line recommendation is to build the interfaces and test harnesses first – THEN build the rest of the integration code.

Third, perform integration continuously and not just during the life of the project. This is one of the fundamental premises behind an Integration Competency Center or Center of Expertise – that integration is a distinct discipline that requires the ongoing focus and attention of a coordinated team in order to ensure that integration solutions, once implemented, don’t disintegrate. The best reference on this topic continues to be the Integration Competency Center⁶ book.

7. Optimize the whole

As stated earlier, optimizing the whole requires sub-optimizing the parts. This applies not just to the integration domain, but the enterprise as a whole. For individual teams or functional groups to make trade-offs in the interest of the overall enterprise requires a common set of values, mission and goals. To that end, there are several key recommendations for an integration team.

First, focus on the entire value stream rather than unit or functional activities. For example, the integration team at a large bank was chartered to refactor a number of legacy middleware systems in the interests of reducing operational costs, simplifying the environment to accelerate change, improving throughout, and reducing production incidents. The team could have used these as measures of success, but instead they decided to link the project success to how satisfied bank customers were despite the fact that a) the integration systems were one or two degrees removed from directly impacting end-customers, and b) there were many other factors beyond integration systems which arguably have a much bigger impact on customer satisfaction. Nonetheless, since the integration systems played a critical role in delivering data quickly and reliably to ATM’s, the website, teller workstations and call center staff, the team considered it important to keep the end-customer in mind. The

effect of using this metric is that the rest of the organization rallied around to help the integration team achieve the results because everyone had an interest in improving customer satisfaction.

Second, deliver a complete solution. By complete we mean not just delivering the integration software components, but also understanding how any immediate changes play a role in the end-to-end data flows, how overall performance will be impacted, how the components can be changed over time, how the operations of the system can be monitored and controlled effectively, and how the capacity can be monitored and planned to ensure sustainable throughput.

Third, choose appropriate metrics to maintain a focus on the big picture and develop and engaged and collaborative extended team. The core concepts are to select a few broad-based holistic measures rather than many narrowly focused metrics. This doesn’t mean that you shouldn’t measure detailed activities in all areas and all steps of the process – to the contrary, these detailed measures are necessary for data collection in support of continuous improvements. But these detailed measures should not be used to incent or motivate individuals or teams, but rather as inputs for a data-driven improvement process. Instead, three overarching integration metrics may be all that is needed.

1. **Cycle time** of total project implementation – from concept to deployed solution.
2. **Business value** which may be financial measures such as operational cost or sales revenue, but could be any other metric which is meaningful from a business perspective.
3. **Net Promoter Score** which is a measure of customer satisfaction. This could be an internal metric for an integration team that has multiple internal customers or could refer to the external customer of the enterprise. The Net Promoter Score is a measure developed by Fred Reichheld in *The Ultimate Question*⁷ which describes a formula for subtracting the number of dissatisfied customers from delighted customers to end up with a net score on a scale of 0 to 10.

Deming’s 14 Points

The philosophy of W. Edwards Deming is a key contribution to Lean practices. Deming believed that by adopting appropriate principles of management, organizations could increase quality and reduce costs

while increasing customer loyalty. The key is to practice continual improvement and think of manufacturing as a system, not as separate steps.

Deming offered 14 key principles for effective business management and were first presented in his book *Out of the Crisis*⁸ in 1986. The original 14 principles, and an interpretation of how to apply them to the integration discipline, are listed below.

1. *Create constancy of purpose toward improvement of product and service, with the aim to become competitive and stay in business, and to provide jobs.*

Interpretation: Provide for the long-range data integration needs of the enterprise. Don't focus on just short-term project demands. The goal is to sustain integration as an ongoing discipline. Integration Competency Centers that do so will see their teams grown and provide more, not less, jobs.

2. *Adopt the new philosophy. We are in a new economic age. Western management must awaken to the challenge, must learn their responsibilities, and take on leadership for change.*

Interpretation: The IT world has changed and integration professionals need to become leaders in the organization both in terms of bringing together functional stove-pipes, and in driving organizational change.

3. *Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.*

Interpretation: Build quality into your integrations by a) shifting to a "test first" approach where test scripts are developed before the code is written, b) implementing many small incremental changes rather than large massive changes, and c) using metadata to maintain accurate documentation.

4. *End the practice of awarding business on the basis of price tag. Instead, minimize total cost. Move towards a single supplier for any one item, on a long-term relationship of loyalty and trust.*

Interpretation: Stop using hand-coding tools and instead select an integration platform for use across the organization. Reducing variation will minimize total cost and establish long-term relationships with a small number of suppliers.

5. *Improve constantly and forever the system of production and service, to improve quality and productivity, and thus constantly decrease cost.*

Interpretation: Invest in reducing time and cost of repeatable integration patterns. As soon as one pattern is effectively automated, look for the next opportunity and repeat indefinitely.

6. *Institute training on the job.*

Interpretation: Improve the quality of the integration program by developing and coaching staff across the enterprise to build their individual and collective performance. Encourage staff to ask open-ended probing questions to define problems, uncover needs and clarify objectives as part of their day-to-day activities.

7. *Institute leadership. The aim of supervision should be to help people and machines and gadgets to do a better job. Supervision of management is in need of overhaul, as well as supervision of production workers.*

Interpretation: Continually examine competitor and best in class performers to identify ways to enhance the integration service and value to the enterprise. Proactively connect others to best in class performance to drive enhancement of the integration practice.

8. *Drive out fear, so that everyone may work effectively for the company.*

Interpretation: Institute post-project reviews for all projects, capture key lessons, and institutionalize those lessons in the training of new staff. View failures as learning opportunities and not as a "search for the guilty".

9. *Break down barriers between departments. People in research, design, sales, and production must work as a team, to foresee problems of production and in use that may be encountered with the product or service.*

Interpretation: Collaborate across functional groups to determine impact of implementing new processes and procedures. Integrate efforts across business lines to support strategic priorities. Uncover hidden growth opportunities within market and industry segments to create competitive advantage.

10. *Eliminate slogans, exhortations, and targets for the work force asking for zero defects and new levels of productivity. Such exhortations only create adversarial relationships, as the bulk of the causes of low quality and low productivity belong to the system and thus lie beyond the power of the work force.*

Interpretation: Focus first on the substance of the integration task. That is, describe the services provided by the integration team/function, develop

integrated service request and fulfillment processes, measure the processes, and continuously improve the total cycle time. In other words, view the delivery of integration activities as a “system” that must be optimized.

11.a. Eliminate work standards (quotas) on the factory floor. Substitute leadership.

b. Eliminate management by objective. Eliminate management by numbers, numerical goals. Substitute workmanship.

Interpretation: Do capture and measure processes, but for the purposes of gathering data for continuous improvement and not for the purpose of rewarding or punishing staff.

12.a. Remove barriers that rob the hourly worker of his/her right to pride of workmanship. The responsibility of supervisors must be changed from sheer numbers to quality.

b. Remove barriers that rob people in management and in engineering of their right to pride of workmanship. This means abolishment of the annual or merit rating and of management by objective.

Interpretation: Strive to eliminate integration wastes and give staff the tools they need to do their job. Once again, use metrics to manage the “system” and to continuously improve it, not to reward or punish staff.

13. Institute a vigorous program of education and self-improvement.

Interpretation: Establish a repository of best practices that are taught to all new staff, including temporary consultants. Make it a requirement that all integration contribute to best practices and spend some time each year teaching others.

14. Put everyone in the company to work to accomplish the transformation. The transformation is everyone's work.

Interpretation: Establish big-picture end-to-end metrics that engage everyone. Always keep the end customer in mind. Gather necessary data to define the symptoms and root causes (who, what, why and costs) of a problem. Develop alternatives based on facts, available resources, and constraints.

Management by Fact

Management by Fact (MBF) is a tool used in several methodologies, including Six-Sigma, CMM, and QPM. It is used to promote data-driven decision-making and is a concise summary of quantified problem statement,

performance history, prioritized root causes, and corresponding countermeasures.

A key characteristic of MBF is that it uses facts to eliminate bias, which makes it a particularly powerful tool for gaining agreement across functional silos. Furthermore, MBF integrates problem solving with resources and effort in a prioritized fashion. Finally, the core concepts of MBF are quick to learn which make it easy for newly formed teams to gain rapid alignment around a technique for shared problem solving. MBF consists of 3 steps:

1. Clarify the problem using the “4 Whats” technique
2. Determine root cause using the “5 Whys” technique
3. Present the results in a concise 1-page summary

Clarify the problem

The “4 Whats” technique is used to help quantify the problem statement and the gap between actual and desired performance. The basic idea is to start with a general problem statement and to keep asking questions (at least 4 times) until a very specific problem statement is reached. Following is an example to illustrate the technique.

- Initial Problem Statement: Customer “wallet share” is too low.
- *What is too low?* Compared to other companies, which have 8 accounts per customer, our organization has an average of 4 accounts per customer.
- *What is the impact of this gap?* It represents lost revenue and earnings potential.
- *What is the correlation between accounts per customer and revenue?* A customer with 4 accounts generates an average of \$220 of revenue per year, while one with 8 accounts is \$850
- *What is the potential impact if the gap is closed?*

Final Problem Statement: *Customer account penetration for our company is half that of the best-in-class competitor. Doubling the number of accounts per customer to match the industry leader would increase revenue 386%, which corresponds to \$1.8B annually.*

Determine root cause

The goal of the “5 Whys” technique is to move past beliefs and opinions and use facts to identify underlying problems. The danger of NOT identifying the root cause is that a superficial symptom of the underlying problem may be viewed as the core problem to be solved. The general idea is that by the time you have answered the 5th “why” you should understand the root cause. The example below illustrates the technique.

- Initial Problem Statement: Customer “wallet share” is too low.
- Why? We aren’t cross-selling enough.
- Why? We can’t cross-sell products on our website.
- Why? The web application doesn’t know what products have been offered and/or rejected through other channels.
- Why? We don’t have an integrated view of customer data that all channel systems can use.

- Why? Each line of business operates independently and continues to implement incompatible systems.
- Final Solution: Create an enterprise-level business case targeted at aligning all business groups and gaining top-level executive buy-in for an enterprise initiative.

Note that if we had stopped after 4 whys, the team may have initiated a project to build a web application that interfaces with other departmental systems – only to discover after implementation that other departments are changing their systems/processes too, which would invalidate the integrated solution. In this scenario, the real root cause is lack of alignment across business units, which should be addressed before a technology solution is implemented.

Present the results

The 3rd step in an MBF is packaging the facts and analysis in a concise format. The prescribed format is outlined in Figure 1 below.

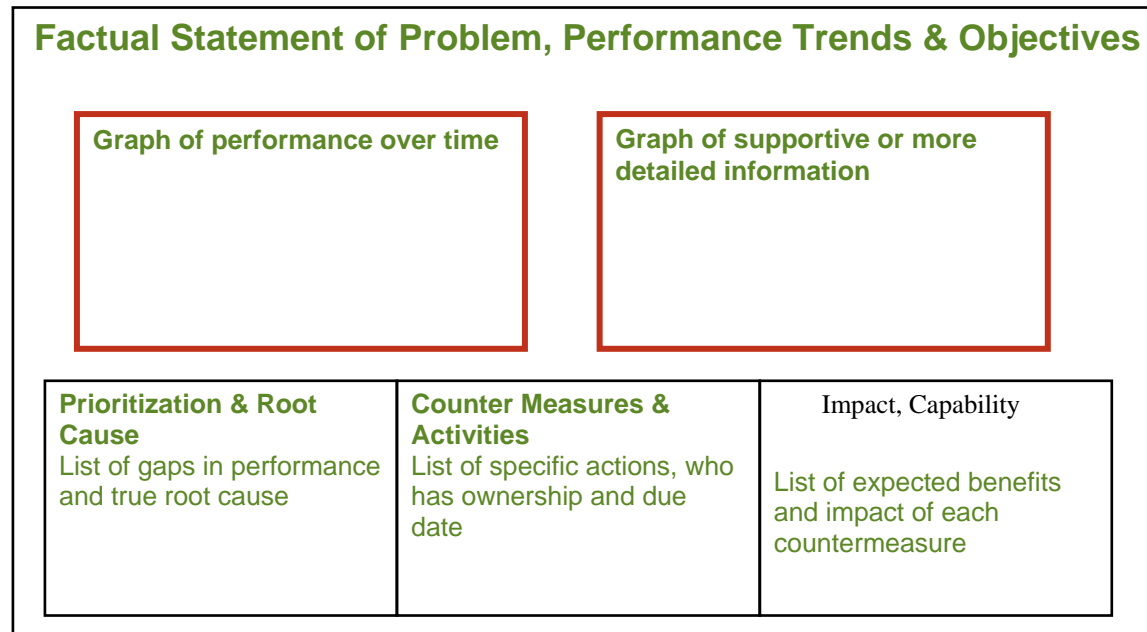
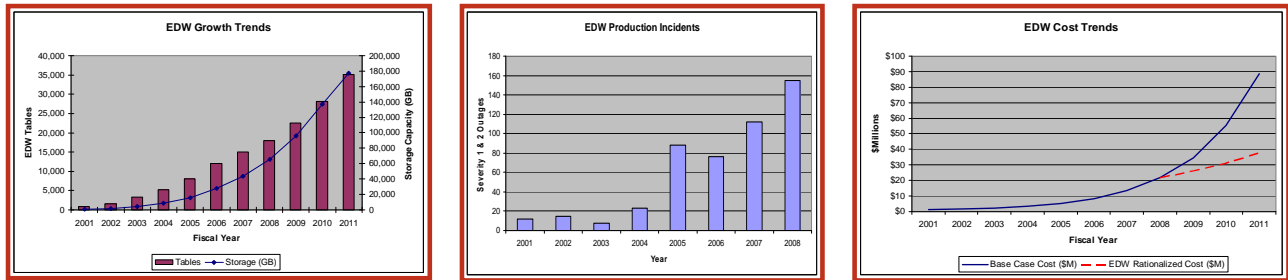


Figure 1, MBF One-Page Format

The figure below shows an example template that has been completed for data warehouse consolidation project. The initial problem statement was, *“The costs for the data warehouse are out of control and it’s not helping our business. Either we can’t find information we need, or the system is down so we can’t get at the data, or if we get the reports, they show conflicting performance metrics.”*

The final problem statement is shown at the top of Figure 2, with supporting data in graphical format, prioritized root causes, and corresponding countermeasures.

The EDW storage capacity is doubling every 18 months and production incidents have increased to an average of one Severity 1 or 2 outage every two days. The cost of the EDW is \$22M in 2008 and is increasing at 60% per year despite lower storage costs. Implementing the EDW Rationalization Program would reduce the size of the EDW by 30% and simplify the environment resulting in the annual cost increase dropping from 60% to 20% which would save BIGCO \$84M over the next three years.



Prioritization & Root Cause	Counter Measures & Activities	Impact, Capability
<ul style="list-style-type: none"> Each LOB creates their own DB tables using their own standards There is no visibility as to what data already exists in the EDW Problems occur frequently due to poor understanding of interdependencies Reports from different LOB's show conflicting data due to lack of standard enterprise definitions 	<ul style="list-style-type: none"> Implement a BI Competency Center (John Smith, plan due by 4/1/09) Acquire a Metadata Repository (Jane Doe, Issue RFP by 3/15/09) Establish EDW Configuration Mgmt Board (Jane Doe, by 4/14/09) Launch a Data Governance program (John Smith, 6/1/09) 	<ul style="list-style-type: none"> About ½ the goal to reduce growth trend About ½ the goal to reduce growth trend Reduce production incidents by >50% Resolve all priority 1 and 2 audit findings by year-end

Figure 2, MBF Example for Data Warehouse Rationalization Initiative

Author Information

Mr. Schmidt is vice president of Global Integration Services and director of the Integration Competency Center Practice at Informatica Corporation. He advises clients on the business potential of emerging technologies and leads in the creation of strategies for enterprise initiatives. He has worked in Banking, Retail, Telecommunications, Education, Government and Utilities industries. He Chairs the Integration Consortium, has written a book and many articles on Systems Integration and Enterprise Architecture, developed Program Management best practices and is a frequent speaker at industry conferences.

References

- Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit*, Addison Wesley, 2003
- Mary Poppendieck and Tom Poppendieck, *Implementing Lean Software Development: From Concept to Cash*, Addison Wesley, 2007.
- George Stalk, *Time - The Next Source of Competitive Advantage*, Harvard Business Review, 1988

- John Schmidt, *The EAI Laws*, Business Integration Journal, July 2002
- Deborah Ancona, Henrik Bresman, *X-Teams: How to build teams that lead, innovate, and succeed*, 2007, Harvard Business School Publishing Corporation
- John Schmidt and David Lyle, *Integration Competency Center: An Implementation Methodology*, Integration Consortium and Informatica, 2005
- Fred Reichheld, *The Ultimate Question: Driving Good Profits and True Growth*, Harvard Business School Press, 2006.

- W. Edwards Deming, *Out of the Crisis*, MIT Press, 1986

Copyright

Copyright © 2009 John Schmidt.

The author grants a non-exclusive licence to the Integration Consortium to publish this document in full on the World Wide Web (prime sites and mirrors) and in printed form. Any other usage is prohibited without the express permission of the author.