

Traveling the Open Road: Using Open Source Practices to Transform Our Organization

Phillip Smith
Specialist Consultant – DTE Energy
smithpl@dteenergy.com

Chris Garber-Brown
IT Manager – DTE Energy
garber-brownc@dteenergy.com

Abstract

At DTE Energy, we introduced Open Source practices to improve the performance of our 300-person software organization, achieving measurable improvements in code quality and reuse. We chose a community-based approach after a previous false start using tools and outside consultants. Two of our key principles for the transformation program were respect for the individual and creating a safe environment. We designed the program to support self-directed learning, with more skilled staff providing guidance as required. We also opened our technical environment, following an Open Source approach, to support sharing and communication. As always, cultural issues were critical to the success of the program. This paper describes the history of and lessons learned from our adoption of Open Source practices.

1. Context and History

DTE Energy, a Fortune 300 company headquartered in Detroit, Michigan, USA, is a diversified energy company involved in the development and management of energy-related businesses and services worldwide. DTE Energy's Information Technology Services (ITS) Organization provides leadership, oversight, delivery, and support on all aspects of information technology (IT) across the enterprise on behalf of its 11,000 employees.

1.1. Beginning with Agile

In 1992 the IT group formalized its solution delivery methodology with a tailored System Development Life Cycle (SDLC) based on James Martin's Information Engineering (IE) approach [1]. Consultants guided our customization of the waterfall methodology built on IE principles, processes, and procedures.

By the late 1990's, however, another movement in the software engineering industry was gaining momentum. Pioneered at nearby Chrysler Corporation, an alternative approach to delivering software emerged that would later be christened "eXtreme Programming" [2].

Lynne Ellyn, our CIO and formerly in senior leadership positions with Netscape Communications, Organic Online Inc., Xerox Corporation, and Chrysler Corporation, sought a meaningful opportunity to apply and learn from these nascent agile approaches here at DTE Energy.

Meanwhile, our company began a ground-breaking initiative to transform the enterprise from a traditional regulated utility to a major force in the deregulated market. Like much of the U.S., Michigan pursued utility deregulation and we embraced the challenge to provide open access in 1998, two years ahead of final legislation.

At this time we began to mature and formalize a "house blend" of agile methods and techniques that *worked* in our culture. Our cross-functional Electric Choice Implementation Team successfully met its organizational objectives and legislative challenges. Release after release, we enabled business results with our time-boxed, iterative, priority-driven approach to delivering and enhancing software [3].

1.2. SEMS Development Practices

As we left the nineties, we began to realize the benefits of cross-functional methods and techniques and migrated away from IT staff directly aligned with business units. This led to the creation of the Software Engineering, Methods, and Staffing (SEMS) organization, which is responsible for pooling software engineering skills and deploying those skills on demand across the enterprise.

By the end of 2003, SEMS had matured into a high-level set of pooled practices: Business Analysis, Project Management, Development, and Test. *The 'pool' concept speaks to the administrative aspect of people management and the 'practice' concept speaks*

to the birds-of-a-feather aspect of the self-subscribing organization [4].

With the large number of the development staff, the development practice was further partitioned by application tier: Presentation, Middle Layer, and Persistence. The Presentation Practice focused on reporting and desktop applications, the Middle Layer Practice focused on the business logic of N-Tier applications, and the Persistence Practice focused on data storage and client/server applications.

2. Strategic Technology Direction Change

With 2004 came a refocusing around strategy for the entire ITS organization and the **Information Technology Services Strategic Plan for 2004-2008** was commissioned. As the software engineering organization, SEMS was responsible for one goal in particular and its corresponding objectives and action plans.

Goal: Deliver and Support World-Class Business Solutions to the DTE Energy Business Community

- Improve the Quality of ITS Business Solutions (The **Better** of Better, Faster, Cheaper)

Leverage Object Orientation: ... to deliver object oriented systems, we will use Java for custom in-house development ...

- Enhance the Timeliness of ITS Business Solution Delivery

(The **Faster** of Better, Faster, Cheaper)

Establish a Reuse Program: Establish a process that will identify, mature, and promote reusable assets...

- Increase the Affordability of ITS Business Solutions (The **Cheaper** of Better, Faster, Cheaper)

Pool Solution Delivery Resources: Continue to pool the Solution Delivery staff into practices having similar skills, and deploy them as needs arise...

As SEMS was already pooling resources, the focus moved to Java and reuse.

3. Reuse and Past Struggles

Given the 2004 mandate of software reuse, we purchased dedicated hardware and installed software tools specifically aimed at embedding reuse into the development practices. The tools were configured and rolled into production, thinking “if we build it they will come.” They didn’t.

Several attempts were made to heighten interest including discussions at organizational gatherings and bringing in an industry leading author and consultant to speak, but to no avail. Eventually, the hardware was recycled to other efforts and the licenses for the tools lapsed. We needed a new approach.

Around the same time, we realized that aligning with our strategy of building new custom applications in Java increased our reliance on contractors because the technical and business knowledge to support our existing application suite lay with our employees. This translated into meeting the spirit of the goal’s objective in the short-term, but what did it mean for the long-term support and viability of the portfolio? Again, we needed a new approach.

The next realization involved how the pool was organized, specifically development. A supervisor was assigned to each practice and was responsible for assigning developers to projects. Each developer then selected their main practice (and would directly report to the corresponding supervisor), and they had the option to subscribe to any or all of the other development practices lead by different supervisors.

Eventually, this full self-subscription model, based on trying to make everyone feel accepted, clashed with the reality that we needed to 1) increase our ability to develop in Java, 2) address a constrained but rapidly growing need for employee Java developers, and 3) accept that not all subscribed employees would be able to make the transition.

Reuse was struggling, we lacked trained employees, and we had no visibility into who should be trained. We needed more than new approach; we needed a new paradigm. The shift began with focusing the Middle Layer Practice on J2EE. This meant setting entry criteria for subscribing to the Middle Layer – something that had not been done before – and training those who met it.

4. The Agile Training Program

Agile was working on our software engineering projects, so we decided to treat the creation of a Java training program as an agile project [5] – one targeted at creating and maintaining Java skills in our employees. Using the project metaphor helped to ask standard questions:

- What is our purpose?
- Who is our sponsor?
- Who are our customers/audience?
- What are our targets?

We quickly received answers to those questions:

Our purpose was to build organizational employee development capability in the Java and J2EE technology platform, as we wanted to reduce our reliance on contractors and position our employees to support the long-term needs of the business.

Our sponsor was Lynne Ellyn, CIO and Senior Vice President, as securing senior sponsorship is vital.

Our audience initially was those employees who could be immediately assigned to do Java development, as being able to apply what you learned was a key success factor.

Our targets were to maintain three Java Developers for every Tester, as we found that our projects were most successful with that ratio.

With the answers to our questions in hand, we brought our iterative and incremental approaches to bear and created a principle-based training program.

4.1. Respect the Individual

The first principle was ‘respect the individual’ [6]. This demonstrated that the organization was committed to individuals and their success, and the more tailored the program was to the individual, the more we would demonstrate that commitment. With that respect for the individual we divided the entire development staff into three categories based on capability: a) Can perform with help; b) Can perform alone; and c) Can help others perform.

The next step was to research which training areas and topics the individuals in each grouping or ‘track’ required. Although some data was readily available through project and activity retrospectives, which highlighted shortcomings and process breakdowns, it was more complicated to uncover the remainder.

We took a direct approach, utilizing industry and organizational standards, guidelines, and best practices to create an initial outline of knowledge expectations; then cross-referenced it with job responsibilities and track categories. *This research eventually yielded a knowledge and responsibility matrix for everyone in the community.*

After the research was complete, it was important to remember the individual – some may have already demonstrated proficiency in a topic. This person was given the option to participate (or not) in those courses. Capturing exceptions demonstrated the importance management placed on individual contributions. *Management also recognized proficient individuals*

who participated and assisted others in learning the material [7].

Defining the topics and tracks answered ‘what’, categorizing individuals and capturing exceptions answered ‘who’, so a next step was to answer ‘how’. This is when we looked to the Open Source movement and appreciated the strength of those communities, which led on to our second principle of ‘community-orientation’.

4.2. Training Program vs. Builder

When we began, we approached this as a typical training program; however, the second principle caused another shift and mindset change. If we wanted to focus on community, then we must lay the foundation for it. Because we were not only building skills but also building community, we changed our language to use the word ‘builder’ – taken from an ITS-wide effort titled KnowledgeBuilder. With our skill and community focus being Java, we called our program JavaBuilder.

4.3. Design for the Community

Community-Orientation led to several design decisions. First, instructor-led classes were avoided, especially when the instructor was not part of the community. Instead, we used sessions to enable conversation and collaboration within the community. Senior developers in the community, those who were the first employees on Java projects, facilitated these sessions and posed thought-provoking questions.

Second, to supplement these sessions, reading or coursework was assigned. Depending of the topic, the due date of those assignments was before the session, to enable the session’s conversation, or after a session to allow participants a question and answer opportunity prior to submission. If computer-based training (CBT) was available, it was used to supplement more collaborative learning formats.

Third, we created mailing lists, which were monitored (not moderated) by the senior members. This allowed participants to collaborate remotely on subtle or difficult topics. These mailing lists existed at varying program perspectives, creating different targeted slices of the community.

Finally, we held labs with a senior member on hand to answer questions. We found that applying the appropriate learning formats to each competency in the knowledge-responsibility matrix produced a curriculum with variety that kept the program fresh and the participants engaged.

A key issue for coursework submissions was the review and feedback process. Having the ‘Can help others perform’ category review the submissions worked well. Each submission was reviewed at least twice, once by the member who will be providing the feedback, and by another community member from the same category.

The next major hurdle was the ‘when’ – scheduling. We took much care to ensure that time commitments and expectations were realistic; additionally we wanted to underscore to every participant that this was important for DTE Energy and the employee. This point was driven home by our CIO at the kick-off.

We also remained flexible and responded to feedback and morale. We gathered feedback through surveys at the end of each session, regular facilitator meetings where overall progress was discussed, and a dedicated feedback email account. We measured morale by monitoring time-entry (was anyone in JavaBuilder consistently working overtime – a foretelling sign of burnout), and gauging session engagement (was the participation distributed throughout the attendees). One piece of immediate feedback was to decrease complexity, and we responded by spreading assignments across multiple weeks, sequencing related material, and keeping meeting times and due dates consistent.

As the program progressed, management again demonstrated their commitment through targeted accountability. This crucial piece required that supervisors fully understood and held the individuals accountable for all submissions related to their track. Management consistently conveyed that participation was rewarded. *Once the program was established peers held each other accountable.*

4.4. Creating a Safe Environment

After the facilitated sessions concluded, the CBTs completed, the readings discussed, and assignments reviewed, we brought the participants together to apply what they have learned through an event called **Capstone** [8].

The purpose of Capstone was to create an environment that was safe but realistic. We split the large group into smaller groups of three (one from each track), assigned a Project Manager and an Architect, and gave each team requirements to build a small application. This allowed them to produce a small, clearly defined, achievable, measurable, real-world application.

Using agile methods, we time-boxed development to a single one-week iteration and they utilized many technical skills learned earlier and other process skills like pair programming. This controlled subset of reality created another feedback loop, and possible holes in the training and misunderstandings of the material were identified and corrected first hand.

4.5. Celebrating the Community

The final principle and arguably the most important, was “Celebration.” The employees had attended four months of JavaBuilder, which included personal time to read books and complete assignments, as well as participate in sessions and labs during working hours – all while ‘keeping a day job.’ Therefore it was important that DTE Energy recognize and celebrate that level of commitment.

We pulled out all the stops. We reserved the campus auditorium for two hours during the work day so everyone could attend, catered food and drink, demonstrated Capstone applications, and told stories of lessons learned. We closed with our sponsor, who kicked-off the event four months earlier, congratulating the participants’ successful completion.

One of the Capstone applications tallied electronic votes. Later, when the community wanted to prioritize topics for community meetings, as well as set the start date for the next Capstone, they used their application.

5. Reuse Revisited and Current Struggles

Under the guidance of a builder program, the infrastructure for an open community was built and exercised. The facilitated sessions and labs allowed individuals performing similar functions on different projects and at different sites to interact. The mailing lists allowed those working on the same assignment or trying to answer similar questions to help each other remotely. Those bonds and the resulting community continued well after the celebration ended.

There were other benefits too; this new community now had a shared language on which to debate a vast variety of topics ranging from the existing coding standards and guidelines to the introduction of new technology into the standard operating environment.

One of the many side-effects was a significant increase in quality and consistency of application development practices. The metrics averaged over the five projects *before* JavaBuilder and the same metrics averaged over the five projects *after* JavaBuiler tell an undeniable story.

Metric	Before	After
No. of Classes > 500 Lines	24.6	12.6
Highest Cyclomatic [9]	83.6	24.2
No. of Methods > 10 Cyclomatic	66.8	18.4

Table 1. Code metrics

Even more surprising was the increase in quality and consistency extended to the satellite offices. The next application developed outside of headquarters, which was not included in the above averages, had just two classes with more than 500 lines of code, a highest cyclomatic complexity (number of distinct code paths through a single method) of 21, and only five methods with a cyclomatic complexity over 10.

The community matured throughout 2004. With wiki technology, a TWiki implementation, the developers had a persistent place where they could share ideas and approaches. We also set up a javadoc server, where documentation for all applications (and common external libraries) was centralized to an easily accessible location.

In 2005 we attempted reuse again. However, this time we approached it through community, and it worked [10]. First we opened up the source code repository. Previously developers had to get special permission to glance at a project's source code. That was changed so all developers could see all code across the enterprise.

Then the community created a project start-up component called "Project in a Baggie" (PIB). This component – in a zip file, naturally – contained a common flexible build script, created by a community developer, that institutionalized a standard directory structure. A different developer integrated standard technical components ranging from logging to JNDI naming, as well as example EJB configurations. Later a third developer added comprehensive documentation. Seen as a project accelerator, the community rallied around PIB (currently over 20 reuses) and it has served as the anchor for distributing other components into the community.

5.1. Sustaining the Momentum

With the success of JavaBuilder and reuse, it was important that we did not atrophy. This is when the strength of the community delivered. For the next JavaBuilder, those who had completed courses previously now facilitated sessions – the next level of applying an 'each one teach one' approach. Moreover, the structure of the builder changed as the community matured – migrating from a prescriptive high school style curriculum to a college cafeteria style catalog.

Even more encouraging was that reuse which started as a dedicated effort, changed from being evangelized by a few to integrated and supported by the community. Currently the reuse effort is approaching \$600,000 USD in captured savings through a library of more than 60 components.

In addition to reuse, the community continued to show strength by self-organizing into Pods [11]. Each of these cross-project teams consists of a Lead Developer and two to four other developers of varying skill levels – essentially creating a family unit not bound by project lifecycle. This allows Lead Developers to continue meeting their goal of mentoring project teammates as well as focusing their community mentoring efforts on Pod members. Moreover, it gives others in the Pod a consistent adviser when they are solo on projects. *The Pods worked so well that we keep them together for Capstones.*

We are also extending the builder concepts and principles to areas outside of development. In 2007, we will host a BABuilder for Business Analysts. We also plan on coordinating the Capstones and connecting the requirements definition and maintenance processes of business analysis with the application design and implementation processes of development.

All of this underpins a collaborative culture that feels ownership in how the organization operates and their place in it. Not only are we shifting our culture but we are also meeting our strategic goals of **better** solutions measured by our code quality metrics, **faster** solutions measured by our reuse metrics, and **cheaper** solutions measured through our trained employees.

5.2. Lessons Learned

In JavaBuilder I (2004), we initially named the tracks Junior, Solid, and Lead. The staff did not respond well those names, especially those in the Junior track. For JavaBuilder II (2005) we tried 'New to Java', Solid, and Lead. For JavaBuilder III (2006) we settled on Programmer, Developer, and Lead Developer.

Lesson: **Names are important.**

In JavaBuilder I, we had the concept of office hours, where throughout the week senior developers and architects were available to answer questions and help with assignments. Only a handful of participants took advantage of office hours, for the majority preferred to use the mailing lists because the lists allowed them to multitask between receiving help and performing project duties.

Lesson: **Concentrate on electronic communication.**

JavaBuilder I had a time constraint and we could not allow everyone to demonstrate their Capstone

application at the celebration. Therefore we held demonstration meetings to select which applications would be shown when the CIO was present. This created competition between the teams. The competition mentality continued through JavaBuilder II and was in direct conflict with JavaBuilder III's SOA (service-oriented architecture) theme. This disconnect was extremely visible in JavaBuilder III's Capstone which required cross-team collaboration [12].

Lesson: **Competition hinders collaboration.**

JavaBuilder I had one mailing list per track. This effectively eliminated the ability of the higher level tracks to help the lower level ones. Starting with JavaBuilder II, we created one mailing list; although it received more traffic, it allowed more to participate. Moreover, having one list reduced maintenance and giving it a community-oriented name (j2eetalk) allowed it to be used all year round.

Lesson: **Once you have focus, be inclusive.**

6. Final Thoughts

The benefit of Open Source type communities is the same benefit that any network provides. They become greater than the sum of their parts. As each individual in a community begins to leverage the power, knowledge, and resources of the collective; capacity is increased. This leverage manifests itself in many ways.

One powerful way is in finding the solution to a problem. If one individual has already solved a problem, another individual encounters the same problem, and there is no connection between them then capacity is reduced, as the second individual must solve the problem again. If, however, there is a community and a connection between them, capacity is increased as the existing solution is applied to the problem and the knowledge of that solution is distributed.

There is also the situation where no one in the community has solved the problem. In this capability case, having multiple members of the community working to solve the problem increases the likelihood of reaching a solution and the speed at which a solution will be reached. Moreover, once the solution is found, it can be consistently applied throughout the community – meaning as members move inside the community, and encounter similar usage patterns and idioms, training requirements and ramp-up time decrease. This directly translates into increased capacity and capability.

Creating a builder program that is community-oriented encourages and enables individuals to create

communities. Being cyclic, once the organization realizes the benefit of the open-community, it will continue to enable it and continuously improve it – either through enhancing the program or by allowing the community-focused mechanisms of that program to live independently.

7. References

- [1] James Martin, *Information Engineering: Introduction*, Prentice Hall, ISBN 013464462X, February 1991
- [2] Kent Beck with Cynthia Andres, *Extreme Programming Explained: Embrace Change (Second Edition)*, Addison-Wesley, ISBN 0321278658, November 2004
- [3] Steven W. Baker, *Formalizing Agility: An Agile Organization's Journey toward CMMI Accreditation*, 2005 Agile Conference, July 2005.
- [4] Jeannette Cabanis-Brewin, *Project Management Best Practices Report, Communities of Practice in the Projectized Organization*, The Center for Business Practices, February 2001.
- [5] Christopher M. Avery and Steven W. Baker, *In Their Own Words: Agile IT Leadership and Teamwork at DTE Energy*, Cutter Consortium, Vol. 18, No. 2, pp. 39, February 2005.
- [6] Rodd Wagner and James K. Harter, Ph.D., *12: The Elements of Great Managing*, ISBN 159562998X, Gallup Press, pp. 63, 2006.
- [7] Kay Pentecost, *Corralling the Cowboy Coder*, Cutter Consortium, pp. 11, February 2005.
- [8] Jerry Golick, and Brad Waller, *(Capstone) Case Study in Infrastructure Integration*, Learning Tree International, Course D25/CN(1)/B.1/207/A.1, 2002.
- [9] Wikipedia, *Cyclomatic complexity*, Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Cyclomatic_complexity, August 2006.
- [10] Gary H. Anthes, *Software Reuse: Making it Work*, ComputerWorld, Vol. 39, No. 22, May 2005.
- [11] Peter George, Jim Highsmith, and Josh Kerievsky, *Implementing Agile Methods in Large Organizations*, 2006 Agile Conference, July 2006.
- [12] Rob Lebow and Randy Spitzer, *Accountability: Freedom and Responsibility Without Control*, Berrett-Koehler Publishers, Inc., ISBN 157675183X, pp. 227, 2002.