

Seven Dimensions of Agile Maturity in the Global Enterprise: A Case Study

Robert Benefield
British Telecom plc
robert.benefield@bt.com

Abstract

Agile rollouts often struggle to succeed in large complex organizations. This is often due to a misunderstanding of the complexity of interdependencies of vast disparate teams that often exist, resulting in limited local optimizations. Understanding and mapping the maturity of practices for interdependent teams and units provides a method to discover and remove bottlenecks between groups that enable the organization to continuously improve. Maturity mapped to a superset of XP-style [1] technical and Agile program management [3,4,5] practices appears to provide a powerful model for improving efficiency and alignment of cross-organizational engineering teams. This is a case study of the model developed by BT Design, the IT division of the telecommunications provider BT, which has been implemented across development streams comprising of hundreds of teams and components to improve organizational agility.

1. Introduction

Despite possessing comparatively vast resources, large global enterprises constantly struggle to effectively utilize the breadth of capabilities available to them to produce elegant yet innovative products and services of high quality quickly and at low cost. The prodigious rate of change and complexity in technology has accelerated the speed of the product lifecycle, further stretching the corporate fabric and risking fragmentation at organizational and technological boundaries. Communication falters at these fault lines, components become disjointed and solutions degrade or fail. Under these conditions, the great size and complexity of a global giant can place it at a disadvantage to more nimble upstarts.

BT, a 150 year old telecommunications company and former monopoly with over 160,000 employees operating in over 170 countries, is a classic example. The telecommunications industry has experienced a revolution over the last 30 years, and the rate of change is accelerating. Not only have new methods of

communication appeared, but older methods often look little like their predecessors. Dial tone no longer requires a switched dedicated analog line with a tethered telephone handset on either end. A voice conversation might utilize a software program such as Skype, a wireless mobile device, a smart phone using its voice capability or Voice over IP (VOIP) software, Instant Messaging software on a computer, or a wide variety of more traditional handsets. It may be transmitted between handsets in a variety of network protocols that might transverse a traditional telephone pair, the air, shared networks crossing fiber, coaxial cable, electrical, cellular, satellite and possibly terrestrial relay stations. The network being traversed may simultaneously be carrying broadcasts bound for televisions, traffic for computers and systems, and even perhaps electrical power. New technologies have converged many traditionally standalone products and services allowing formerly unrelated industries to compete headlong offering “triple and quadruple-play” telephone, television and mobile services. The market is also simultaneously being fragmented and pulled in different directions such as gaming, content distribution, and business and retail service delivery by entirely new industries that place increasing demands upon and further compete with incumbents.

In an increasingly converged world the competitive landscape and products offered has drastically changed, and the stacks providing them have become increasingly complex. A service such as BT’s on demand television service Vision requires hundreds of disparate systems from the network to the device on the customer premises to the systems managing content libraries, streaming, provisioning and billing to all work seamlessly together for an acceptable customer experience. Each of these systems is built by a different team, often for a variety of different uses. For a product such as Vision, a failure in any of these elements is a failure of the entire service as far as a customer is concerned.

Such complexities, compounded by a rapidly changing market as well as regulatory requirements inserting additional barriers for between BT business units to eliminate the possibility of anticompetitive behavior, have created enormous challenges to offering

seamless services that are both compelling as well as cost effective. Development cycles of 36-60 months that were once the norm became unacceptable as the fickleness of the market demanded ever more rapid responses to the latest innovations. The business also had little appetite to wait such long periods for a return on an investment.

To dramatically improve its market responsiveness, BT embarked upon an Agile transformation, with a heavy bias towards adaptive and iterative project management. While cycle time improved within teams, technical integration between components increasingly became a problem, slowing down releases. An investigation into the causes was initiated which included studies of engineering methods in use by component teams across release streams. It was hypothesized that variances in engineering best practices, coupled with insufficient coordination at key technical touch points, might be a major contributor to the issue. Subsequent experiments were run on a handful of component teams across two separate release streams to determine a potential working model. After two 6-week release cycles such significant improvements were made within some of the trial teams that demand for a wider effort grew virally. These included an over 60% reduction in defects found in end-to-end test, a nearly total elimination of reported in-life faults, and a noticeable improvement in team velocity and cross-team collaboration. A company-wide initiative was kicked off to build a framework that could accelerate the rollout while simultaneously uncovering high risk areas that could be targeted for scarce coaching resources.

2. Seven Dimensions

In early investigations, it had been found that there were significant gaps and inconsistencies endemic to many of the teams across areas that later became the dimensions. Engineering practices such as repeatable builds, configuration management and code quality standards were often incoherent within and between teams, introducing unpredictability that slowed integration unnecessarily. Developer testing, let alone Test Driven Development [4], were often foreign concepts or improperly applied. With these teams test was viewed as a QA activity to happen through functional testing later in the cycle. This model was slow, not comprehensive, and often done well after development when efforts should have been focused on operational resilience rather than basic functionality and integration testing. This created large quality issues too late in the cycle to be adequately addressed,

and caused a perpetual late wave of work that would overwhelm teams.

Seven practices, or dimensions, had been uncovered during the team experiments that when used together led to significant quality and velocity improvements across the solution stack. Individually, each dimension provides useful benefits that help with delivery. However, the practices each compliment the others, providing a combined multiplicative benefit. Each of these practices were also chosen as they all had an extensive list of externally as well as internally available reference material and tools that was highly leveraged to facilitate understanding and adoption. The dimensions were defined as follows:

- Automated Regression Testing – identifying a regression in functionality soon after committing code reduces the cycle time required to find and fix the offending code. Automated regression testing that runs alongside a build allows the process to accelerate and be less intrusive
- Code Quality Metrics – establishing code standards and building a culture of design and code reviews allows for better management of code maintainability and complexity
- Automated Deployment and Backout – it is important to ensure uniformity of packaging and deployment to guarantee consistent behavior between development, test and production environments. Automation further removes the ability for inconsistency, and also allows for deployments to be done automatically as part of a larger build and test automation suite.
- Automated Builds and Configuration Management best practices – build consistency is important, as is making builds so straightforward to run that they can and are run all the time. Having an effective configuration management strategy, including a standard and known source code version control system and a sensible repository layout, both aids automated builds as well as allows for effective code management.
- Interlocked Delivery and Interface Integration testing – required to provide cross component delivery transparency of the progression of development work within each component team, and identify important touch points between components. This allows for component teams to better align their work so that they do not hold up integration efforts due to cross component misalignments
- Test Driven Development (TDD) – the

practice of having developers write and execute unit tests to ensure they fail before they write the code to ensure that when code is written, it is only that which is necessary to pass the tests. This ensures that the code is tested effectively to ensure it behaves as the customer would expect it to behave, rather than at best testing the code as it was written than how it was intended to be used.

- Performance and Scalability Testing- ensuring there is production quality built in, as well as building and improving the capacity and scaling model for the component. This both ensures that development teams are thinking through the performance characteristics of what is being built, as well as provides a potential early indicator if the solution will be cost effective.

3. Levels of Maturity

While not only applicable to technical organizations, understanding and mapping the maturity of engineering practices for interdependent components provides a method to discover and remove bottlenecks that allow for continuous improvement. A maturity model format was adopted at BT in order to provide a mechanism that could be mutually understood by both the technical and business sides of the company. Each component with any development spend above a certain threshold is required to be baselined across the maturity measures of the seven different dimensions, then work within their team and delivery stream to build and execute a plan to steadily improve their maturity across each dimension.

The model contains 5 levels of maturity, with listed measures and evidence required to verify maturity across each dimension. The maturity levels were influenced by the SEI Capability Maturity Model Integration (CMMI), with each level providing key building blocks that together would help the organization move from incomplete localized knowledge and reactive behavior to a flexible and dynamic organizational knowledge web that enables all parts of the business to quickly and proactively respond to market conditions while continually improving. It was expected that each component team might have a different level of maturity for each dimension, and that some teams might practice some but not all the practices within some of the more advanced maturity levels. However, to claim a particular maturity level in a dimension a team needed to show evidence to all relevant aspects within that level. Overall component maturity was defined as the

“lowest common denominator” or the dimension with the lowest maturity. Assistance to improve maturity in any dimension was coordinated through a central team of key stakeholders and domain experts to ensure consistency as well as effective targeting of resources.

3.1. Level 1: Emergent Engineering Best Practices

The goal of Level 1 maturity was to break the cycle of inconsistency between teams by establishing a common clean baseline of engineering best practices that could then be used as a solid foundation to build upon. Concepts such as unit testing, code reviews, repeatable builds, configuration management, code quality, and test driven development must be practiced and the value understood by the team, even if only practiced in rudimentary ways.

As Level 1 compliance was a remedial step, it was typically the first engagement made with teams, and often involved the most handholding for the team by domain experts. Improvement mechanisms at this level were industrialized to engrain each practice within the team and allow them to progress quickly to the next maturity level. For code quality metrics code standards were agreed and published, a manual code review would then be led by domain experts, followed by the establishment of a mechanism to record code reviews for impacted code for each delivered user story. Automated regression testing started by establishing rudimentary regression packs run at least once prior to delivery into delivery into End-to-End test. TDD and unit testing training were put in place. Builds and configuration management were shored up, and assessments and plans were put in place for the remaining dimensions. Investments to standardize tools were made wherever applicable. Teams were encouraged to build special interest groups and share knowledge of more mature and knowledgeable teams. This proved invaluable both for accelerating maturity but also to establish the first integral bonds between teams.

3.2. Level 2: Continuous Practices at Component Level

While Level 1 was bringing order to chaos, the goal of Level 2 was to embed a repeatable rhythm within the component team. Reusable automation, such as build, deploy and test scripts, was typically introduced. The same deployment and test automation created for development and test also began to be used more consistently and effectively in production launches. Interface testing was industrialized, and performance

scalability testing moved from capacity plans with a basic estimation model to the implementation of test harnesses to begin to prove the model. TDD moved into the implementation phase with the introduction of unit test harnesses integrated into the build. As teams at this level matured, data began to be collected and flow between the source tools and dashboards and story/requirements management tools. Earlier improvements solidified and grew, and the results became much more tangible and real to senior management and the business. As the special interest groups matured, domain expert attention began to concentrate more on the integrated release streams to prepare for the next level.

3.3. Level 3: Cross Component Continuous Integration

The first two levels were very much about the component teams themselves. However, as stated earlier the challenges at a large global enterprise tend to cluster between units, especially those that need to closely coordinate. While continuing to improve the level of maturity and automation within a component, Level 3 is where focus begins to test and strengthen the robustness between components within a flow through regular synchronized build/test cycles across interface boundaries. Builds, installation and regression tests must be heavily automated in order to meet increasingly demanding cycle times. Teams are required to better synchronize work, as well as integrate and test multiple drops during a development sprint. As this requires mechanistic alignment between teams, a less mature component can prevent a more mature team from advancing. It was found that this typically highlighted the main source of problems that constrained the smooth release of the affected product or service. If peer pressure and assistance from the more mature team did not fix the problem, the resultant visibility would allow for action to be taken at the program, architecture, or business unit level. BT discovered a number of these bottlenecks, including some in partner organizations that had previously gone unnoticed. The visibility and supporting data helped constructively resolve a number of these situations.

Within Level 3, the release stream begins to encounter opportunities to further reduce the length of the release cycles, as the time spent for set up and fixing integration issues that hold up testing steadily shrinks while improving the quality of the test coverage of the product itself. This also has the side benefit of allowing for potentially more smaller releases containing available feature functionality to be potentially released and used sooner, further improving

the ability of the business to respond to the market and for the technical and operational teams to learn more about the behavior of the product and customer use of it to provide improved focus on what the market may or may not care about.

3.4. Level 4: Cross Journey Continuous Integration

Whether it be in the example of Vision given earlier, or even a more basic example of broadband delivery, multiple products and services may be required to work in unison across a customer journey to provide the requisite service. At Level 4, it is expected that engineering practices of component teams are going to increasingly mirror those exhibited within XP teams. Beyond mature automation, refactoring becomes routine. More sophisticated injection test harnesses and instrumentation begin to proliferate, and performance and scalability extends from component to product and journey. Integration points between product lines, parallel release streams, and partner products become more industrialized, with closer and more organized coordination between product line teams. While in Level 3 a product line or release stream could be hindered by the immaturity of a particular component team, within Level 4 the immaturity of a product team can prevent a more mature one from advancing. This very quickly allows for constraints to be highlighted that can then be addressed. The ever shrinking End-to-End test cycles become much more heavily focused on operational and UAT testing, further improving the quality of the operational and user experience aspects of a given product. The quality of the instrumentation across the entire journey can also be further tapped by the technical operations teams themselves to provide complex service management views that can potentially proactively discover problems before a customer notices. This can be an incredibly powerful tool, to reduce operational costs while improving service levels.

3.5. Level 5: On Demand Just in Time Releases

At Level 5, teams have become highly productive, and are able to quickly understand and communicate dependencies and coupling between components and products. Code refactoring has become so routine as teams consistently and demonstrably leave code touched in a better state than they found it to improve quality, supportability and reuse. A SOA-based model matures within the organization, with effective risk

assessment built into the planning and daily build cycle to allow for decoupled releases delivered on demand. New products and offerings can be quickly assembled and trialed on the fly. The ability to understand the implications of removing or replacing components or products can be much more easily ascertained, allowing for the much more cost effective and timely response to technical or business challenges that are faced with already deployed modules and services. Configuration management of components, products and services becomes much better understood as mappings between automatedly deployed packages, the characteristics of performance across the product stack, and the touch points and the tests for them across the entire journey become industrialized to a point where there is the opportunity to model and tune a service on the fly. The lines between end-to-end test, development, operations and the business blur to the point where improvements and ideas can flow much more freely across the company, and opportunities to proactively experiment can take place with small multidisciplinary teams in a matter of days if not hours. While this level can be incredibly difficult to achieve at a large scale, the market differentiating characteristics of it are so compelling that even reaching it on a small scale make it a very worthwhile goal.

4. Assessments

Initially, parallel assessment activities were run, with one being done as a self assessment by the component team, and a second done by an unbiased expert third party. This was done to determine misunderstanding of the definitions by the component team, as well as to find opportunities for component teams to produce evidence of their maturity that might not have initially been in a consumable form. A Spider graph of maturity across each of the seven dimensions was produced for each team, as were targets to improve for the first quarter. In following quarters, component evidence was collected and reviewed by a central team of domain experts.

In conjunction with the component team lead and the business unit delivery stream leads targets for each dimension for each quarter would be adjusted. Initial knowledge sessions, reading material and a road show were set up to help bolster team understanding of the dimensions and maturity levels. Training programs, intense deep dive sessions and hothouses were set up with teams when deeper problems were encountered. More mature teams were identified and presented to other teams to further bolster the training programs and special interest groups. Partners were also introduced

to the program, and where applicable were asked to participate.

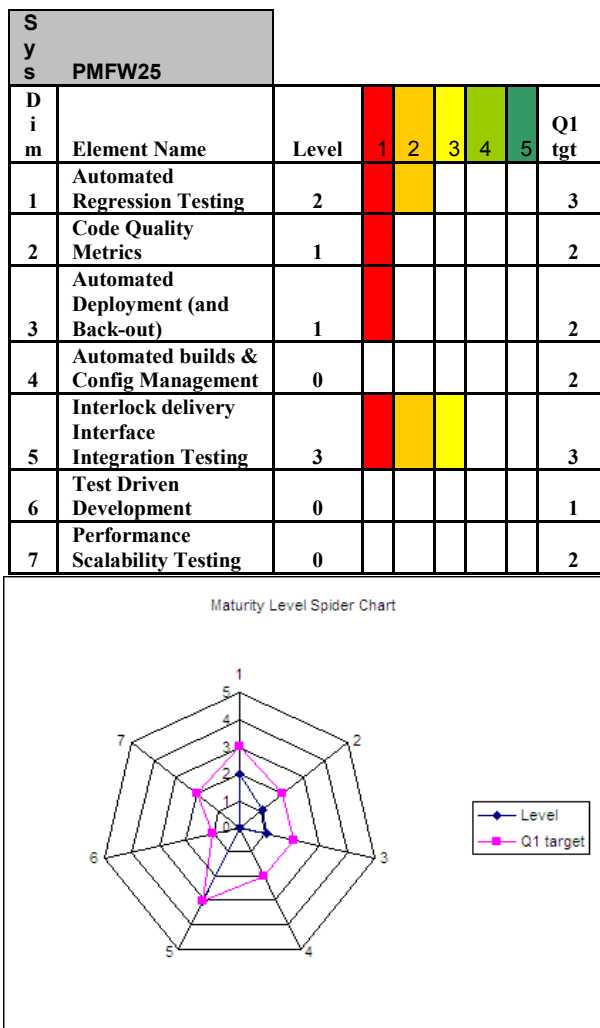


Figure 1: Example Maturity Assessment

Working with the business unit delivery stream lead, product flow maps between components were produced with component maturity level spider charts superimposed to identify potential trouble areas or opportunities. These areas were then prioritized and tackled. Stream reviews were held at the end of each release cycle between the central team and delivery stream lead while progress within each team was tracked fortnightly with quarterly reviews of targets. Wherever possible, stream leads were also placed in special interest groups to share ideas and practices across streams. As the component teams matured, domain experts increasingly worked with the stream leads to improve the maturity at the product and

service view, while jumping in where required to help with problematic components.

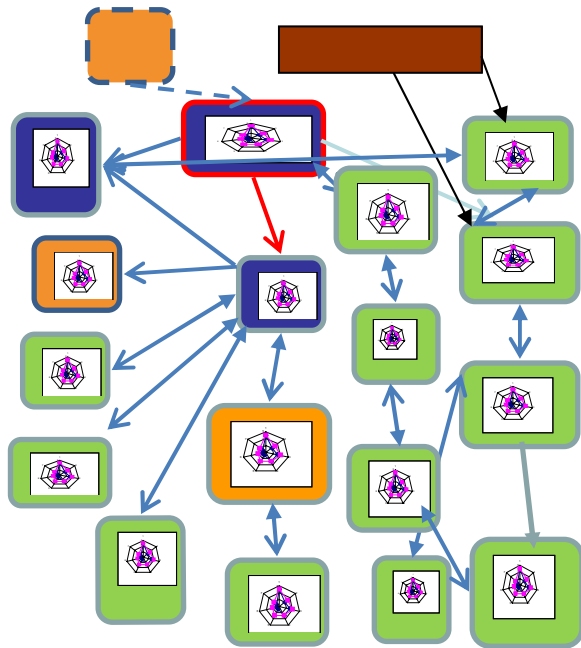


Figure 2: Example Product flow map

5. Program Trials

Two early adopter streams were identified to begin trials, one on an element of the backend infrastructure stack and one on a customer facing stack. After a rough start marshalling resources and changing embedded processes, the model showed striking improvements. Preliminary results from individual component teams that adopted the practices suggest an over 60% drop in defects entering End to End test, with failures in life dropping near zero. After an initial dip, velocity as of the teams improved between 8-20%. As hypothesized, two problems immediately surfaced. One was with improving the build and deployment velocity of one team that was holding up integration, while the other struggled with code quality. Both were reflected as potential problems in their assessments, and as impacts to the business became visible help was better targeted to assist.

As the number and scope of the trials increased, defects found later in the cycle continued to drop dramatically. As build/deploy/test cycles became more automated, repeatable and comprehensive, thousands of man-hours were saved in break-fix cycles, and new bottlenecks in other parts of the organization, including in the operations area and with some partner organizations, began to become more visible. Early

addressing of these, including some of which has influenced the technical design of the product, has further improved the operational and resource intensity of services. Early experiments in reduced release stream length have also shown promising results in organizational responsiveness. While the program is still in progress, it has been viewed as a huge success by the business. More data will be forthcoming as it becomes available.

6. Challenges

While the results thus far have been very positive, there are still numerous challenges that continue to be raised. The first has been with resistance from partners and suppliers. Many of the component teams are provided by offshore suppliers who see this as an encroachment on their territory and a potential threat to their business model. Contracts have been rewritten to accommodate for the new model, and some of the better supplier teams have begun to see the new way of working as a competitive advantage. However, it is still very much an uphill battle.

There has also been a challenge in education and outreach to key stakeholders throughout the business. Most of the model has been very engineering focused, which has turned off some from the business side. As improvements become tangibly realized, and the tools for expressing improvement continue to take shape, engagement and understanding has increased. It has been recognized that roles and responsibilities in the organization, such as End to End test and Operations, may change as a result of the work.

Commercial Off the Shelf (COTS) products, as well as hardware and process integration, do not integrate into this framework in a straightforward fashion. Modified models that still capture the goals of the seven dimensions have been developed, yet there is still an initiative to tighten them up to ensure that integration across COTS, process, hardware, and developed software flow smoothly. Difficulties to work with COTS products have already been slated to be abandoned by the business, as the value of the program has been viewed as outweighing the benefits of those products. It is believed that particularly stubborn vendors, especially those that unnecessarily complicate such activities as automated deployment and configuration, will be replaced by more flexible alternatives whenever possible. In the interim, challenges will continue.

8. References

- [1] Beck, Kent, and Cynthia Andres (2004). Extreme

Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional.

[2] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, Boston, 1999.

[3] Thomsett, R. (2007). Project Management Cultures: The Hidden Challenge. Agile Product and Project Management

[4] Highsmith, J. (2002). Agile Project Management: Principles and Tools. Cutter Consortium.

[5] Ken Schwaber and Mike Beedle, Agile Software Development with Scrum, ISBN 978-0130676344, Prentice-Hall, 2001.

[5] Martin, R.C., *Agile Software Development, Principles, Patterns and Practice*. 2002: Prentice Hall.

[6] T. DeMarco and T. Lister, Peopleware, 2nd Edition, Dorset House Publishing, New York, 1999.

[7] T. DeMarco, Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency, Broadway Books, New York, 2001.

[8] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, 1999.

[9] T. Gilb, Principles of Software Engineering Management, Addison-Wesley, Boston, 1988.

[10] E.M. Goldratt, Critical Chain, North River Press, Great Barrington, MA, 1997.

[11] C. Larman, Agile and Iterative Development: A Manager's Guide, Addison-Wesley, Boston, 2004.

[12] Mike Cohn, Agile Estimating and Planning, Addison-Wesley, 2005

[13] James C. Collins, Good to Great, Harper Business, 2001

[14] Michael N. Kennedy, Product Development for the Lean Enterprise, Oakela Press, 2003

[15] Matthew May, The Elegant Solution: Toyota's Formula for Mastering Innovation, Free Press, 2006

[16] Taiichi Ohno, Toyota Production System, English, Productivity, Inc. 1988, published in Japanese in 1978

[17] Mary Poppendieck and Tom Poppendieck, Lean Software Development, Addison Wesley, 2003

[18] Mary Poppendieck and Tom Poppendieck, Implementing Lean Software Development, Addison Wesley, 2006

[19] Ken Schwaber, Agile Project Management with SCRUM, Microsoft Press, 2004

[20] Steven Spear, Chasing the Rabbit, McGraw Hill, 2008

[21] Allen Ward, Lean Product and Process Development, Lean Enterprise Institute, 2007

[22] James P. Womack, Daniel T. Jones, and Daniel Roos, The Machine That Changed the World; the Story of Lean Production, Rawson and Associates; 1990