Improving Speed to Market in E-commerce

By focusing on organizational enablers and robust software engineering practices, e-commerce companies can shorten the development lifecycle, outmaneuver the competition and remain relevant in the eyes of customers.

Executive Summary

E-commerce providers are under intense pressure from a variety of stakeholders to continuously add new features at an increasingly rapid pace. Whether it is an urgent need to fulfill a new business requirement, keep up with a competitor or protect against an emerging threat, the stakes are high to stay both competitive and compliant with the latest security and regulatory requirements.

At the same time, it is extremely important for the e-commerce business to continue operating flawlessly, as each minute that an online store is down means lost revenue and diminished brand reputation. As a result, one of the biggest challenges for e-commerce leadership teams is to understand how to rapidly roll out features into production, while ensuring that the new capabilities work well with the rest of the e-commerce ecosystem. This pressure will only get worse with the exponential increase in complexity introduced by the ever-growing number of third-party and internal system integrations, as well as the constant threat of attrition of key talent.

As business and marketing teams aggressively push for features to be rolled out “as soon as possible,” development teams are left to battle with questions such as how to fit new requests into an already packed release schedule, how much testing is needed before rolling out the new release or feature, how exhaustive the testing should be, and what to do if the new feature breaks something else, given the complex set of interfaces and interactions.

In medium and large online businesses, today’s technology operations are built on robust technology stacks that power Web storefronts. They cannot function without a plethora of internal and external supporting systems, such as catalog, content, pricing, inventory, order management and marketing interfaces, to name a few.

Despite the complexity of the environment, e-commerce businesses can quickly bring new capabilities to market quickly. This white paper describes some of the proven best practices that successfully shorten the development lifecycle, as well as how organizations can use these practices to avoid mishaps and keep their e-commerce sites competitive by speeding time to market. These best practices apply to organizational setup, business processes, prioritization, technical design and tooling.

Organizational Enablers for Speed

Most organizations want to achieve more with less by improving operational efficiencies. Before we discuss the technical enablers for efficiency, we’ll...
look at a few organizational enablers, without which the desired results may not be achieved. One of the key ingredients for efficiency is to ensure strong teamwork among the stakeholders so that leaders can provide a focused direction to the entire team.

Given that a rich body of knowledge already exists around Agile software development’s best practices, we will focus on elements beyond implementing an Agile methodology. Many of the best practices we discuss are over and above the typical Agile implementation, and several of these are applicable to non-Agile methodologies, as well. (For more on the topic of Agile software development, see our white paper, “Software Development in the World of Code Halos.”)

- **Business, marketing and IT function as one team:** Mature organizations ensure that their release planning is carried out as a joint exercise with all their internal stakeholders. These plans are typically reviewed and adjusted on a monthly basis. A prioritized “product backlog” should be created so the team knows exactly what to work on and in which order. If all the stakeholders have participated in the prioritization, the most valuable features usually bubble to the top. This builds trust, predictability and teamwork, providing a sense of purpose and direction to the whole team. During execution, most reviews – including daily calls – should be a team effort between business, marketing and IT.

  If all the stakeholders have participated in the prioritization, the most valuable features usually bubble to the top. This builds trust, predictability and teamwork, providing a sense of purpose and direction to the whole team.

- **Paradigm shift from project-based to feature-based execution:** Moving from project-based to feature-based execution requires a mindset change across all teams. This cannot be done overnight. A release calendar is usually published, and the actual transformation begins with Release 1 and continues until the process is established as the default method for all online commerce delivery. In each release, the teams deliver committed features and functions. Every release should be followed by a retrospection, in which lessons learned are applied to subsequent releases.

  Typically, a transformation could be complete in about four to five such releases, giving the organization time to change gradually while delivering on its current commitments to stakeholders and customers.

- **Bandwidth allocation for unforeseen, last-minute requests and production issues:** Planning is absolutely essential, but the nature of e-commerce demands that last-minute requests will always exist; therefore, a percentage of the overall capacity and budget should be allocated to these needs. Doing so increases the predictability of delivery, as development of the core features would not need to be interrupted to accommodate these unplanned requests, and the associated dependency chains would not be impacted. In the best case, if there are no unexpected requests, the extra bandwidth would be used to address the next highest priority item(s) in the backlog.

- **Engage a “co-sourcing partner” not an “outsourcing vendor:** Organizations require a true partner when it comes to sourcing global services in today’s hyper-competitive e-commerce space. The right co-sourcing partner is not just about offering inexpensive resources; it should also offer deep experience with implementing best practices and e-commerce domain knowledge. Such a partner should also provide a flex capacity model that can quickly bring in additional resources when required.

Applying Software Engineering Practices

Along with organizational enablers, it is critical to implement a robust set of software engineering practices, many of which are being successfully leveraged today by several e-commerce shops and dot.coms. Many mature software engineering shops are also following these practices to improve speed to market and code quality. These practices include continuous integration, automated software testing, traceability and measuring code quality metrics to monitor compliance with agreed-upon best practices and coding standards (see Figure 1). Let us explore how each of these fits into the bigger puzzle.
Enablers for Speed to Market

**Figure 1**

- **Continuous integration:** As defined by Martin Fowler, continuous integration (CI) is a software development practice in which team members integrate their work frequently; usually, each developer integrates at least daily, leading to multiple integrations per day. Each integration is verified by an automated build (including testing) to detect errors as quickly as possible. Many teams find that this approach significantly reduces integration problems and allows a team to develop cohesive software more rapidly. CI is a prescribed Agile best practice, but its benefits can be realized by iterative (and even waterfall) teams. It requires a minimal initial investment in terms of a build server and associated scripting, but the ROI is worth every penny.

- **Automated software testing:** This kind of testing is made possible with a set of test suites. Starting with a minimalist smoke test suite, a more comprehensive regression testing suite needs to be built, using tools like Selenium. These can be automatically invoked after each build is deployed to the testing environment so that a pass/fail report is generated for each feature even before a human tester gets to it. Scripts to set up the right data sets in the right places are also carefully woven together, ensuring that the tests are valid and current. This requires an investment in automation test scripting resources to keep the scripts current from release to release. Testing is seen as an area that typically slows down the process, but automation helps recover some of that time.

Coding violations are usually the symptoms of less disciplined software development, the long-term price of which is prohibitively high.

- **Code quality measurements:** Measurement is critical to the long-term success of the team. Tools such as PMD, Checkstyle and JSLint can automatically flag coding standard violations, while Copy Paste Detector flags areas of code that have been mass-copied without being properly structured. Such coding violations are usually the symptoms of less disciplined software development, the long-term price of which is prohibitively high. Using tools like Crucible, metrics around manual code reviews can be collected to identify pockets of skill gaps and coaching needs. Note that this is intended to augment and not replace manual code reviews, as it is impractical to achieve comprehensive coverage with manual code reviews. All these metrics are usually consolidated in a dashboard such as SonarQube.

- **Traceability:** Traceability from requirements to code and deployment is maintained by individually keeping track of each feature, functionality and story, using tools such as Jira or Rally and interfacing them with source code control systems, using tools such as Fisheye. This provides the ability to exactly trace each requirement to code and vice versa, enabling
developers to better understand the dependencies and side effects of the changes they make. It also facilitates quickly rolling back a change if needed. The processes listed so far provide a strong level of mitigation against the constant threat of attrition, but they do not completely address it.

- **Support for multiple versions of services and key APIs:** Any service or API that is being built should support multiple versions in production; this ensures that consumers of the service do not need to synchronize their releases with the rollout of the service, and that the service can be rolled out on its own schedule without impacting others. The consuming applications can be modified later and tested against the new version of the service. This also allows consumers to quickly roll back to the older version if problems arise during adoption. This best practice has been very successfully adopted by Amazon Web Services and is the foundation of “everything-as-a-service” architectures.

- **Everything-as-a-service architecture:** With this approach, large application ecosystems are built from a multitude of services (which can be hosted in a distributed fashion, for example in a cloud), instead of being deployed as a monolithic package on traditional infrastructure. The exponential growth in the social, mobile, analytics and cloud space (or the SMAC Stack™), has resulted in many new capabilities being built in this way. This is also congruent with the design patterns of how dot.coms (such as Amazon) succeeded in building scalable systems. As other enterprises adopt this model, they are breaking up their formerly monolithic e-commerce systems into individual services, such as catalogs, offers, promotions, search, browse, cart, check-out, etc. and deploying them independently. This reduces the tight coupling and interdependency in an otherwise densely connected e-commerce ecosystem, thereby enabling faster, independent deployment of the parts.

- **Search-driven commerce:** A compelling offshoot is search-driven commerce. Unlike search as a component of the commerce stack, this approach breaks off the search and browse mechanism as its own entity in the ecosystem. In most e-commerce systems, search accounts for the majority of traffic (and hence computing power), far more than the actual commerce functions. Since typical commerce stacks charge licensing fees according to the size of the infrastructure, the bulk of the fee goes toward search, which is not really a core competency of the commerce stack.

As other enterprises adopt this model, they are breaking up their formerly monolithic e-commerce systems into individual services, such as catalogs, offers, promotions, search, browse, cart, check-out, etc. and deploying them independently.

- **Distributed software development:** This style of development is inevitable today, with globally distributed teams and the new paradigm of everything-as-a-service architectures. While Agile recommends co-location, it is not always practical given that the partners in the e-commerce ecosystem are spread across different organizations and around the globe. The best practices mentioned above mitigate the risks of distribution of the teams. The risks are further reduced by having daily “stand-ups,” or quick meetings, between the representatives of each team/ location, using the code quality metrics from these tools.

Looking Forward: An Alternative Approach

While these basic software engineering best practices increase predictability and quality, there are some additional e-commerce-specific best practices often used by the likes of Amazon, Google and other dot.coms. These are more suited to organizational cultures that can shift their mindsets away from the need to thoroughly and exhaustively test (almost) every possible use case and have all stakeholders sign off before anything is rolled out.

Instead, these organizations test the majority of use cases using an automated approach and then — in order to achieve speed-to-market — conduct a controlled, limited rollout by targeting a small fraction of the user base trying the new feature.
That way, if something breaks, it would only impact a very small percentage of the user base before it can quickly be turned off or rolled back to its previous state.

Some fast-growing sites perpetually stay in a “beta” state by accepting such a controlled risk, as it speeds up the lifecycle and offers two significant advantages: the ability to rapidly roll out features (and roll back if necessary) and the ability to measure the usage of features to determine ROI. While large corporations cannot perpetually stay in a beta state as dot.coms and startups can, we believe a middle ground can be established by fully testing the core e-commerce site, with some new features introduced as beta, using the following techniques.

- **Fail fast, fail often**: This is a model in which rapid experimentation is encouraged by rolling out several new ideas to see which one succeeds. To accomplish this without disturbing the online business, a sliver of the production environment is set aside for rolling out beta features. Some experiments can be quickly conducted in production (of course, with careful control of the throttle to send only qualified traffic), using a sliver of production; i.e., using one server in each layer with a newer version that can interoperate with the current software. This allows for the deployment of a new or updated component in parallel, with minimal impact to the existing ecosystem.

- **Feature switches**: All major and high-risk features and enhancements should be built so they can be turned off (or rolled back to a previous behavior) with a configuration switch. There should only be a few design patterns of implementing switches; if there are too many types of configuration settings, multiplied by many different lower environments, it will quickly become a nightmare to manage these configurations in each environment, requiring yet another dashboard or control system for configurations. We do not recommend this level of complexity, unless a multi-tenant system is being built, so moderation is key.

- **Multivariate testing**: Instrumenting these switches together using a multi-variate testing tool (such as Adobe Target) enables a fraction of this traffic to be sent to this new feature. It is usually turned on for about 5% to 10% of the traffic during the first week(s) of the rollout and is then notched up, eventually rolling forward or backward, based on the ROI metrics.

  The traditional reason for multivariate testing is to measure the difference in user adoption and behavior for the “before and after” scenarios under similar conditions. This helps product owners implement the right solution with the highest ROI. It also helps them refine their ROI estimation models so they can focus on the highest priority items.

  Dot.coms use this method for a more important reason, as well: To limit the damage in case something goes wrong in spite of all the testing that was done (or could not be done). Note that when some features impact multiple systems and batch processes in the ecosystem, it might take up to a week to realize something went wrong until all the back-end feeds and payment or refund processes are reconciled; this can sometimes be difficult to catch even in end-to-end testing. This limited exposure to the new feature contains the damage and minimizes manual recovery efforts. The throttle can be notched down to 0%, eliminating further damage until the issue is fixed.

- **Build the minimum viable product for 80% of the use cases**: Very often, organizations fall into analysis-paralysis trying to solve for all cases, including the edge cases. Instead, they should use the 80-20 rule, which states that the first 80% of functionality can be built for 20% of the effort, while the last 20% of functionality requires 80% of the effort. This rule should be applied to the organization’s advantage. After developing a minimum viable product for 80% of the use cases, companies should develop a manual catch-all process flow for the rest of the edge cases, or move them out of scope initially. The rest of the functionality can be built in as enhancements in a later release, while realizing ROI on the most used features that are deployed sooner. The requirements, scoping and estimation activities usually add four to six weeks to the schedule, and focusing on the minimum viable product cuts that time down, as well.
• **Incorporate metrics:** Metrics are a critical enabler – what gets measured usually improves. Our view is that this should go deeper than measuring the traditional operational metrics. For example, measuring the usage of every new feature would enable data-driven decisions to achieve ROI-driven prioritization. This helps focus limited resources on what matters the most. It is important to take a stab at the KPIs that will be affected by each new feature before it is built so that the business can measure against them and refine the model accordingly.

Leaders of mature, nimble organizations meet on a weekly basis and review the key metrics to ensure they are on track with their organizational goals and are spending their time on the right activities. These organizations typically have dashboards built to display these metrics so that the entire staff has visibility into the progress being made toward the annual goals, thereby significantly improving the chances of success.

Once the above pieces of the puzzle are in place, a typical problem scenario turns into something like this:

A marketing director comes up with an urgent requirement that requires changes in the browse, checkout, fulfillment and marketing systems. The teams are already working on a scheduled release, but fortunately, some bandwidth was allocated for unexpected requirements, and the co-sourcing partner chips in with its flex resources to quickly work in parallel on this requirement.

The code that is developed is traceable, using the tools provided, and can be quickly reverted either at the code level or configuration level or by diverting traffic, providing three layers of fallback in case of any issues. However, most issues are caught due to the automated testing and targeted rollout. In the worst case, the feature is disabled for new traffic until the issues are fixed. The other pre-scheduled features committed on the roadmap are not affected because of this unexpected requirement, resulting in a win-win situation.

The organization is now resilient. Note that all the work intake cannot be processed using this “unexpected requirement” route but only a small fraction that is truly exceptional.

With the right combination of design, tools, engineering, processes and teamwork, enterprises can react very rapidly to change and deliver high-quality releases to production with minimal downtime.

Note: All logos and trademarks mentioned in this paper belong to their respective owners. The tools mentioned above are illustrative, and their applicability to different situations might vary. The right toolset for any shop can be recommended only after an assessment of the current processes, toolsets and skillsets in place.
Footnotes
2 Selenium is a suite of tools to automate Web browsers across many platforms, http://www.seleniumhq.org/.
4 Crucible is a code review tool from Atlassian, https://www.atlassian.com/software/crucible/overview.
5 SonarQube is an open platform to manage code quality, http://www.sonarqube.org/.
7 Fisheye is a source code management software tool from Atlassian, https://www.atlassian.com/software/fisheye/overview.

About the Author
Prasad Tangirala is a Director in Cognizant’s E-Commerce Practice and is responsible for e-commerce delivery to large clients in the southern U.S. region using the best practices mentioned in this paper. He is based in Houston and has been with Cognizant for three years. Before he came to Cognizant, Prasad worked at Amazon.com and Apple, and he brings 20-plus years of experience leading the implementation of large-scale, customer-facing systems at Fortune 50 companies as well as startups. Prasad has a master’s in computer science from IIT Kanpur and is a certified Scrum master. He can be reached at Prasad.Tangirala@cognizant.com | www.linkedin.com/in/tvprasad.

About Cognizant
Cognizant (NASDAQ: CTSH) is a leading provider of information technology, consulting, and business process outsourcing services, dedicated to helping the world’s leading companies build stronger businesses. Headquartered in Teaneck, New Jersey (U.S.), Cognizant combines a passion for client satisfaction, technology innovation, deep industry and business process expertise, and a global, collaborative workforce that embodies the future of work. With over 75 development and delivery centers worldwide and approximately 199,700 employees as of September 30, 2014, Cognizant is a member of the NASDAQ-100, the S&P 500, the Forbes Global 2000, and the Fortune 500 and is ranked among the top performing and fastest growing companies in the world. Visit us online at www.cognizant.com or follow us on Twitter: Cognizant.