The Benefits of Policy-Driven Software Development

Software developers are making business decisions for you every day. The code they write determines the safety, security, performance, and reliability of the software that drives the business, giving them the power to introduce or minimize risks. The software developers make serves as the organization's primary interface to the customer. By allowing developers to make critical business decisions related to the software, managers, directors and C-level executives have delegated an extraordinarily high level of business responsibility to the developers. Their decisions directly affect immediate or future success, growth, damages or liabilities, as well as the stability of business leadership positions. The relationship between business outcomes, the interaction between business and development personnel, and the software code cannot and should not be understated.

The key to reining in these risks is to align software development activities with your organization's business goals. This can be achieved through "policy-driven development," which ensures that engineers deliver software according to your expectations. Policy-driven development involves 1) clearly defining expectations and documenting them in understandable polices, 2) training the engineers on the business objectives driving those policies, and 3) monitoring policy adherence in an automated, unobtrusive way. Integrating these principles into the development process gives businesses the ability to accurately and objectively measure productivity and application quality. The result is lower cost over the total software development lifecycle from build to support and reduced risk.

Adopting a policy-driven development process is key for achieving the following goals:

- Ensuring that engineers don't make tradeoffs that potentially compromise reliability and performance.
- Ensuring that engineers build security into the application, safeguarding it from potential attacks.
- Preventing defects that could result in costly recalls, litigation, or a damaged market position.
- Accurately and consistently applying quality processes.
- Gaining the traceability and auditability required to ensure continued policy compliance.

How Does Policy-Driven Development Work?

1. Management defines expectations for how software is written and tested.
2. Engineers are trained on how the expectations relate to the business objectives.
3. An automation infrastructure sits in the background of the development environment to automatically monitor compliance according to defined expectations. For example, a policy may require all code to follow a designated set of secure development practices, undergo peer review, and be verified by at least one test case.

4. When requirements are implemented, the automation infrastructure unobtrusively identifies policy violations.

5. If policy violations are identified, the infrastructure alerts the responsible engineer that policy violations need to be addressed immediately.

The ability to passively and unobtrusively monitor engineers' work is paramount to policy-driven development. If it is done properly, you will improve quality, increase productivity, lower development lifecycle costs from build to support, and reduce risks.

Elevating Guidelines to Policies

Many businesses traditionally view software development as a creative art that should be uninhibited by business-related policies. To this end, organizations have kept business processes on one side of the house and production on the other. But the fact of the matter is that the two are inseparable. Software engineers are making business decisions line by line as they develop, test, and deploy software. As such, it is appropriate, even necessary, that those in leadership positions answer the following question: How well do the engineers understand the business objectives driving the software they are developing?

Most organizations have guidelines for how they want their software to be developed, but a few barriers prevent these guidelines from truly being enforced and consistently followed:

1. **Cultural barriers:** Many businesses view software and engineering as "dark arts" that are magically done by creative people. Undeniably, engineers are highly skilled and their work very technical—but that does not excuse them from understanding and acting upon the business needs associated with the software they are developing.

2. **Lack of understanding:** Communicating requirements is a difficult task further complicated by the subjectivities of the business and development teams. Policy-driven development requires engineers to understand the business objectives behind the policies driving their code, which eliminates the need for business people to communicate with engineers on a technical level.

3. **Belief that adherence cannot be measured or enforced:** In most cases, engineers are working with an existing code base. Attempting to suddenly enforce a full set of policies would not only overwhelm the team, but it would make measuring and enforcing the policies extremely difficult. This is why policies should be introduced gradually.

The reason that organizations fail to overcome these barriers is that their *guidelines* are not elevated to *policies*. Guidelines describe *suggested behavior*, whereas policies define *expected behavior*. For example, "wash your hands after using the restroom" or "look both ways before crossing the street" are guidelines. They're great suggestions, but they cannot be fully enforced. Overcoming these barriers is essential for graduating from "nice-to-have" guidelines to "must-have" policies.

How to Get Started with Policy-Driven Development

1. Define What Policies You Want to Implement

Policies can be formed around any aspect of the development process, but to be effective they must be definable, enforceable, measureable, and auditable. You can define as many policies as necessary to help you achieve your goals, but you should start implementing them on a small scale. Introduce a few policies at a time, and as you become proficient with those policies, you can introduce more in small batches.

There are three types of policies that align with different goals and testing methods.

Process policies drive overarching sets of tasks through to completion and usually establish quality gates associated with the completion of a release cycle. Examples include policies that govern:

- Software development processes
- Software lifecycle processes
- Compliance with industry standards or recommendations (e.g., for secure application development)

People (or human) policies can be configured based on role, responsibility, skill-level, etc. They govern processes related to human interactions, for example:

- Task estimation
- Peer review
- Scope

Quality policies promote business objectives that are defined as nonfunctional requirements. They are measured by process monitors that deliver unprecedented visibility into the software. Examples include policies that cover:

- Application development (monitored by static code analysis)
- Code reliability (monitored by coverage analysis)
- Change impact (monitored by regression testing)

Adopting a combination of these policy types leads to a number of benefits that fundamentally improve the development process. You will be able to:

- **Find and isolate potential defects:** Code that is in violation of your policies is more likely to contain defects.

- **Gain early visibility into potential disasters:** As policies are further defined, they form a matrix that enables you to pinpoint specific types of defects.

- **Centralize organizational expectations and acceptable performance:** Maintaining policy documentation becomes less complex; software engineers actually have a consistent access point for learning about the policies.

- **Raise the 'policy IQ' of the engineering team:** With gentle coaching from the process monitors, engineers will gradually start writing and testing code in the expected manner—as a matter of habit.

- **Bring consistency to projects:** Consistency is achieved across implementation of requirements, usage of approved test data, and the design and execution of tests for various levels of granularity. This uniformity helps control complexity, as well as reduces risk.

2. Explain Policies in the Context of the Business Goal

Encourage the lead engineer to work with the business analyst to document policies in natural, understandable human language within the context of the associated business goals. If the connection between the policy and goal isn't clear, the employee can't be expected to understand why they need to follow the policy. This, of course, is in direct violation of the definition of a policy as setting expectations. For example, if an organization wants a policy that requires all methods to be unit tested, management should describe how the business benefits from the policy.

3. Train Software Engineers on Policies

Beyond documenting the how and the why of your policies, you also want to take steps to ensure that the connection between the two is clear. The absence of training is the number one reason policies fail. If a policy requires code to be structured in a certain way, the engineer may not immediately see the potential for the bug that the structure is intended to prevent. If the engineer doesn't make the connection during this cycle or even the next few cycles, then the policy looks more like a guideline to him or her. Thus, the code may not be properly structured before the product goes to market and defects may surface in the field. At this point, the implementation of the policy has failed.

4. Use Automation to Drive a Sustainable Process

Automating policy monitoring, as well as the process for routinely notifying engineers of violations, ingrains policies into the day-to-day workflow. Without this level of automation policies will quickly fade and expected behavior will degrade back into suggested behavior. When policies revert to guidelines, software engineers are more likely to apply the quality strategies ad-hoc, as opposed to integrating them into their workflow.

Testing done at the end of the project (rather than incrementally as a regular part of the software development lifecycle) habitually returns an enormous log of errors that overwhelm engineers. The likelihood that the errors will be mitigated is minimal. At best, engineers cherry-pick the errors that are easiest to fix. At worst, team leaders sign off on the project as is and fix with patches when defects are exposed, which may weaken market position, damage the business reputation, or even result in litigation.

An automated, exception-based system also creates efficiency by making incremental testing a part of the engineer's daily process. Coding is a complex process that requires concentration and the right frame of mind. When requests for fixes are submitted outside of the engineer's routine, it takes him or her out of their rhythm, which is extremely difficult regain.

Efficiency is also gained by allowing junior engineers to be coached by the policy notifications, which improves their understanding and keeps senior engineers on task. The key is that the policies are in a natural language, understandable, and meaningful to engineers in context of the business.

5. Use Automation to Ensure Traceability and Auditability

An automated system that logs and traces the activity that occurred during development is the most efficient way to measure, analyze, and enforce strict quality processes. It is ideal for demonstrating compliance with the expected policy, as well as ensuring accountability.

Manually defining a strict quality process, collecting the metrics associated with a broad set of potentially distributed policies, and proving that the process was actually followed can be overwhelming. At best, it is a time-consuming activity that introduces potential breakpoints that introduce unnecessary risk.

A centralized infrastructure capable of managing policies will go a long way toward realizing the benefits of policy-driven development. Ideally, a single platform that monitors adherence to multiple types of policies and enables effective implementation will be in place to deliver the traceability and auditability required for certification and for audit purposes.

Conclusion

The greatest factor limiting the ability to create defect-free software is the inability to conceive policies to govern its development. In this regard, it is fortunate that humans are naturally creative. At the same time, creating defect-free software requires discipline, which is cumbersome and time-consuming.

Creating policies prior to the need does not come naturally, so the resistance to policy-driven development is understandable. The resistance, though, is misguided by the prejudices engrained by traditional software development strategies. We are confident that this resistance will fade as organizations begin to realize that elevating guidelines to definable, enforceable, measureable, and auditable policies will drastically improve how they develop software. It not only helps organizations accurately measure application quality and development productivity, but also improve it by preventing errors and eliminating waste.

Fortunately, there are tools available to make implementing policy-driven development no more harrowing than any other development methodology. It is our position that with the testing tools currently available, we should have every reason to employ our creative minds to creating defect-free software.

About Parasoft

For 25 years, Parasoft has researched and developed software solutions that help organizations deliver defect-free software efficiently. By integrating end-to-end testing, dev/test environment management, and software development management, we reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, functional testing with requirements traceability, service virtualization, and more. The majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently. For more information, visit the Parasoft website and ALM Best Practices blog.

Contacting Parasoft

USA
101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe
France: Tel: +33 (1) 64 89 26 00
UK: Tel: + 44 (0)208 263 6005
Germany: Tel: +49 731 880309-0
Email: info-europe@parasoft.com

Other Locations
See http://www.parasoft.com/contacts

Author Information

This paper was written by:

- Adam Trujillo (atrujillo@parasoft.com), Technical Writer at Parasoft
- Wayne Ariola (wayne.ariola@parasoft.com), VP of Strategy at Parasoft
- Cynthia Dunlop (cynthia.dunlop@parasoft.com), Lead Technical Writer at Parasoft