

The arrival of modern technologies like containers and the emergence of architectures like *Continuous Integration and Continuous Delivery* (CI/CD) have made the practice of software development increasingly complex—and overwhelming. There is no best practice guide, no one size fits all, and good looks different for everyone.

The framework below endeavors to demystify these complexities and create an accepted, best-of-breed lifecycle that provides a solid foundation, around which you can build your organization's plan—and achieve your goals.

CONTINUOUS INTEGRATION CONTINUOUS DELIVERY CONTINUOUS OPTIMIZATION Sizing/Breakdown Release often in microincrements Keep it Simple, Stupid (KISS) and always deliver functional Don't try to be clever—write software. This is key to being on functional code that is elegant top of user requirements and **Test-Driven Development** Behavior-Driven but simple and easy to maintain. delivering relevance. Development (BDD) DevSecOps Non-Functional Develop to deliver business Write the tests that need to be Requirements (NFRs) Include security as early as passed before writing any code. features rather than technical possible in the development cycle Define NFRs early and ensure they features. These business features to avoid applying security are part of the development cycle need to be objective, attainable, hardening with brute force later. or sprint and not an afterthought. and monitorable. While **Test** is a distinct phase, it is actually a continual part of the entire cycle. You need to be constantly validating ideas, deployments, user experience, and features throughout the entire cycle to get rapid and continuous feedback. PLAN CODE BUILD **TEST** RELEASE COLLABORATE **DEPLOY OPERATE MONITOR ANALYZE** • Agile, Scrum, and Kanban Traffic/user migration Dev frameworks Continuous build Observability ITSM technologies facilitate Unit testing Code repository Operational processes Analysis of key application Integrated development Security checks Sprint planning Integration testing A/B or canary testing DevOps ownership Reliability the approval acquisition Container registry (transaction latency, tps, etc) & NFR validation Sizing Environment (IDE) A/B build On-call rotas Record NFRs and infrastructure metrics (cpu, process for optimization insights. Real user acceptance Security testing Release committed and tagged Code repository Working code (master) Immutability testing provide feedback Insights that are approved are fed NFR validation memory, disk i/o, etc) to determine Task breakdown Documentation improvements Feature validation Checkout/pull request/commit Fast feedback License validation Operational feedback back to the deployment phase for Release notes Continuous testing optimization possibilities. Fast fix Vagrant/Containers Common Vulnerabilities and Release documentation Continuous penetration testing execution. Testing can continue through the At this point, rollback should not be Application utilization pattern Sandbox Image build Exposures (CVE) checks Automated documentation (via machine learning) Deploy phase, especially when there an option (unless some other Fast feedback Validation against best practices Operational documentation Generation of business Small tasks allow for rapid builds, processes have failed), but the is a change to end-user experience. Clean Code validation Operational handover Fast feedback with a Dev minimizing merge conflicts and *This allows for final consumer* feedback and improvement process intelligence reports for app Human acceptance environment/framework allows for keeping master code functional at feedback before committing to the owners' approval *minimum viable product (MVP)* all times. Rejected merges or full release. Automated testing is often part of non-functional commits get the build process, so this isn't rework. f final user testi always a separate phase. A reworked. ails, a rollback successful build is a result of elease may be passing all validation tests. necessary eliver machine-learned ptimization recommendations to arameter repository, heck for approval, and lynamically rewrite If a **Build** or **Test** fails, review the objections and nfrastructure as code determine the resolution, then return to the **Code** phase



