1.1 Patterns and Anti-Patterns

1.1.1 Continuous Integration

- Build software at every change:
  - Pattern – run a software build with every change applied to the central code repository;
  - Anti-Pattern – scheduled builds, nightly builds, building periodically, building exclusively on developer's machines, not building at all.
- Task-Level Commit:
  - Pattern – organize source code changes by task-orientated units of work and submit changes as a Task-Level Commit;
  - Anti-Pattern – Keeping changes local to development for several days and stacking up changes until committing all changes – this often causes build failures or requires complex troubleshooting.
- Label Build:
  - Pattern – Tag or label the build with unique names so that you can refer to run the same build at another time;
  - Anti-Pattern – Not labelling builds, using revisions or branches as 'labels'.
- Build Management - Automated Builds:
  - Pattern – i) Automate all activities to build software from a source without manual configuration. ii) Create build scripts that will be executed by a CI system so that software is built at every change;
  - Anti-Pattern – Continually repeating the same processes with manual builds or partially automated builds requiring numerous manual configuration activities.
- Build Practices - Pre-Merge Builds:
  - Pattern – Verify that your changes will not break the integration build by performing a pre-merge build either locally or using CI system;
  - Anti-Pattern – Checking in changes to a version control repository without running a build on a developers environment
- Build Practices - Continuous Feedback:
  - Pattern – Send automated feedback from the CI system to development team members involved in the build;
  - Anti-Pattern – Sending minimal feedback that provides no insight into the build failure or is non-actionable. Sending too much feedback, including team members uninvolved with the build. This is eventually treated like spam, which causes people to ignore valid messages.
- Build Practices - Expeditious Fixes:
  - Pattern – Fix build errors as soon as they occur;
  - Anti-Pattern – Allowing problems to stack up (build entropy) or waiting for them to be fixed in future builds.
- Build Practices - Developer Documentation:
  - Pattern – Generate developer documentation with builds based on checked-in source code;
  - Anti-Pattern – Manually generating developer documentation. This is both a burdensome process and one in which the information becomes useless quickly because it does not reflect the checked-in source code.
- Build Configuration – Independent Build:

    - Pattern – Create build scripts that are decoupled from IDEs but can be invoked by an IDE. These build scripts will be executed by a CI system as well so that software is build at every change;
    - Anti-Pattern – Relying on IDE settings for Automated Build. Build cannot run from the command line.
- Build Configuration – Dedicated Resources:
    - Pattern – Run master builds on a separate dedicated machine or cloud service;
    - Anti-Pattern – Relying on existing environmental and configuration assumptions (can lead to the "but it works of my laptop" problem).
- Build Configuration – Single Command:
    - Pattern – Ensure all build processes can be run through a single command;
    - Anti-Pattern – Requiring people or servers to enter multiple commands and procedures in the deployment process, such as copying files, modifying configuration files, restarting a server, setting passwords and other repetitive, error prone actions.
- Build Configuration – Externalize and Tokenize Configuration:
    - Pattern – i) Externalize all variable values from the application configuration into build-time properties. ii) Use tokens so the build process knows where to add variable values;
    - Anti-Pattern – Hardcoding values in configuration files or using GUI tools to do the same.
- Database – Scripting Changes and Use of Version Control System:
    - Pattern – All changes made to a database during development should be recorded into database scripts that can be run on every database on every platform (including developers machines) hosting the project. Scripts and supporting data must be stored in the version control system;
    - Anti-Pattern – Expecting DBAs to manually compare databases between platforms for changes or to create on-off scripts that are only good for updating a single platform. Storing of scripts in an alternative location e.g. file share
- Database – Sandbox:
    - Pattern – i) Create a light weight version of a database with only enough records to test functionality. Ii) Use a CLI to populate the local database sandboxes for each developer, tester and build server, iii) Use this data in development environments to expedite test execution;
    - Anti-Pattern – Sharing a single development database.
- Testing and Code Quality – Automated Tests:
    - Pattern – Write automated tests for each code path, both success testing and failure testing;
    - Anti-Pattern – Not running tests. No regression testing. Manual testing.
- Testing and Code Quality – Build Quality Threshold:
    - Pattern – i) Notify team members of code aberrations such as low code coverage or the use of coding anti-patterns, ii) Fail a build when a project rule is violated, iii) Use continuous feedback mechanisms to notify team members;
    - Anti-Pattern – Deep dive reviews of every code change. Manually calculating or guesstimating code coverage.
- Testing and Code Quality – Smoke Tests:
    - Pattern – Create smoke tests that can be used by CI servers, developers, QA and testing as a pre-check to confirm the most important functionality as they work, or before committing resources to a full build;

  - Anti-Pattern – Manually running functional tests. Forcing QA to run the full suite before every session. Manually checking deployment sensitive sections of the project.
- Continuous Integration – Commit Often:
  - Pattern – Each team member checks in regularly to trunk – at least once a day but preferably after each task to trigger the CI system;
  - Anti-Pattern – Source files are committed less frequently than daily due to the number of changes from the developer.
- Continuous Integration – Stop the Line:
  - Pattern – Fix software delivery errors as soon as they occur. No one checks in on a broken build, as the fix becomes the highest priority.
  - Anti-Pattern – Builds stay broken for long periods of time, thus preventing developers from checking out functioning code.

1.1.2 Continuous Delivery

- Configuration Management – Configurable Third-Party Software:
  - Pattern – Evaluate and use third party software that can be easily configured, deployed and automated;
  - Anti-Pattern – Procuring software that cannot be externally configured. Software without an API or CLI that forces teams to only use GUI.
- Configuration Management – Configuration Catalogue:
  - Pattern – Maintain a catalogue of all options for each application, how to change these options and storage locations for each application. Automatically create this catalogue as part of the build process;
  - Anti-Pattern – Configuration options are not document. The catalogue of applications and other assets is “tribal knowledge”.
- Configuration Management – Mainline:
  - Pattern – Minimise merging and keep the number of active code lines manageable by developing on a mainline;
  - Anti-Pattern – Multiple branches per project.
- Configuration Management – Merge Daily:
  - Pattern – Changes committed to the mainline are applied to each branch on at least a daily basis;
  - Anti-Pattern – Merging every iteration once a week or less often than once a day.
- Configuration Management – Protected Configuration:
  - Pattern – Store configuration information in secure remotely accessible locations such as a database, directory or registry;
  - Anti-Pattern – Open text passwords and / or single machine or share.
- Configuration Management – Repository:
  - Pattern – All source files – executable core, configuration, host environment, and data – are committed to a version control repository;
  - Anti-Pattern – Some files are checked in, others, such as environment configuration or data changes, are not. Binaries – which can be recreated through the build and deployment process – are checked in.
- Configuration Management – Short Lived Branches:
  - Pattern – Branches must be short lived – ideally less than a few days and never more than an iteration;

    - Anti-Pattern – Branches that last more than an iteration. Branches by product feature that live past a release.
- Configuration Management – Single Command Environment:
    - Pattern – Check out the projects version control repository and run a single command to build and deploy the application to any accessible environment, including the local development;
    - Anti-Pattern – Forcing the developer to define and configure environment variables. Making the developer install numerous tools for the build/deployment to work.
- Configuration Management – Single Path to Production:
    - Pattern - Configuration management of the entire system - source, configuration, environment, and data. Any change can be tied back to a single revision in the version-control system;
    - Anti-Pattern – Parts of the system are not versioned. Inability to get back to a previously configured software system.
- Testing – Automate Tests:
    - Pattern – Automate the verification and validation of software to include unit, component, capacity, functional, and deployment tests;
    - Anti-Pattern – Manual testing of units, components, deployment, and other types of tests.
- Testing – Isolate Test Data:
    - Pattern – Use transactions for database-dependent tests (e.g. component tests) and roll back the transaction when done. Use a small subset of data to effectively test behaviour;
    - Anti-Pattern – Using a copy of production data for Commit Stage tests. Running tests against a shared database.
- Testing – Parallel Tests;
    - Pattern – Run multiple tests in parallel across hardware instances to decrease the time in running test;
    - Anti-Pattern – Running tests on one machine or instance. Running dependent tests that cannot be run in parallel.
- Testing – Stub Systems:
    - Pattern – Use stubs to simulate external systems to reduce deployment complexity;
    - Anti-Pattern – Manually installing and configuring interdependent systems for Commit Stage build and deployment.
- Deployment Pipeline:
    - Pattern – A deployment pipeline is an automated implementation of your application's build, test, deploy, and release process;
    - Anti-Pattern – Deployments require human intervention (other than approval or clicking a button). Deployments are not production ready.
- Deployment Pipeline – Value Stream Map:
    - Pattern – Create a map illustrating the process from check in to the version control system to the software release to identify process bottlenecks;
    - Anti-Pattern – Separately defined processes and views of the checkin to release process.
- Build and Deployment Scripting – Dependency Management:
    - Pattern – Centralise all dependent libraries to reduce bloat, class path problems, and repetition of the same dependent libraries and transitive dependencies from project to project;
    - Anti-Pattern – Multiple copies of the same binary dependencies in every project. Redefining the same information for each project. This is classpath hell.
- Build and Deployment Scripting – Common Language:

    - Pattern – As a team, agree upon a common scripting language — such as Perl, Ruby, or Python — so that any team member can apply changes to the Single Delivery System;
    - Anti-Pattern – Each team uses a different language making it difficult for anyone to modify the delivery system reducing cross-functional team effectiveness.
- Build and Deployment Scripting – Fail Fast:
    - Pattern – Fail the build as soon as possible. Design scripts so that processes that usually fail run first. These processes should be run as part of the commit stage;
    - Anti-Pattern – Common build mistakes are not uncovered until late in the deployment process.
- Build and Deployment Scripting – Fast Builds:
    - Pattern – The commit build provides feedback on common build problems as quickly as possible — usually in under 10 minutes;
    - Anti-Pattern – Throwing everything into the commit stage process, such as running every type of automated static analysis tool or running load tests such that feedback is delayed.
- Build and Deployment Scripting – Scripted Deployment:
    - Pattern – All deployment processes can be written in a script, checked in to the version-control system, and run as part of the single delivery system;
    - Anti-Pattern – Deployment documentation is used instead of automation. Manual deployments or partially manual deployments.
- Build and Deployment Scripting – Unified Deployment:
    - Pattern – The same deployment script is used for each deployment. The protected configuration – per environment — is variable but managed;
    - Anti-Pattern – Different deployment script for each target environment or even for a specific machine. Manual configuration after deployment for each target environment.
- Deploying and Release Applications – Binary Integrity:
    - Pattern – Build your binaries once, while deploying the binaries to multiple target environments, as necessary;
    - Anti-Pattern – Software is built in every stage of the deployment pipeline.
- Deploying and Release Applications – Canary Release:
    - Pattern – Release software to production for a small subset of users (e.g. 10%) to get feedback prior to a complete rollout;
    - Anti-Pattern – Software is released to all users at once.
- Deploying and Release Applications – Blue Green Deployments:
    - Pattern – Deploy software to a non-production environment (call it blue) while production continues to run. Once it's deployed and "warmed up," switch production (green) to non-production and blue to green simultaneously;
    - Anti-Pattern – Production is taken down while the new release is applied to production instance(s).
- Deploying and Release Applications – Dark Launching:
    - Pattern – Launch a new application or features when it affects the least number of users;
    - Anti-Pattern – Software is deployed regardless of number of active users.
- Deploying and Release Applications – Rollback Release:
    - Pattern – Provide an automated single command rollback of changes after an unsuccessful deployment;
    - Anti-Pattern – Manually undoing changes applied in a recent deployment. Shutting down production instances while changes are undone.
- Deploying and Release Applications – Self Service Deployment:

    - Pattern – Any Cross-Functional Team member selects the version and environment to deploy the latest working software;
    - Anti-Pattern – Deployments released to team are at specified intervals by the “build team.” Testing can only be performed in a shared state without isolation from others.
- Infrastructure and Environments – Automate Provisioning:
    - Pattern – Automate the process of configuring your environment to include networks, external services, and infrastructure;
    - Anti-Pattern – Configured instances are “works of art” requiring team members to perform partially or fully manual steps to provision them.
- Infrastructure and Environments – Behaviour Driven Monitoring:
    - Pattern – Automate tests to verify the behaviour of the infrastructure. Continually run these tests to provide near real-time alerting;
    - Anti-Pattern – No real-time alerting or monitoring. System configuration is written without tests.
- Infrastructure and Environments – Immune System:
    - Pattern – Deploy software one instance at a time while conducting behaviour-driven monitoring. If an error is detected during the incremental deployment, a rollback release is initiated to revert changes;
    - Anti-Pattern – Non-incremental deployments without monitoring.
- Infrastructure and Environments – Lockdown Environments:
    - Pattern – Lock down shared environments from unauthorized external and internal usage, including operations staff. All changes are versioned and applied through automation;
    - Anti-Pattern – The “Wild West:” authorised user can access shared environments and apply manual config changes, putting the environment in an unknown state leading to deployment errors.
- Infrastructure and Environments – Production Like Environments:
    - Pattern – Target environments are as similar as possible to production
    - Anti-Pattern – Environments are “production-like” only weeks or days before a release. Environments are manually configured and controlled.
- Infrastructure and Environments – Transient Environments:
    - Pattern – Utilizing the Automate Provisioning, Scripted Deployment, and Scripted Database patterns. Any environment should be capable of terminating and launching at will;
    - Anti-Pattern – Environments are fixed to “DEV," "QA,” or other predetermined environment.
- Data – Database Sandbox:
    - Pattern – Create a lightweight version of your database – using the Isolate Test Data pattern. Each developer uses this lightweight DML to populate his local database sandboxes to expedite test execution;
    - Anti-Pattern – Shared database. Developers and testers are unable to make data changes without it potentially adversely affecting other team members immediately.
- Data – Decouple Database:
    - Pattern – Ensure your application is backward and forward compatible with your database so you can deploy each independently;
    - Anti-Pattern – Application code data are not capable of being deployed separately.
- Data – Database Upgrade:
    - Pattern – Use scripts to apply incremental changes in each target environment to a database schema and data;

    - Anti-Pattern – Manually applying database and data changes in each target environment.
- Data – Scripted Database:
    - Pattern – Script all database actions as part of the build process;
    - Anti-Pattern – Using data export/import to apply data changes. Manually applying schema and data changes to the database.
- Incremental Development – Branch by Abstraction:
    - Pattern – Instead of using version-control branches, create an abstraction layer that handles both an old and new implementation. Remove the old implementation;
    - Anti-Pattern – Branching using the version-control system leading to branch proliferation and difficult merging. Feature branching.
- Incremental Development – Toggle Features:
    - Pattern – Deploy new features or services to production but limit access dynamically for testing purposes;
    - Anti-Pattern – Waiting until a feature is fully complete before committing the source code.
- Collaboration – Delivery Retrospective:
    - Pattern – For each iteration, hold a retrospective meeting where everybody on the Cross-Functional Team discusses how to improve the delivery process for the next iteration;
    - Anti-Pattern – Waiting until an error occurs during a deployment for Dev and Ops to collaborate. Having Dev and Ops work separately.
- Collaboration – Cross-Functional Teams
    - Pattern – Everybody is responsible for the delivery process. Any person on the Cross-Functional Team can modify any part of the delivery system;
    - Anti-Pattern – Siloed teams: Development, Testing, and Operations have their own scripts and processes and are not part of the same team.
- Collaboration – Root-Cause Analysis:
    - Pattern – Learn the root cause of a delivery problem by asking “why” of each answer and symptom until discovering the root cause;
    - Anti-Pattern – Accepting the symptom as the root cause of the problem.
- Collaboration – Visible Dashboards:
    - Pattern – Provide large visible displays that aggregate information from your delivery system to provide high quality feedback to the Cross-Functional Team in real time;
    - Anti-Pattern – Email-only alerts or not publishing the feedback to the entire team.