Pull Requests

Changes to any main codebase - main branch in Git repository, for example - must be done using pull requests (PR).

Pull requests enable:

- Code inspection - see Code Reviews
- Running automated qualification of the code
  - Linters → agreed coding conventions
  - Compilation → run without Error or warning
  - Unit tests
  - Integration tests etc.

The requirements of pull requests can and should be enforced by policies, which can be set in the most modern version control and work item tracking systems. See Evidence and Measures section for more information.

General Process

1. Implement changes based on the well-defined description and acceptance criteria of the task at hand
2. Then, before creating a new pull request: * Make sure the code conforms with the agreed coding conventions * This can be partially automated using linters * Ensure the code compiles and runs without errors or warnings * Write and/or update tests to cover the changes and make sure all new and existing tests pass * Write and/or update the documentation to match the changes
3. Once convinced the criteria above are met, create and submit a new pull request adhering to the pull request template
4. Follow the code review process to merge the changes to the main codebase

The following diagram illustrates this approach.

```
sequenceDiagram
New branch->>+Pull request: New PR creation
```

```
Pull request->>+Code review: Review process
Code review->>+Pull request: Code updates
Pull request->>+New branch: Merge Pull Request
Pull request-->>-New branch: Delete branch
Pull request ->>+ Main branch: Merge after completion
New branch->>+Main branch: Goal of the Pull request
```

Size Guidance

We should always aim to keep pull requests small. Small PRs have multiple advantages:

- They are easier to review; a clear benefit for the reviewers.
- They are easier to deploy; this is aligned with the strategy of release fast and release often.
- Minimizes possible conflicts and stale PRs.

However, we should keep PRs focused - for example around a functional feature, optimization or code readability and avoid having PRs that include code that is without context or loosely coupled. There is no right size, but keep in mind that a code review is a collaborative process, a big PRs could be difficult and therefore slower to review. We should always strive to have as small PRs as possible that still add value.

Best Practices

Beyond the size, remember that every PR should:

- be consistent, → Should solve 1 goal (1 story)
- not break the build, and
- include related tests as part of the PR.

Be consistent means that all the changes included on the PR should aim to solve one goal (ex. one user story) and be intrinsically related. Think of this as the Single-responsibility principle in terms of the whole project, the PR should have only one *reason to change* the project.

Start small, it is easier to create a small PR from the start than to break up a bigger one.

These are some strategies to keep PRs small depending on the "cause" of the inevitability, you could break the PR into self-container changes which still add value, release features that are hidden (see feature flag, feature toggling or canary releases) or break the PR into different layers (for example using design patterns like MVC or Observer/Subject). No matter the strategy.

Pull Request Description

Well written PR descriptions helps maintain a clean, well-structured change history. While every team need not conform to the same specification, it is important that the convention is agreed upon at the start of the project. discuss convention

One popular specification for open-source projects and others is the Conventional Commits specification, which is structured as:

```
<type>[optional scope]: <description>

[optional body]

[optional footer]
```

The `<type>` in this message can be selected from a list of types defined by the team, but many projects use the list of commit types from the Angular open-source project. It should be clear that `scope`, `body` and `footer` elements are **optional**, but having a required `type` and short description enables the features mentioned above.

See also Pull Request Template

Resources

- Writing a great pull request description
- Review code-with pull requests (Azure DevOps)
- Collaborating with issues and pull requests (GitHub)
- Google approach to PR size
- Feature Flags
- Facebook approach to hidden features
- Conventional Commits specification
- Angular Commit types

Last update: February 15, 2024