Entwicklungsguidelines

nope

24. November 2023

Inhaltsverzeichnis

Code

- veröffentlichbarer Code
- lesbarer Code
- wartbarer Code
- Trennung von Technik und Fachlichkeit

Ziel

Grundsätzlich soll Code erzeugt werden der auf github veröffnetlicht werden kann (nicht muss)

Sprache

- Entwickliung in Englisch
- Kommentare in Englisch
- Dokumentation ausserhalb in deutsch, englisch oder mandarin

Ziel

Form follows function - Pragmatische Auswahl der Sprache nach Bedarf, Anwendung und

Formatter

- google code style als Basis
- google formatter verwenden
- erreichbar über git-hooks bei commit und push
- erforderlich bei push in den main branch

Prämisse

Eigene branches (feature oder bugfix) können eigene Formatierung haben. Der Main branch muss konform zum google code style sein.

Aussagekräftige Funktions- und Methodennamen

- it, has, does, can liefert einen **boolean** zurück

Beispiel

public boolean **is**Active()
protected boolean **has**License()

Aussagekräftige Funktions- und Methodennamen

- Funktion/Methode, die einen Wert zurückliefert, enthält entsprechendes Verb
  (get, fetch, compute, generate, create, calculate)

Beispiel

public int **get**TotalCostFromStart()
private int **calculate**EuclideanDistance(Node source, Node target)
public String **generate**InviteCode(Role role, String identifier)
private ValueGraph<Node, Integer>
createGraph(List<List<Node> > nodes)
private int **compute**Score(User user)

Aussagekräftige Funktions- und Methodennamen

- nichtsprechende Verben liefern nichts zurück (void!) → Exception

Beispiel

public void **check**Credentials())

- für checks bietet sich die Klasse Preconditions von guava an.

Beispiel

checkArgument(value >= 0, “input is negative: %s", value);

Aussagekräftige Funktions- und Methodennamen

- sprechende Funktionssignaturen sind besser als Dokumentation

Beispiel

long ~~getTimestamp()~~ → long getTimestamp**InMilliseconds**()

Branches

- Main-branch heisst main
- Entwicklungsbranches sind (kurze) featurebranches
- Merge in main erfolgt immer ueber pull-request
- In der Regel sollte ein Code-review stattfinden
- Featurebranchname:
  $< JIRA - TicketNo > \_ < sprechende\_Bezeichnung >$

Ziel

Im main-branch sollte immer lauffaehiger, getesteter und deployfahiger Code liegen

git

- squash commits bei merge
- Grundsaetzlich nur fast-forward merges
- Ansonsten: *main* $\rightarrow$ *feature* $\rightarrow$ *main*
- rebase $\rightarrow$ Gehe in das Gefängnis. Begib Dich direkt dorthin. Gehe nicht über Los. Ziehe nicht EUR 1000 ein

Ziel

Uebersichtliche und nachvollziehbare Entwicklungszyklen sowie git als single point of truth

CI

- Verwendung der Basis-CI
- Kein voodoo in der CI (dwyce - do what your customer expect)
- Feature-branch pushes sollen KEINE Artefakte o.ae. erzeugen
- Ein push auf main muss ein Artefakt erzeugen (release)

Ziel

Die CI stellt sicher dass getesteter und lauffaehiger Code im repositry liegt. Zudem stellt Sie sicher dass Artefakte luaffaehig und versioniert abgelegt werden

CI 2

- Artifakte sollten nur als release abgelegt werden
- CI fuehrt tests durch z.B. spotless:check, pmd:check, sonarcube, jacoco ...
- Builds sollen reproduzierbar sein

Ziel

Einige Metriken der CI sollen auswertbar sein und Codeveraenderungen aussagekraeftig anzeigen koennen.

Git Repository / gitlab

- Multimodul vor Multiprojekt
- Branches muessen jederzeit baubar sein
- Ein Entwickler sollte keine Anpassungen vornehmen muessen um einen Code bauen zu koennen
- Voodoo sollte vermieden werden

Ziel

Das Gitlab sollte jederzeit in einem Zustand sein sodass es fuer die Allgemeinheit als open source zur Verfuegung gestellt werden koennte. (Rein qualitativ betrachtet)

Nomenklatur

- artifactId == Name des Moduls
- groupdid == de.Firmenname.bereichsname.projekt
- package name == groupdid.artifactid:Versionsnummer

Ziel

Einheitliche Namensstrukturen, sprechende Namen und Nachvollziehbarkeit