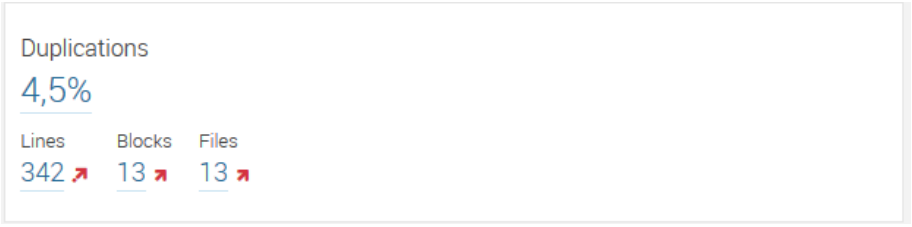
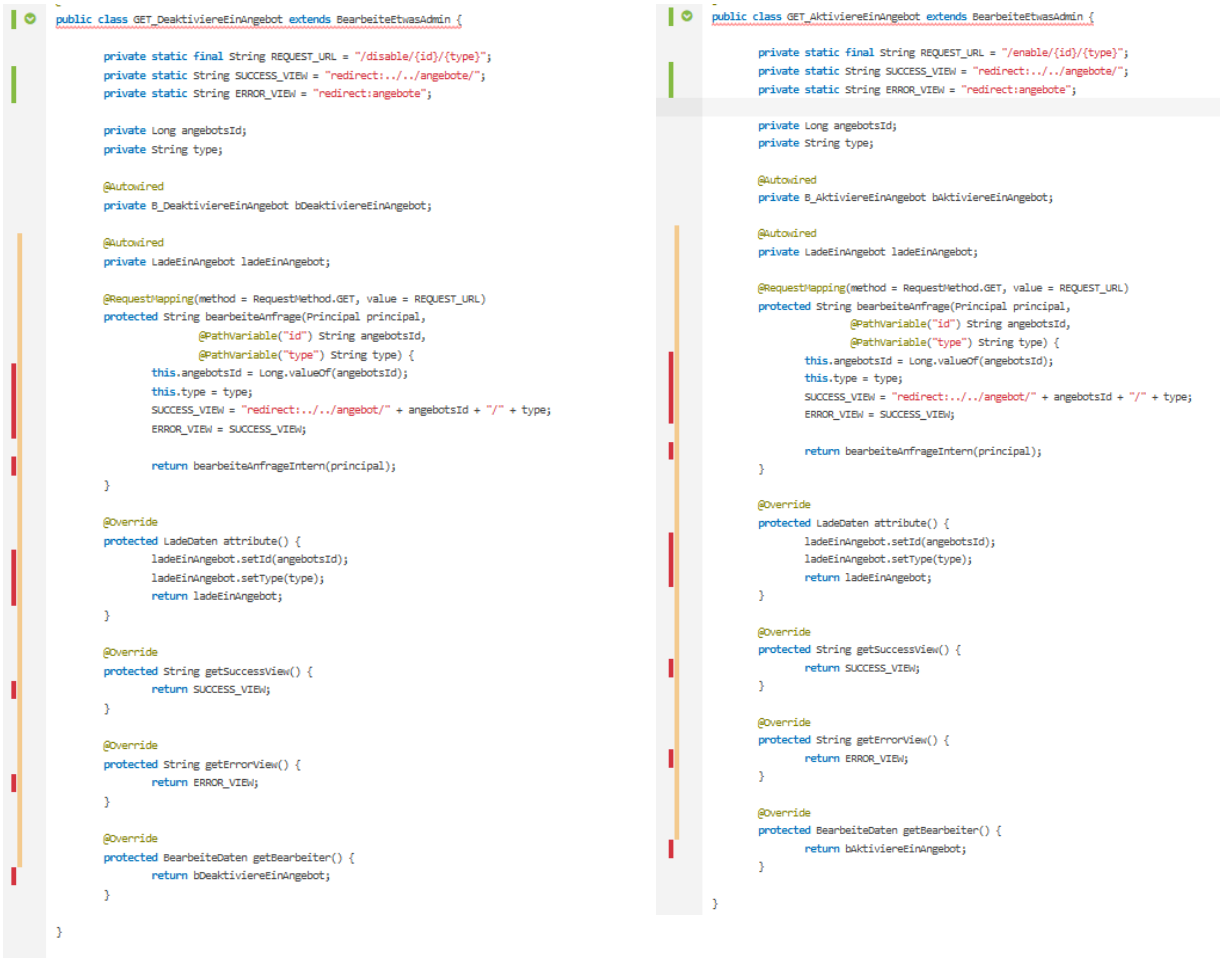
Metrics

Duplications (Duplizierter Code) & Complexity (Cyclomatic Complexity) wurden als Metrics ausgewählt.

Duplizierter Code



Durch die orangefarbene Markierung in der unteren Abbildung wird leicht ersichtlich, dass die Klassen sich nur sehr wenig voneinander unterscheiden. Der duplizierte Code kann durch eine gemeinsame Oberklasse reduziert werden.



```
public class GET_DeaktiviereEinAngebot extends BearbeiteEtwasAdmin {

        private static final String REQUEST_URL = "/disable/{id}/{type}";
        private static String SUCCESS_VIEW = "redirect:../../angebote/";
        private static String ERROR_VIEW = "redirect:angebote";

        private Long angebotsId;
        private String type;

        @Autowired
        private B_DeaktiviereEinAngebot bDeaktiviereEinAngebot;

        @Autowired
        private LadeEinAngebot ladeEinAngebot;

        @RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
        protected String bearbeiteAnfrage(Principal principal,
                        @PathVariable("id") String angebotsId,
                        @PathVariable("type") String type) {
                this.angebotsId = Long.valueOf(angebotsId);
                this.type = type;
                SUCCESS_VIEW = "redirect:../../angebot/" + angebotsId + "/" + type;
                ERROR_VIEW = SUCCESS_VIEW;

                return bearbeiteAnfrageIntern(principal);
        }

        @Override
        protected LadeDaten attribute() {
                ladeEinAngebot.setId(angebotsId);
                ladeEinAngebot.setType(type);
                return ladeEinAngebot;
        }

        @Override
        protected String getSuccessView() {
                return SUCCESS_VIEW;
        }

        @Override
        protected String getErrorView() {
                return ERROR_VIEW;
        }

        @Override
        protected BearbeiteDaten getBearbeiter() {
                return bDeaktiviereEinAngebot;
        }

}
```

```
public class GET_AktiviereEinAngebot extends BearbeiteEtwasAdmin {

        private static final String REQUEST_URL = "/enable/{id}/{type}";
        private static String SUCCESS_VIEW = "redirect:../../angebote/";
        private static String ERROR_VIEW = "redirect:angebote";

        private Long angebotsId;
        private String type;

        @Autowired
        private B_AktiviereEinAngebot bAktiviereEinAngebot;

        @Autowired
        private LadeEinAngebot ladeEinAngebot;

        @RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
        protected String bearbeiteAnfrage(Principal principal,
                        @PathVariable("id") String angebotsId,
                        @PathVariable("type") String type) {
                this.angebotsId = Long.valueOf(angebotsId);
                this.type = type;
                SUCCESS_VIEW = "redirect:../../angebot/" + angebotsId + "/" + type;
                ERROR_VIEW = SUCCESS_VIEW;

                return bearbeiteAnfrageIntern(principal);
        }

        @Override
        protected LadeDaten attribute() {
                ladeEinAngebot.setId(angebotsId);
                ladeEinAngebot.setType(type);
                return ladeEinAngebot;
        }

        @Override
        protected String getSuccessView() {
                return SUCCESS_VIEW;
        }

        @Override
        protected String getErrorView() {
                return ERROR_VIEW;
        }

        @Override
        protected BearbeiteDaten getBearbeiter() {
                return bAktiviereEinAngebot;
        }

}
```

Nach Bildung der Oberklasse und somit Reduzierung des duplizierten Codes sehen die Klassen folgendermaßen aus:

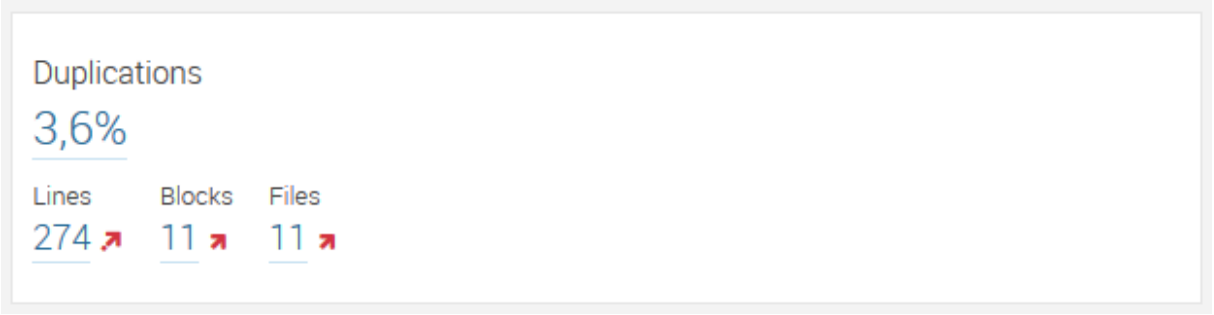
```java
@Controller
public class GET_DeaktiviereEinAngebot extends AktivierenDeaktiveren {

	private static final String REQUEST_URL = "/disable/{id}/{type}";

	@Autowired
	private B_DeaktiviereEinAngebot bDeaktiviereEinAngebot;

	@RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
	protected String bearbeiteAnfrage(Principal principal,
			@PathVariable("id") String angebotsId,
			@PathVariable("type") String type) {

		return bearbeiteAnfrageIntern(principal, angebotsId, type);
	}

	@Override
	protected BearbeiteDaten getBearbeiter() {
		return bDeaktiviereEinAngebot;
	}

}
```

```java
@Controller
public class GET_AktiviereEinAngebot extends AktivierenDeaktiveren {

	private static final String REQUEST_URL = "/enable/{id}/{type}";

	@Autowired
	private B_AktiviereEinAngebot bAktiviereEinAngebot;

	@RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
	protected String bearbeiteAnfrage(Principal principal,
			@PathVariable("id") String angebotsId,
			@PathVariable("type") String type) {
		return bearbeiteAnfrageIntern(principal, angebotsId, type);
	}

	@Override
	protected BearbeiteDaten getBearbeiter() {
		return bAktiviereEinAngebot;
	}

}
```

Die entstandene Oberklasse:

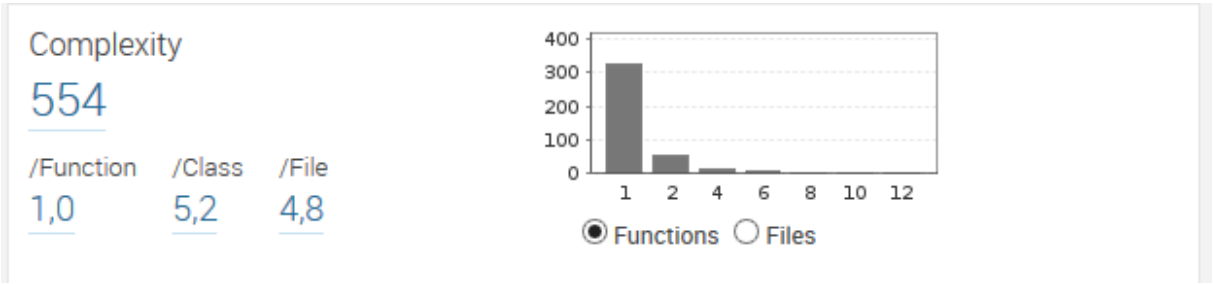
```
public abstract class AktivierenDeaktiveren extends BearbeiteEtwasAdmin {

	protected static String SUCCESS_VIEW = "redirect:../../angebote/";
	protected static String ERROR_VIEW = "redirect:angebote";

	protected Long angebotsId;
	protected String type;

	@Autowired
	private LadeEinAngebot ladeEinAngebot;

	protected String bearbeiteAnfrageIntern(Principal principal,
			String angebotsId, String type) {
		this.angebotsId = Long.valueOf(angebotsId);
		this.type = type;
		SUCCESS_VIEW = "redirect:../../angebot/" + angebotsId + "/" + type;

		return bearbeiteAnfrageIntern(principal);
	}

	@Override
	protected LadeDaten attribute() {
		ladeEinAngebot.setId(angebotsId);
		ladeEinAngebot.setType(type);
		return ladeEinAngebot;
	}

	@Override
	protected String getSuccessView() {
		return SUCCESS_VIEW;
	}

	@Override
	protected String getErrorView() {
		return ERROR_VIEW;
	}

}
```

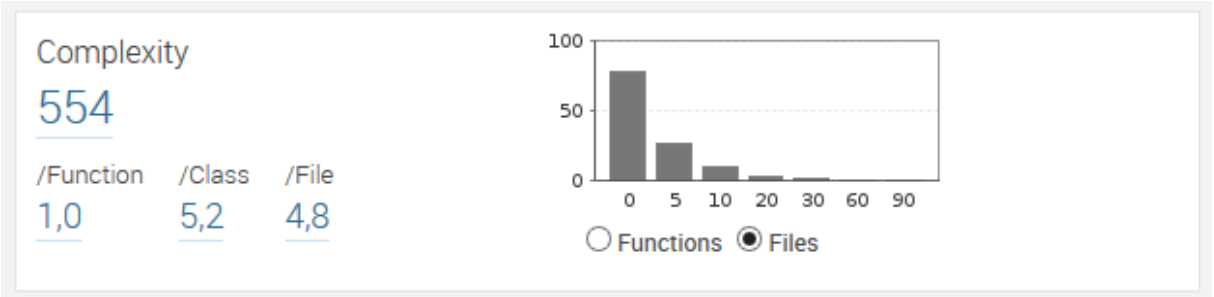
Dadurch verringert sich der Wert der Duplications.



Cyclomatic Complexity

Die Cyclomatic Complexity beschreibt die Komplexität eines Moduls, also beispielsweise einer Funktion oder einer Klasse. Dabei wird die Anzahl der linear unabhängigen Pfade eines Moduls berechnet. Diese Anzahl gibt damit also auch die maximale Anzahl der Testfälle an um einer vollständige Testabdeckung zu erreichen.





Durch obenstehende Abbildung der Complexity in Files konnte die Klasse *BenutzerServiceImpl* mit einer Complexity von über 30 identifiziert werden. Dies wurde folgendermaßen behoben.

Ursprünglicher Code:

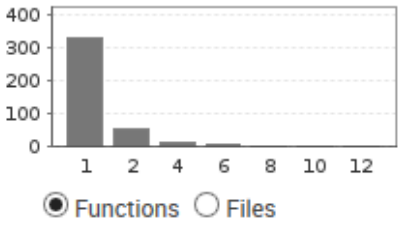
```
@Override
public Benutzer findByAngebotsIdAndType(Long id, String type) {
        Benutzer benutzer = null;
        switch (type) {
        case "ausleihen":
                benutzer = ausleihartikelDao.findById(id).getBenutzer();
                break;

        case "tauschen":
                benutzer = tauschartikelDao.findById(id).getBenutzer();
                break;

        case "helfen":
                benutzer = hilfeleistungDao.findById(id).getBenutzer();
                break;
        }

        return benutzer;
}
```

```java
@Autowired
private Map<String, AngebotsDAO<?>> angebotDAOs;

public BenutzerServiceImpl() {
        angebotDAOs = new HashMap<String, AngebotsDAO<?>>();
        angebotDAOs.put("ausleihen", ausleihartikelDao);
        angebotDAOs.put("tauschen", tauschartikelDao);
        angebotDAOs.put("helfen", hilfeleistungDao);
}

@Override
public Benutzer findById(long id) {
        return benutzerDao.findById(id);
}

@Override
public Benutzer findByAngebotsIdAndType(Long id, String type) {
        return angebotDAOs.get(type).findById(id).getBenutzer();
}
```

Complexity

545

/Function	/Class	/File
1,1	5,0	4,6



Duplications

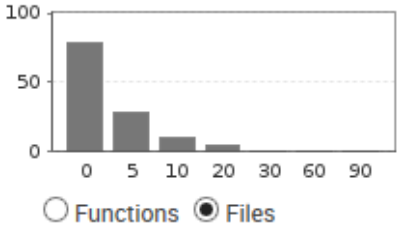
2,6%

Lines	Blocks	Files
196	8	8

Complexity

545

/Function	/Class	/File
1,1	5,0	4,6



Wie zu erkennen konnte die Complexity gesenkt werden.