

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from the bar, containing the date.

24-6-2015

Review Software

PTS6

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Mike Rooijackers, Noor van Oekel, Jordi Knol,
Maaïke Jansen, Tim Hermens
GROUP E

Inhoudsopgave

INLEIDING	2
DOMEIN.....	3
MEETRESULTATEN.....	3
VISUELE INSPECTIE	4
CONCLUSIE	4
JMSLAYERMODULEEE	6
MEETRESULTATEN.....	6
VISUELE INSPECTIE	6
CONCLUSIE	7
MOCJMS	8
MEETRESULTATEN.....	8
VISUELE INSPECTIE	8
CONCLUSIE	9
SERVICES MODULE	10
MEETRESULTATEN.....	10
VISUELE INSPECTIE	10
CONCLUSIE	11
WORKSPACE MANAGEMENT MODULE.....	12
MEETRESULTATEN.....	12
VISUELE INSPECTIE	13
CONCLUSIE	14
JENKINS	15
GLOBALE CONCLUSIE	16

Inleiding

De code van het project wordt door Jenkins en Sonar gecontroleerd op fouten. Zo geeft Jenkins aan of het project gebuild kan worden en geeft Sonar de kwaliteit van de code weer. Als er issues zijn voor verbeteringen zal Sonar deze weergeven.

Tijdens de gehele periode heeft sonar de voortgang bijgehouden van de kwaliteit van de code. Zo heeft elke module binnen het project een eigen Jenkins en Sonar check. Dit is gedaan om een duidelijker overzicht te geven van fouten in de verschillende delen van de code.

Voor de code review is voornamelijk gekeken naar de grotere modules:

- Domein
- Services Module
- Workspace Management Module

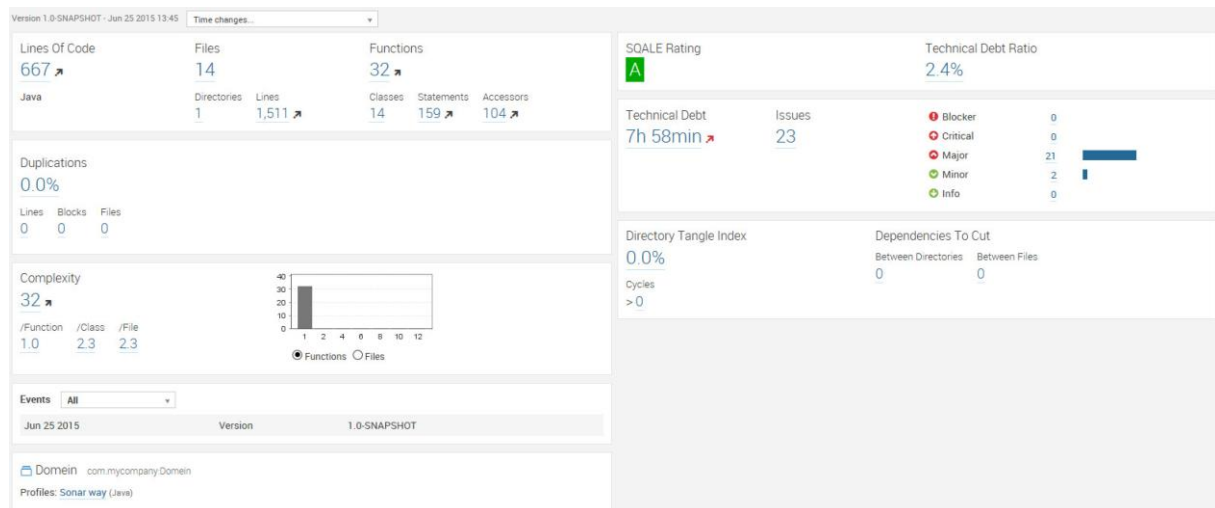
Daarnaast zijn de modules met betrekking tot JMS (JMSLayerModuleEE en MoCJMS) nog geanalyseerd om de voornaamste problemen in kaart te brengen.

Per module is bepaald wat de kwaliteit van de code is en hoe deze eventueel verbeterd zou kunnen worden. Daarnaast is een visuele inspectie uitgevoerd op delen van de code.

Domein

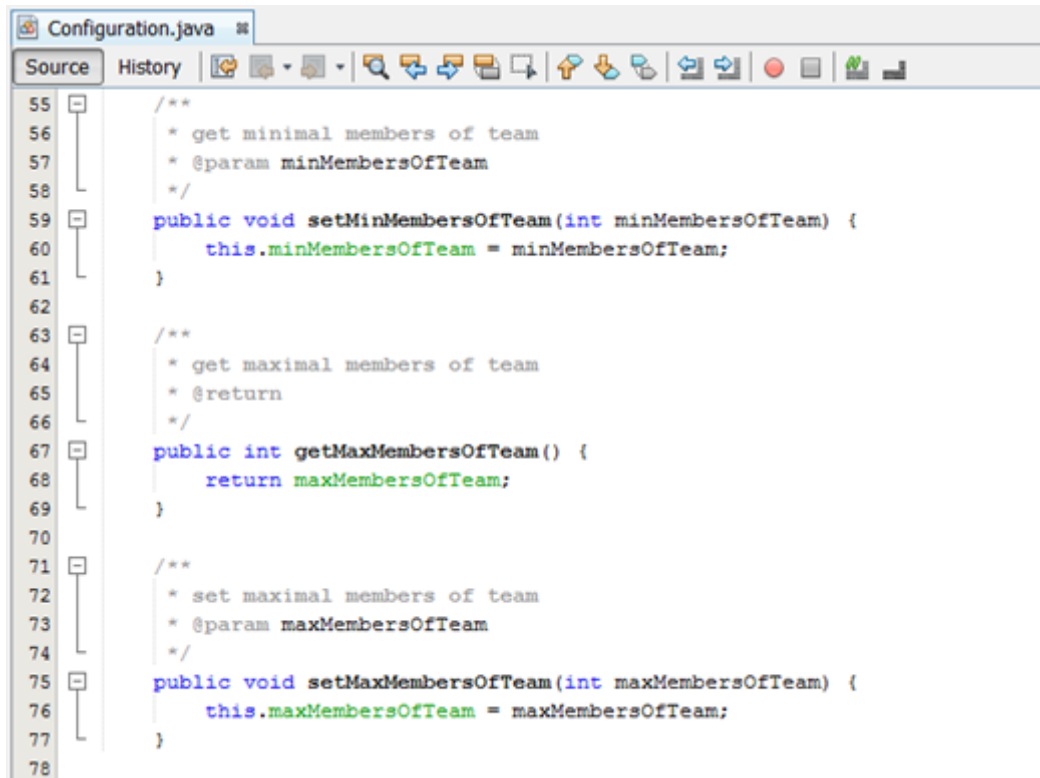
Meetresultaten

De analyse van Sonar levert de volgende resultaten op:



Onderdeel	Hoeveelheid
Aantal Java files	14
Aantal fysieke regels code	667
Aantal klassen	14
Aantal methodes	32
Aantal statements	159
Gemiddelde cyclomatische complexiteit	32
Gemiddelde diepte van een block	1

Visuele inspectie



Het bovenstaande screenshot is een gedeelte van de code uit de Configuration klasse van de Domein module. Binnen dit codefragment zijn een aantal getters en setters voor het minimale en maximale aantal spelers per team te zien.

De Domein module bevat voornamelijk getters en setters voor de verschillende objecttypen. Deze getters en setters zijn voorzien van Javadoc, maar deze kunnen in een aantal gevallen nog worden aangevuld (bijvoorbeeld de @return in de Javadoc voor getMaxMembersOfTeam).

De complexiteit van de code is laag, waardoor de code eenvoudig uit te breiden is.

In Sonar worden de volgende problemen aangegeven:

- Package namen moeten voldoen aan naming conventions.
- Veldnamen moeten voldoen aan naming conventions.
- Lokale variabelen en methodeparameters moeten voldoen aan naming conventions.
- Velden in een "Serializable" klasse moeten transient of serializable zijn.
- Methodes moeten niet te veel parameters hebben.
- Ongebruikte methodeparameters moeten verwijderd worden.

Conclusie

De code van deze module krijgt klasse A toegekend in Sonar. De technical debt is laag (2,4%). Het kost 7 uur en 58 minuten om alle problemen op te lossen. Er zijn geen kritieke problemen binnen de code.

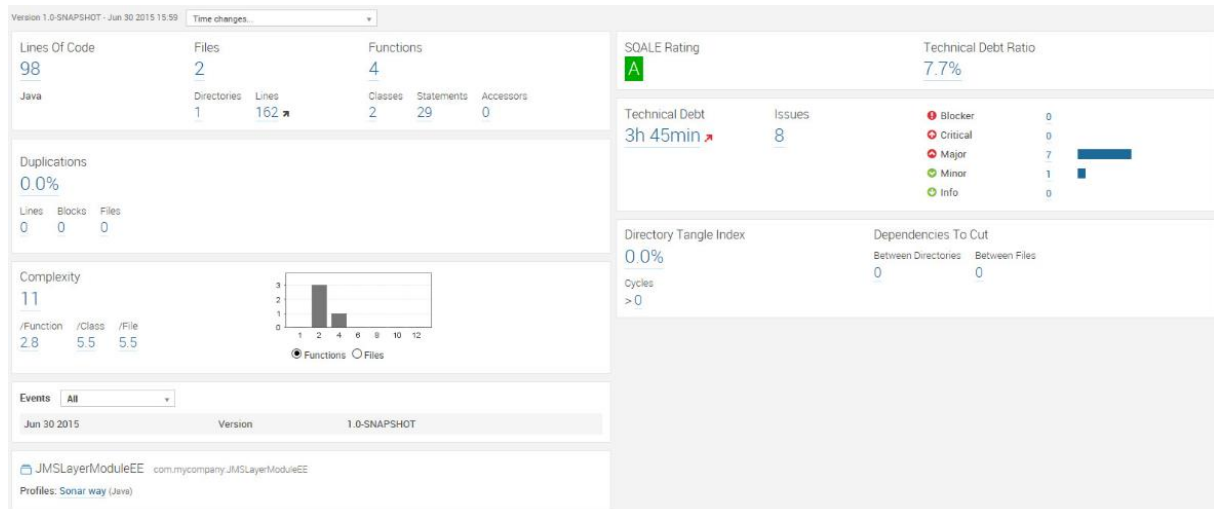
De cyclomatische complexiteit van deze module is wat aan de hoge kant, maar komt dicht in de buurt van de gewenste score van 10 tot 15, tevens is de diepte van een blok weer laag. Het zal voor andere projectteams eenvoudig zijn om de Domein module uit te breiden met nieuwe functionaliteiten en de module te onderhouden.

Softwarekwaliteit kan verhoogd worden door de problemen die in Sonar worden aangegeven op te lossen.

JMSLayerModuleEE

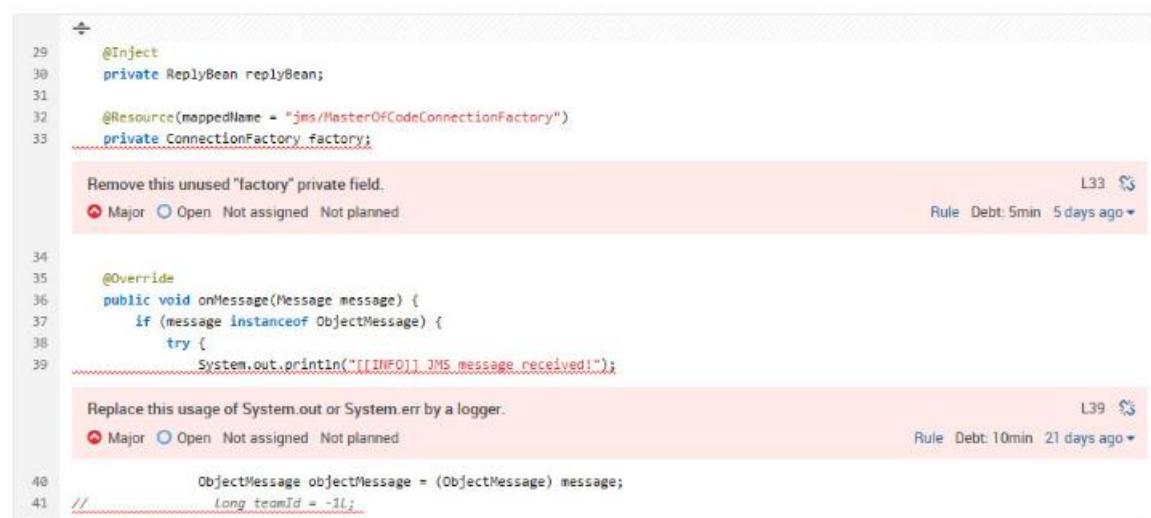
Meetresultaten

De analyse van Sonar levert de volgende resultaten op:



Onderdeel	Hoeveelheid
Aantal Java files	2
Aantal fysieke regels code	98
Aantal klassen	2
Aantal methodes	4
Aantal statements	29
Gemiddelde cyclomatische complexiteit	11
Gemiddelde diepte van een block	2,9

Visuele inspectie



Het fragment in het bovenstaande screenshot is afkomstig uit de RequestBean. In Sonar zijn de volgende problemen bij de code aangegeven:

- System.out en System.err moeten niet als loggers gebruikt worden.
- Vermijd uitgecommentarieerde code.
- Ongebruikte private velden moeten verwijderd worden.

Conclusie

De code van de JavaLayerModuleEE module krijgt klasse A toegekend in Sonar. De technical debt is 7,7%, dus het is niet echt hoog. Het kost 3 uur en 45 minuten om alle problemen op te lossen. Er zijn geen kritieke problemen binnen de code.

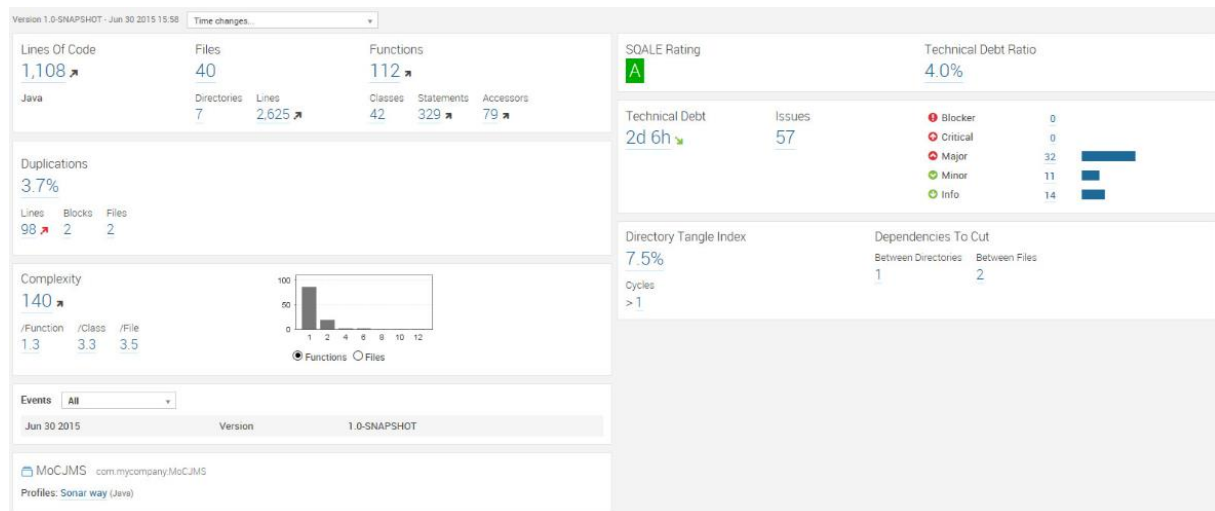
Zoals beschreven in de visuele inspectie is de complexiteit van de code zeer laag. Het valt binnen de marge van 10 tot 15 en zal dus prima uitbreidbaar zijn voor andere projectteams.

Ter verbetering kunnen de problemen die in Sonar worden aangegeven worden opgelost. Tevens kunnen er unittesten worden toegevoegd om het versturen van berichten te testen.

MoCJMS

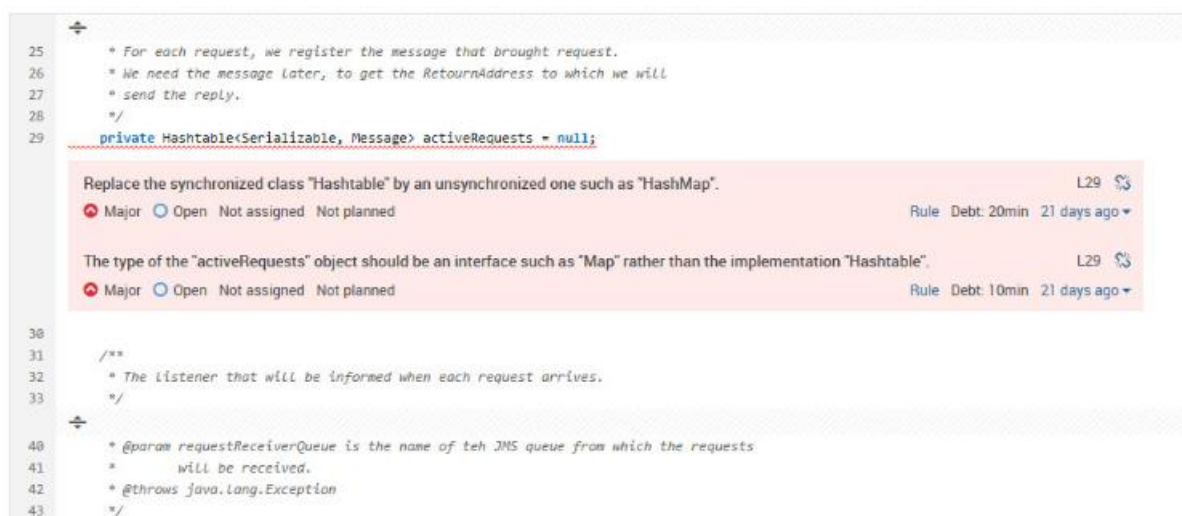
Meetresultaten

De analyse van Sonar levert de volgende resultaten op:



Onderdeel	Hoeveelheid
Aantal Java files	40
Aantal fysieke regels code	1108
Aantal klassen	42
Aantal methodes	112
Aantal statements	329
Gemiddelde cyclomatische complexiteit	140
Gemiddelde diepte van een block	1,3

Visuele inspectie



Het bovenstaande screenshot is een codefragment uit de `AsynchronousReplier` class. In Sonar zijn de volgende problemen bij de code aangegeven:

- Velden in een “Serializable” klasse moeten transient of serializable zijn.
 - Vermijd uitcommentarierde code.
 - Verouderde elementen moet voorzien zijn van een annotatie en Javadoc.
 - De excepties `Error`, `RuntimeException`, `Throwable` en `Exception` zouden niet opgegooid moeten worden.
 - Gebruik Java collection interfaces zoals “List” in plaats van specifieke implementatieklassen zoals “LinkedList”.
- Fout:** `LinkedList<Object> myList = new LinkedList<>();`
- Goed:** `List<Object> myList = new LinkedList<>();`
- Onnodige “{” en “}” moeten verwijderd worden om misverstanden te voorkomen.

Conclusie

De code van de MoCJMS module heeft een SQALE rating van A, wat betekent dat het van goede kwaliteit is. Het is goed onderhoudbaar, maar de complexiteit is wel hoog. De diepte van een block is juist weer laag.

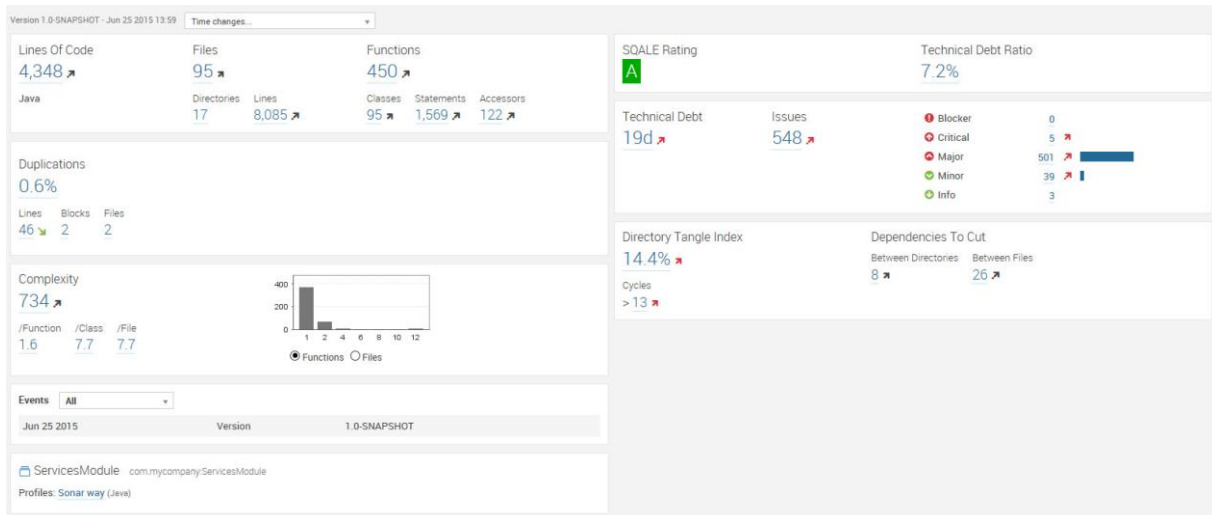
De technical debt van deze module is opnieuw laag (4,0%). Het zou in totaal 2 dagen en 6 uur duren om de aanwezige problemen op te lossen.

Om de code te verbeteren, dienen de bovenstaande problemen opgelost te worden. Daarnaast kunnen er nog unittesten toegevoegd worden om de `CommandRunner` klasse te testen.

Services Module

Meetresultaten

De analyse van Sonar levert de volgende resultaten op:



Onderdeel	Hoeveelheid
Aantal Java files	95
Aantal fysieke regels code	4348
Aantal klassen	95
Aantal methodes	450
Aantal statements	1569
Gemiddelde cyclomatische complexiteit	734
Gemiddelde diepte van een block	1,6

Visuele inspectie

```
RestResource.java
Source History
112     long userId = message.getUserId();
113     long teamId = message.getTeamId();
114     return userService.addToTeam(userId, teamId);
115 }
116
117 @POST
118 @Path("createteam")
119 @Produces({MediaType.APPLICATION_JSON})
120 public Team CreateTeam(CreateTeamMessage message) {
121     String teamName = message.getTeamName();
122     String initiator = message.getInitiator();
123     List<String> members = message.getMembers();
124     long competitionId = communicationBean.getCompetitionDataService().getCurrentCompetition().getId();
125
126     Team team = userService.createTeam(teamName, initiator, members, competitionId);
127
128     communicationBean.sendMessageToWorkspaceManagementBean(new CreateWorkspaceRequestMessage(team.getId(), competitionId));
129     for (String address : members) {
130         if (!address.equals("")) {
131             mailMessenger.sendAddedToTeamMessage(address, members, team.getTeamName(), initiator);
132         }
133     }
134     team.setInitiator(null);
135     team.setMembers(userService.getTeamMembers(team));
136     return team;
137 }
```

Het bovenstaande codefragment is de code voor het aanmaken van een team. Het is afkomstig uit de `RestResource` klasse.

De waarschuwing op regel 120 geeft aan dat de methode `CreateTeam` omgezet kan worden naar een asynchrone methode. Het voordeel hiervan is dat er meerdere teams tegelijkertijd aangemaakt zouden kunnen worden.

De diepte van een block is binnen de `CreateTeam` methode ook terug te zien; er staat een `if`-statement binnen een `for`-loop (regels 129 t/m 133). Het is echter niet ernstig, want de diepte is hier zeer beperkt.

Data wordt vóór de aanroep van een methode in variabelen opgeslagen, bijvoorbeeld de naam van een team en de naam van de initiator. Het voordeel hiervan is dat de code overzichtelijker wordt en er, indien nodig, nog checks kunnen worden uitgevoerd op deze variabelen om te controleren of ze geldige waarden bevatten.

Sonar geeft de volgende problemen aan:

- Switch cases moeten eindigen met een `break` statement zonder condities.
- Inloggegevens zouden niet hard-coded moeten zijn.
- Constante namen moeten voldoen aan naming conventions.
- Package namen moeten voldoen aan naming conventions.
- Methodenamen moeten voldoen aan naming conventions.
- `System.out` en `System.err` moeten niet als loggers gebruikt worden.
- Vermijd uitcommentarierde code.
- Ongebruikte lokale variabelen moeten verwijderd worden.

Conclusie

De code van de `Services Module` is van goede kwaliteit en krijgt binnen Sonar klasse A toegekend. De technical debt is hoger dan bij de `Workspace Management Module`, maar ligt nog steeds onder de 10%. Het zou 19 dagen kosten om alle problemen binnen deze module op te lossen. Er zijn echter geen blockers aanwezig die voor oponthoud kunnen zorgen.

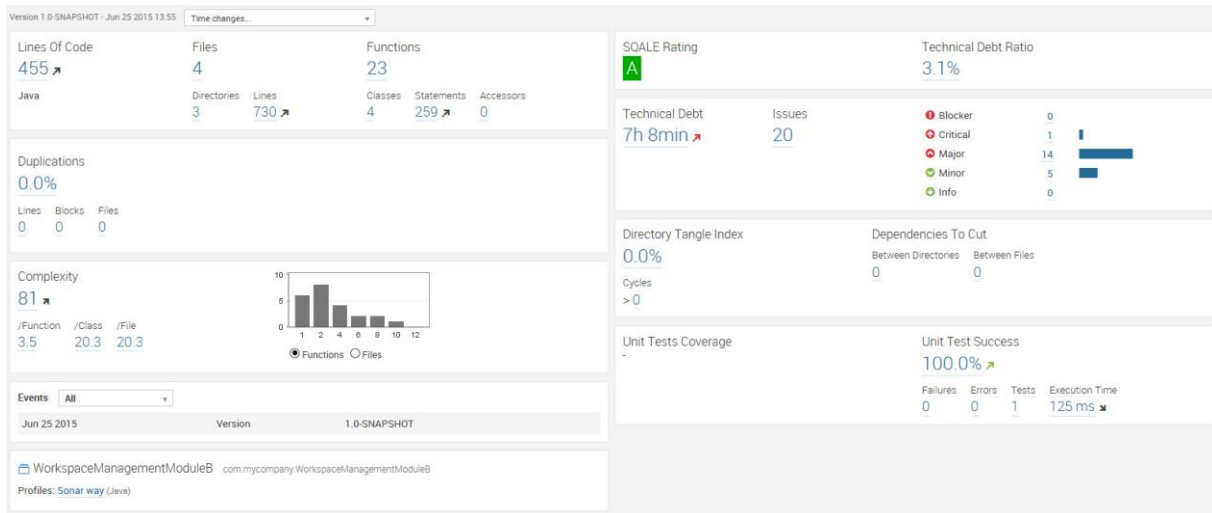
Om de code coverage te verbeteren, kunnen aan deze module nog unittesten toegevoegd worden. Deze zouden bijvoorbeeld de functionaliteiten binnen de `CommunicationBean` kunnen controleren. Verder zijn er nog een aantal verbeteringen mogelijk op het gebied van complexiteit en commentaar. Aan de hand van de problemen die door Sonar worden aangegeven, kunnen deze verbeteringen eenvoudiger worden doorgevoerd.

Ook bij deze module is de cyclomatische complexiteit veel te hoog, maar wederom is de diepte van een blok erg laag.

Workspace Management Module

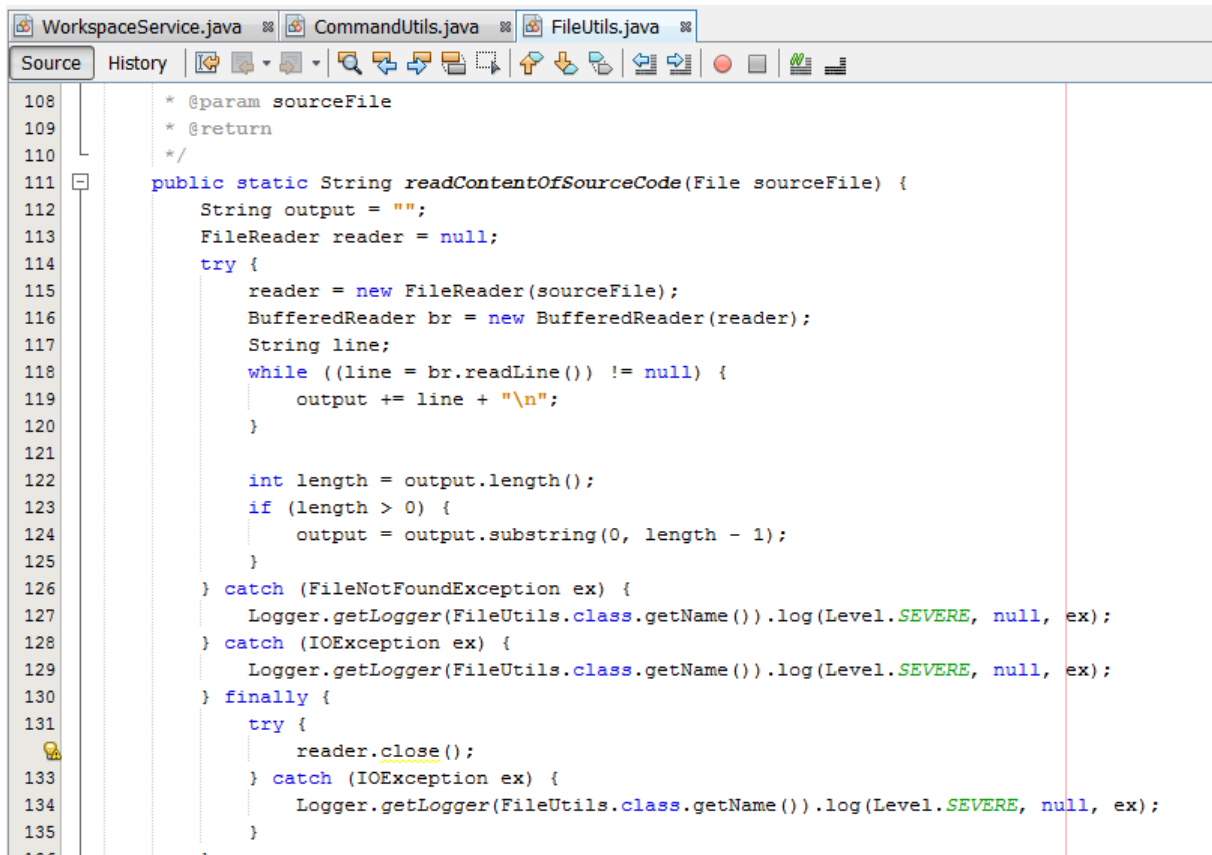
Meetresultaten

De analyse van Sonar levert de volgende resultaten op:



Onderdeel	Hoeveelheid
Aantal Java files	4
Aantal fysieke regels code	455
Aantal klassen	4
Aantal methodes	23
Aantal statements	259
Gemiddelde cyclomatische complexiteit	81
Gemiddelde diepte van een block	3,5

Visuele inspectie



```
108      * @param sourceFile
109      * @return
110      */
111      public static String readContentOfSourceCode(File sourceFile) {
112          String output = "";
113          FileReader reader = null;
114          try {
115              reader = new FileReader(sourceFile);
116              BufferedReader br = new BufferedReader(reader);
117              String line;
118              while ((line = br.readLine()) != null) {
119                  output += line + "\n";
120              }
121
122              int length = output.length();
123              if (length > 0) {
124                  output = output.substring(0, length - 1);
125              }
126          } catch (FileNotFoundException ex) {
127              Logger.getLogger(FileUtils.class.getName()).log(Level.SEVERE, null, ex);
128          } catch (IOException ex) {
129              Logger.getLogger(FileUtils.class.getName()).log(Level.SEVERE, null, ex);
130          } finally {
131              try {
132                  reader.close();
133              } catch (IOException ex) {
134                  Logger.getLogger(FileUtils.class.getName()).log(Level.SEVERE, null, ex);
135              }
136          }
```

Het bovenstaande screenshot is een deel van de code uit de klasse FileUtils. Dit is de code voor de methode waarmee broncode kan worden uitgelezen.

Wat opvalt aan de code is dat er een try-catch statement wordt gebruikt met meerdere catch statement voor verschillende excepties. De verschillende statements kunnen vervangen worden door één multi-catch, aangezien alle catch statements de logger aanroepen.

De waarschuwing op regel 132 luidt “dereferencing possible null pointer”. De reden hiervoor is dat de variabele reader pas binnen de try statement een waarde toegekend krijgt en van tevoren “null” is. Hoe dit precies opgelost kan worden zonder andere problemen te veroorzaken is niet duidelijk, omdat er wel gecontroleerd moet worden of de opgegeven sourceFile een geldig bestand is.

Sonar geeft de volgende problemen aan:

- Throwable.printStackTrace() mag nooit aangeroepen worden.
- Vermijd uitcommentarierde code.
- Exception handler moeten de originele exceptie behouden.
- System.out en System.err moeten niet als loggers gebruikt worden.
- Utility klassen zouden geen publieke constructor moeten hebben.
- Methodes moeten niet te complex zijn.
- String literals zouden aan de linkerkant geplaatst moeten worden als er gekeken wordt of ze gelijk zijn.

Conclusie

De code van de Workspace Management Module heeft volgens Sonar klasse A, wat betekent dat de code van goede kwaliteit is. Er zijn een aantal problemen binnen de code, maar de technical debt om deze problemen op te lossen is relatief laag. Het zou 7 uur en 8 minuten in beslag nemen om deze problemen te verhelpen.

De unittesten voor deze module slagen. Er is echter tot nu toe een klein aantal unittesten om de code te controleren, dus het projectteam is bezig om extra unittesten toe te voegen om de code coverage te verbeteren.

De cyclomatische complexiteit van de module is 81. Het is aan te raden om de complexiteit niet groter te maken dan 10 tot 15. Hieruit kan geconcludeerd worden dat de cyclomatische complexiteit van dit programma veel te hoog is. Dit is dan ook een van de punten die aangepakt zouden kunnen worden in dit programma. Het is mogelijk dat de complexiteit lager wordt wanneer de problemen die Sonar aangeeft worden opgelost.

De diepte van een blok is relatief laag.

Jenkins

Naast de resultaten van Sonar, zijn ook de resultaten van de builds in Jenkins meegenomen in de beoordeling van de softwarekwaliteit.



S	W	Name	Last Success	Last Failure	Last Duration
		Annotations	1 hr 24 min - #54	1 day 1 hr - #46	43 sec
		Assignment	53 min - #144	3 hr 8 min - #140	42 sec
		Domein	1 hr 2 min - #101	8 days 5 hr - #64	38 sec
		JMSLayerModuleEE	48 min - #210	1 day 2 hr - #193	1 min 7 sec
		MoCJMS	49 min - #205	7 days 4 hr - #101	36 sec
		ServicesModule	48 min - #252	3 hr 25 min - #241	1 min 18 sec
		UtilitiesModule	58 min - #53	22 hr - #47	1 min 4 sec
		WorkspaceManagementModuleB	52 min - #230	3 hr 21 min - #221	40 sec

Icon: [S](#) [M](#) [L](#) [add description](#) [Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Het bovenstaande screenshot toont alle builds van het project binnen Jenkins. Voor iedere module binnen het project is een aparte build te zien. Uit het overzicht blijkt dat alle builds slagen.

Voor bepaalde modules duurt het langer om een build uit te voeren, maar de totale tijd die nodig is ligt bij alle modules onder de twee minuten. In de onderstaande tabel is de volgorde op basis van de benodigde tijd weergegeven:

Module	Tijd (min)
ServicesModule	1:18
JMSLayerModuleEE	1:07
UtilitiesModule	1:04
Annotations	0:43
Assignment	0:43
WorkspaceManagementModule	0:40
Domein	0:38
MoCJMS	0:36

Voor het bouwen van de ServicesModule is de meeste tijd nodig. Dit is niet verrassend, omdat de ServicesModule tevens een van de modules is met het grootste aantal klassen en methodes. Bij de Domein module is de benodigde tijd veel minder, omdat die module vooral uit getters en setters bestaat.

Globale conclusie

Elk project dat in Sonar voorkomt heeft de klasse A. Dit wil aangeven dat de code prima onderhoudbaar is.

Er zijn nog een aantal problemen bij de grotere modules (Workspace Management Module en Services Module) van “naming conventions” die niet helemaal goed zijn toegepast binnen de code. Een aantal problemen zijn tijdens de ontwikkeling van de software al opgelost, maar ze zijn nog niet allemaal verholpen. Verder zijn de meeste problemen relatief klein en eenvoudig op te lossen.

Ook is de cyclomatische complexiteit te hoog, dit zou bij het verder ontwikkelen van de modules aangepakt moeten worden.

De grootste uitdaging voor de uitbreiding van de code zal liggen bij de Services Module, omdat dit de meest uitgebreide module is binnen het project.