Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Rodriguez Penin, Hugo; Valveny Llobet, Ernest, dir. Avaluació de la qualitat del codi en exercicis de programació. 2023. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/280716

under the terms of the  license

Avaluació de la qualitat del codi en exercicis de programació

Hugo Rodríguez Penin

Resum— A l'hora d'aprendre programació és molt important el feedback que l'estudiant rep dels treballs fets durant una assignatura per tal de millorar les seves capacitats. Actualment, a la Universitat Autònoma de Barcelona no es disposa de cap eina que ens permeti avaluar els aspectes referents a la qualitat del codi, com pot ser l'estil de codificació seguint una guia d'estil. Aquest projecte consisteix en l'elaboració d'una eina que permeti avaluar la qualitat del codi de diversos exercicis, lliurant llistes ordenades dels alumnes amb una valoració final de la qualitat de codi de cada un, i les mètriques que passen o no el llindar mínim definit per cada mètrica.

Paraules clau— Qualitat, codi, programació, estil, metodologia, cohesió. acoplament, mètrica, deute, tècnic.

Abstract— When it comes to learning programming, the feedback that students receive on their assignments during a course is crucial to improve their skills. Currently, at the Autonomous University of Barcelona, there is no tool available to assess aspects related to code quality, such as coding style following a style guide. This project aims to develop a tool that can evaluate the code quality of various exercises, providing ranked lists of students with a final assessment of the code quality for each student, along with the metrics that meet or fail to meet the defined minimum threshold for each metric.

Index Terms— Quality, code, programming, style, methodology, cohesion, coupling, metric, technical debt.

1 Introducció

A l'hora d'aprendre programació és molt important el feedback que l'estudiant rep dels treballs fets durant una assignatura per tal de millorar les seves capacitats i assegurar-se d'estar aprenent a programar correctament.

Per això existeixen moltes eines que ens permeten avaluar el resultat d'un programa, per tal de comprovar si el seu funcionament del codi és l'esperat i dona les sortides desitjades en funció de les dades d'entrada del programa.

Això no obstant, actualment a la Universitat Autònoma de Barcelona no es disposa de cap eina que ens permeti avaluar els aspectes referents a la qualitat del codi, com pot ser l'estil de codificació seguint les normes d'estil del llenguatge de programació que s'estigui programant, la cohesió o l'acoblament entre d'altres i fins i tot si el codi del programa és realment eficient o si és realment òptim, entre altres factors. Això és a causa del fet que són aspectes més complicats d'avaluar correctament, ja que habitualment no hi ha una resposta correcta sinó, que més bé és una mesura relativa de la que ens podem fer una idea aplicant distintes mètriques que existeixen per tal d'assegurar que el nostre codi té la qualitat més gran possible.

L'objectiu d'aquest projecte consisteix en una primera cerca de les distintes mètriques existents que serveixen per avaluar la qualitat d'un codi i eines o mètodes que existeixen per aplicar aquestes mètriques i utilitzar aquesta informació per donar suport personalitzat als alumnes que ho requereixin. A més d'aplicar mètriques ja existents es vol obtenir dades sobre la qualitat que presenta l'alumne en funció de la similitud que presenta amb la solució proposada pel professorat.

També s'ha de permetre generar informes de seguiment del treball de cada estudiant i del grup complet, per tal de poder visualitzar fàcilment quins són els codis que presenten menor qualitat de codi respecte a les mètriques que escollides, donant informació sobre les parts del codi que suposen una problemàtica, en cas de ser necessari.

- *E-mail de contacte: Hugo.RodriguezP@autonoma.cat*
- *Menció realitzada: Enginyeria del Software*
- *Treball tutoritzat per: Ernest Valveny Llobet (departament)*
- *Curs 2022/23*

En aquest document s'explicarà que són la qualitat del codi i el deute tècnic així com la seva importància. També es resumiran les principals mètriques i code smells que existeixen i les mètriques resultades d'una primera fase de cerca. Seguidament s'explicaran les possibles eines que s'han seleccionat per fer servir en el projecte.

2 ANÁLISIS DE LA QUALITAT DE CODI

2.1 Deute tècnic

Per a desenvolupar una aplicació de la millor manera possible és important prendre's el temps i els esforços necessaris per a aconseguir obtenir com a resultat un codi amb la qualitat més gran de codi possible. Amb la finalitat d'assolir el resultat desitjat és necessari mantenir atenció a tota mena de detalls, com pot ser la sintaxi, les estructures utilitzades i resultats desitjats entre altres factors de gran importància.[1]

No obstant això, l'obtenció d'un codi d'alta qualitat pot ser complexa i, sovint, tediosa tractant-se d'una tasca que pot portar hores o dies fins i tot a programadors experimentats. Això provoca que sovint es percebi com una pèrdua de temps i recursos a curt termini, causant que es cometi l'error de reduir l'avaluació del codi a mètriques tan simples com: "Funciona?, i dona el resultat desitjat?"[1]

Aquest tipus de pensament és un error comú i que pot resultar fatal per a qualsevol classe de projecte, especialment si es tracta de projectes grans i molt a llarg termini. Però per a entendre millor el problema d'aquest pensament i la importància de la qualitat del codi realitzat, hem d'entendre que és el deute tècnic.[1]

Deute tècnic es tracta d'un terme que va ser introduït per Ward Cunningham en 1992 i es pot definir com un deute intangible que sorgeix de la decisió de recórrer a solucions més senzilles o ràpides en lloc de la solució més efectiva. Aquest deute es mesura amb el cost del temps i recursos necessaris per a mantenir i arreglar un codi mal construït a causa de portar un mal control de qualitat, sovint, per buscar fer-ho ràpid. És important entendre que el deute tècnic no és necessàriament negatiu, a vegades es pot recórrer a ella per a poder complir amb els terminis de lliurament i invertir posteriorment el temps necessari per a arreglar els resultats d'aquesta decisió. No obstant això, és crític arribar a un equilibri entre complir amb les dates de lliurament i la qualitat del codi, ja que en abusar del deute tècnic es generen interessos que es tradueixen en un desenvolupament alentit, descens de la productivitat i un augment dels costos.[2][3]

Una vegada entès el que és el deute tècnic és possible entendre la importància de cuidar la qualitat del nostre codi amb l'objectiu de reduir el deute tècnic i evitar costos i problemes en un futur a causa d'una mala gestió d'aquesta, que pot portar a costos afegits elevats i imprevistos de temps.

2.2 Mètriques de qualitat de codi

2.2.1 ¿Què són mètriques de qualitat de codi?

Les mètriques de qualitat de codi són un conjunt de variables que es poden calcular a partir del nostre codi font i mitjançant les quals podem mesurar i determinar la qualitat d'aquest codi. És a dir, es tracta de les variables i fórmules que es poden tenir en compte per confirmar si el nostre codi presenta una bona qualitat de codi i, en cas negatiu, fer-les servir per revisar el codi a la recerca de canvis, cobertura de prova i altres coneixements pràctics per tal de solucionar els problemes presents. [4]

2.2.2 Complexitat ciclomàtica

És una puntuació que es dona a cada mètode i classe del codi i ens indica la complexitat estructural del codi. Les zones del codi amb major complexitat ciclomàtica és més probable que continguin problemes de flux o de seguretat i poden resultar més difícils de mantenir o de ser recorregut per tests. [5]

Per a calcular la complexitat ciclomàtica es mesura el nombre de fluxos diferents d'execució que pot tenir el codi d'un element de codi, és a dir, ens diu quants if else, while, for, switch etc, tenim en el nostre codi.[6]

Es crea un graf del flux de l'element que estem analitzant i, una vegada tenim desenvolupat aquest graf s'aplica una d'aquestes fórmules equivalents:[6]

V(G) = arestes - nodes + 2

V(G) = nodes predicat + 1

V(G) = nombre de regions.

2.2.3 Cohesió

La cohesió mesura la relació funcional entre els elements d'un mòdul. Per un mòdul d'alta cohesió si els seus mètodes estan relacionats entre si, tenen un contingut clar i una temàtica comuna i un abast ben definit i tenen poca relació amb altres elements d'altres mòduls. [5]

Per a calcular la cohesió existeixen diferents mètriques que ens permeten fer-nos una idea de la cohesió en les nostres classes:

Lack Of Cohesió in Methods **(LOCM):** Aquestes mètriques mesuren la falta de cohesió d'una determinada classe. Cal destacar LOCM4 que és una mètrica de Sonar que

calcula un valor per a una classe determinada, si el valor és major a 1 la classe és sospitosa de problemes de cohesió de manera que com més gran sigui el valor major falta de cohesió presenta la classe. LOCM4 mesura el nombre de relacions diferents que hi ha entre els mètodes de la classe sense tenir en compte alguns mètodes com a constructors o accessors. [7]

***Tigth Class Cohesion* (TCC):** Aquesta mètrica mesura la cohesió entre els mètodes públics d'una classe. És a dir, calcula el nombre relatiu de mètodes públics connectats directament en la classe. TCC mesura la relació entre el nombre real de mètodes visibles directament connectats en una classe dividit pel nombre màxim possible de connexions entre els mètodes públics d'una classe. [8]

***Loose class cohesion* (LCC):** És similar a TCC, però en aquest cas es calcula la densitat de connexions. [8]

A continuació es mostren les fórmules d'ambdues mètriques de cohesió per veure'n la diferencia de càlculs:

NP = nombre màxim de connexions possibles = N * (N − 1) / 2
NDC = nombre de connexions directes (nombre d'arestes en el graf de connexió)
NIC = nombre de connexions indirectes

TCC = NDC / NP
LCC = (NDC + NIC) / NP

2.2.4 Acoblament

L'acoblament és el grau en què els mòduls d'un programa depenen els uns dels altres. Com més gran és l'acoblament major dependència hi ha entre diferents mòduls del codi pel qual per a modificar un serà necessari modificar la resta. [5]

Per calcular l'acoblament existeix aquesta mètrica en concret:

Coupling between objects (CBO): És el nombre de classes a les quals una classe està lligada. Es dona dependència entre dues classes quan una classe usa mètodes o variables de l'altra classe. [9]

2.2.5 Líneas de codi

Aquesta mètrica consisteix a comptar el nombre de línies que conformen el codi. Els tipus més importants d'aquesta mètrica són els següents:

Línies de codi font: indica el nombre exacte de línies de codi font que són presents en l'arxiu de codi font, incloses les línies en blanc.

Línies de codi executable: indica el nombre aproximat de línies de codi executable o operacions. Es tracta d'un recompte del nombre d'operacions en el codi executable. [5]

2.2.6 Mètriques de codi duplicat

Aquestes mètriques permeten trobar blocs de codi duplicat presents en el projecte, que constitueixen un indicatiu de males pràctiques que poden ser evitades aplicant el principi Don't Repeat Yourself(DRY). [5]

2.2.7 Métriques de codi innecesari

Aquestes mètriques ens indiquen la quantitat de línies de codi que poden ser reduïdes immediatament, ja que es tracten de blocs de codi que no s'estan utilitzant o, d'altra banda, duplicades. [5]

2.2.8 Estandards de codificació

Es refereix a convencions per a escriure codi font, les quals sovint són dependents del llenguatge de programació. Les convencions més comunes fan referència a: noms de variables, indentacions, espaiat, etc. [5]

2.3 Code smells

2.3.1 Que son els code smells?

Els Code Smells són indicadors de mala qualitat del codi que habitualment indiquen que el codi no està net. Es tracta de deficiències en el disseny que indiquen que, encara que funcioni, no s'estan fent les coses de manera correcta en el nostre codi, la qual cosa acostumen a assenyalar una mala qualitat de codi i ens adverteixen que poden implicar futurs problemes. D'aquesta forma els Code Smells es poden definir com a símptomes que ens adverteixen d'una mala qualitat de codi que pot afectar negativament el nostre deute tècnic [10] .

Els code smells seleccionats són els següents:

2.3.2 Object-Orientation Abusers

Es tracten d'aplicacions incompletes o incorrectes dels principis de la programació orientada a objectes. [11] [12]

Refused bequest: Una subclasse només fa ús d'alguns dels mètodes i atributs de la seva superclasse, factor que pot indicar-nos que una herència és innecessària. [11] [12]

Alternative Classes with Different Interfícies: Es dona quan dues classes realitzen la mateixa funció però amb noms diferents de variables. [11] [12]

2.3.3 Change Preventers

Aquests smell ens indiquen que en fer canvis en un lloc del codi és necessari fer molts altres canvis en altres zones del nostre codi. La qual cosa torna el desenvolupament del nostre programa molt més lent i costós del necessari. [11] [12]

Divergent Change: Es dona quan necessites fer molts canvis en diversos mètodes de la nostra classe quan volem

modificar un només. [11] [12]

Shotgun Surgery: Fer un canvi en una classe requereix fer molts canvis petits en altres classes del nostre codi.[11][12]

2.3.4 Dispensables

Aquests smells es tracten de codi innecessari l'absència del qual faria que el codi fos més net, eficient i fàcil d'entendre. [11] [12]

Dead Code: Una classe, mètode, variable o paràmetre ja no es fa servir. A diferència de la generalitat especulativa el codi mort sí que va ser fet servir anteriorment, però ara està obsolet a causa de canvis en el nostre codi. [11] [12]

Codi duplicat: Presència de codi repetit. [11] [12]

2.3.5 Couplers

Aquests smells contribueixen a un acoblament excessiu entre classes o, d'altra banda, a una delegació excessiva en intentar evitar l'alt acoblament. [11] [12]

Feature Envy: Un mètode accedeix a les dades d'un altre objecte més que a les seves pròpies dades. [11] [12]

Middle Man: Es dona quan una classe únicament realitza una funció, delegant el treball en una altra classe diferent pel que probablement no és necessària l'existència d'aquesta classe. [11] [12]

Inappropriate Intimacy: Una classe fa ús dels atributs i mètodes interns d'una altra classe. [11] [12]

2.4 Eines i llibreries

Degut a que a l'hora d'obtenir algunes d'aquestes mètriques de forma automàtica resulta molt complex s'ha pres la decisió de realitzar una cerca i selecció de diferents d'eines que permetin calcular-les.

Per aquesta raó en aquest apartat es llistaran les eines trobades que poden resultar útils a l'hora de desenvolupar el codi que avaluarà les mètriques abans mencionades. Malgrat que no hi ha moltes llibreries, existeixen moltes eines *open source* que ens permeten avaluar mètriques de qualitat de codi.

Les eines seleccionades són les següents:

Metrinome: Permet convertir els arxius de c i c++ a un graf i després passar aquest graf per a calcular la complexitat ciclómatica. [**13**]

CMetrics: Permet mirar la complexitat ciclomàtica d'un arxiu o una funció. A més permet veure les línies de codi d'una funció i el número d'exit points. [**14**]

pmccabe: Comanda de bash que calcula la complexitat ciclomàtica de McCabe o els recomptes de línies no comentades per a C i programes C++.[**15**]

cppcheck: La funcionalitat que resulta interessant per a aquest projecte és l'anàliside l'estil del codi. [**16**]

cpplint: Comprova que el codi compleixi amb les normes d'estil de google. [17]

duplicate-code-detection-tool: Permet detectar similituds entre arxius dins d'un repositori de codis en java, C++, C#. C i python. [18]

duplo: Duplo és una eina per a trobar blocs de codi duplicats en grans bases de codi. Duple té suport per a diferents llenguatges de programació, cosa que significa que pot filtrar comentaris (multilínia) i directives de compilació. [19]

CK: És una eina que ens permet avaluar distintes mètriques de qualitat de codi i de mesurament en programació orientada a objectes per codis a Java. Les mètriques que permet avaluar que resulten més importants són cohesió, acoblament i mètriques de mesurament de nivell d'herència. [20]

3 OBJECTIU

Una vegada feta la investigació pertinent sobre la qualitat de codi i les eines i mètriques existents podem definir l'objectiu i l'abast d'aquest projecte.

Però abans de fer això s'ha de definir el context d'ús d'aquest projecte. Es tracta d'una eina que utilitzaran els professors de programació d'Enginyeria Informàtica de la UAB per a avaluar la qualitat dels codis pujats per tots els alumnes de l'assignatura a cada exercici. Inicialment, aquest projecte serà fet servir a l'assignatura de metodologia de la programació de primer de carrera, en aquesta assignatura s'imparteixen els conceptes bàsics de la programació orientada a objectes en C++, sense arribar a conceptes una mica més complexos com pot ser l'herència de classes.

Tenint en compte el context d'ús del projecte, l'objectiu principal és crear una eina que permeti avaluar la qualitat de diversos codis de cada exercici proposat a l'assignatura, lliurant llistes ordenades dels alumnes amb una valoració de la qualitat de codi de cada un i les mètriques que passen o no el llindar mínim definit per cada mètrica, que es definiran en reunions amb el tutor més endavant a l'hora de desenvolupar cada mètrica.

Com inicialment es tracta d'una assignatura que abasta només conceptes bàsics on els professors suggereixen una solució per cada exercici moltes d'aquestes mètriques s'avaluaran, a part d'amb valors absoluts de la mateixa mètrica, en comparació del codi de l'alumne amb la solució suggerida, així com mètriques pròpies com comparar quins tipus de bucles o condicionals s'utilitzen. Un altre objectiu és d'aconseguir que l'eina sigui escalable a noves mètriques o llenguatges, per tal de fer possible que en un futur es pugui usar en al-tres assignatures.

D'aquesta manera, l'objectiu principal serà dividit en tres objectius més petits, que representaran les tres fases principals del projecte:

- Desenvolupament de mètriques: La primera fase consistirà a investigar més a fons cada una de les mètriques seleccionades per tal d'entendre-la millor i desenvolupar la part del codi que permeti avaluar cada mètrica lliurant els resultats pertinents per cada mètrica oferint els resultats de la mètrica pel codi i, en cas que faci falta, donar una indicació de les parts concretes del codi que resulten més problemàtiques per aquest codi. En aquesta fase, com es tracta d'una assignatura on s'aborden conceptes més bàsics es donarà prioritat a la comparació de bucles i ifs, així com a l'estil de codificació amb normes definides des de l'assignatura, nombre de línies, codi innecessari i codi duplicat. Deixant en segon pla, per fer si dona temps suficient, mètriques útils però més complexes com la cohesió, l'acoblament, la complexitat ciclomàtica i mètriques de nivell d'herència. Aquesta decisió s'ha pres perquè es tracta d'exercicis tan simples que generalment resultarà menys útil, pel desenvolupament dels alumnes, avaluar-les encara que podria ser útil per altres assignatures en cas que volguéssim utilitzar l'eina en altres assignatures.

- **Avaluació d'un únic alumne:** En aquesta fase s'aprofitarà el codi desenvolupat en la fase anterior per avaluar totes les mètriques desenvolupades pel codi d'un únic alumne i lliurar tots els documents pertinents amb els informes pertinents per cada mètrica.

- **Avaluació de tots els alumnes:** En aquesta última fase es desenvoluparà la part del codi que permetrà avaluar tots els codis dels alumnes i lliurar llistes dels resultats generals de cada alumne.

4 Metodologia

Per tal de dur a terme aquest projecte es pretén fer servir la metodologia scrumban, que consisteix a fer servir una metodologia scrum en termes generals, però aplicant la metodologia kanban, amb el tauler propi d'aquesta metodologia, en cada sprint. S'ha escollit aquesta metodologia per tal d'oferir una planificació de temps lleugerament més estricte per cada sprint, per oferir les entregues als temps estipulats prèviament, però per oferir la flexibilitat que ofereix kanban amb les tasques que conformen cada sprint a dins de cada un per tal de poder desenvolupar de la millor manera possible cada sprint emmotllant les tasques de cada sprint a la disponibilitat de temps i recursos, així com la disponibilitat del tutor en cas de ser necessari fer alguna reunió.

S'ha aplicat realitzant diferents sprints més llargs de mida variable en funció de les necessitats de cada mètrica, de la disponibilitat de temps i la disponibilitat de l'alumne i del tutor per tal de fer reunions en cas de ser necessari.

Com s'ha mencionat prèviament s'han dut a terme diverses reunions amb el tutor durant els sprints, principalment en cas de necessitat per parlar de canvis en la planificació, l'estat del projecte o dubtes de com abordar certes mètriques o els informes lliurats pel codi en avaluar aquestes entre altres motius.

Durant aquesta fase del desenvolupament no s'ha fet servir cap suport visual similar al tauler kanban per fer el seguiment de les tasques de cada sprint i l'estat d'aquestes, en canvi, s'ha fet una llista de les tasques a realitzar durant el sprint amb l'estat d'aquestes per realitzar el seu seguiment.

5 Planificació de temps

La planificació s'ha vist modificada durant la durada del projecte, com que el temps disponible per dedicar-li al projecte ha sigut menor a l'esperat inicialment pel que tot el desenvolupament general del projecte ha acabat endarrerint-se respecte a la planificació inicial.

També es van modificar les tasques de cada fase de desenvolupament, ja que inicialment van ser pensades per realitzar cada fase de desenvolpuament com a un sprint independent de les altres amb tasques genèriques que eren les mateixes per a cada fase i es va decidir crear noves tasques més específiques per a cada una de les fases.

S'ha pres la decisió de modificar l'ordre de les mètriques a desenvolupar endarrerint el desenvolupament de la detecció de codi duplicat i avançant la de l'estil de codi, ja que aquesta última requeria parlar amb el tutor per prendre certes decisions a més que la considerem més important i que pot arribar a portar més temps de desenvolupament que el codi duplicat.

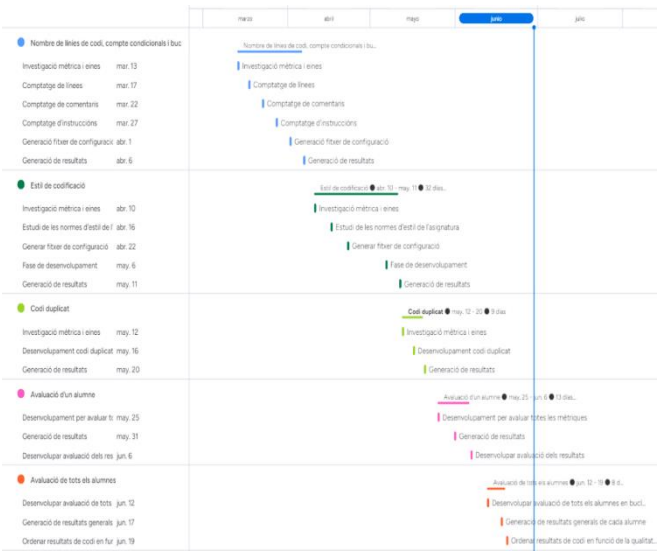
Així mateix, s'ha descartat la detecció de codi innecessari, perquè en estar desenvolupant una anàlisi de codi no estàtic per fitxers aquesta mesura no és del tot fiable, perquè es detectarien com a codi innecessari mètodes que només s'utilitzen en classes presents en un altre fitxer.

També he fet una adaptació de les tasques a causa del canvi en el procés de desenvolupament i seguiment de la metodologia. A més s'han retirat les fases de cohesió i acoblament i complexitat ciclomàtica, ja que en tractar-se d'unes mètriques opcionals que a priori no es pretenen fer, hem considerat millor no afegir-les a la planificació prèvia.

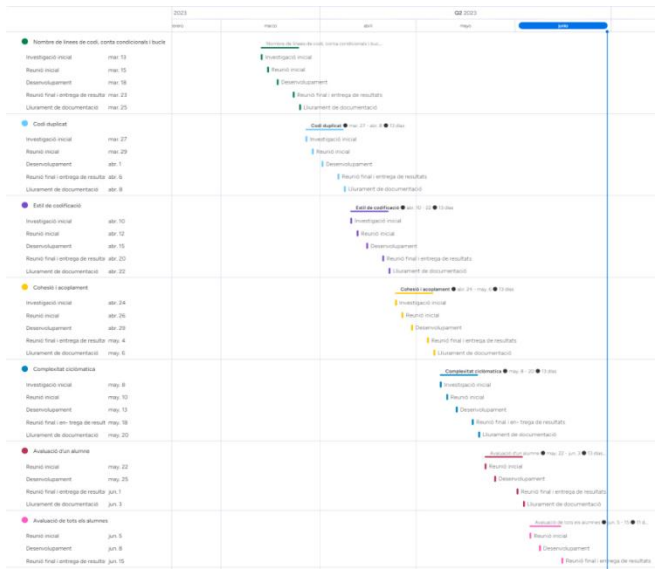
En la última fase s'ha endarrerit més la planificació general, ja que a causa de diversos problemes i obstacles s'ha trigat més de l'esperat en la tasca de generació d'un arxiu de configuració d'estil, i per això s'ha hagut d'ajustar la planificació de la resta de tasques posteriors.

A continuació es mostra les planificacions inicial i final després d'aquestes modificacions per poder veure les diferencies entre ambdues:

5.1 Diagrama de Gantt inicial



5.2 Diagrama de Gantt inicial



6 Desenvolupament

6.1 Nombre de línies de codi, compte condicionals i bucles

Per a la primera fase de desenvolupament el primer pas que es va seguir va ser la investigació de les eines que era possible utilitzar per fer el compte de línies de codi, comentaris i instruccions com for, if etc i com implementar-les al codi. Una vegada feta la investigació es va prendre la decisió que era més ràpid i senzill realitzar-ho jo mateix aprofitant les llibreries de python com glob i os, per obtenir els fitxers cpp de l'exercici de cada alumne i el path on es troben tant el projecte coms els exercicis.[21][22]

Després de prendre aquesta decisió es va començar amb la fase de desenvolupament on es va decidir crear un fitxer excel amb les instruccions que es vol comptar i retornar un excel amb el resultat del compte per arxius cpp i el total del projecte, i va aparèixer el problema de diferenciar aquestes instruccions, com per exemple for, de l'aparició d'aquestes paraules en comentaris i noms de variables i mètodes, investigant una mica vaig decidir solucionar-ho amb la llibreria regex. [23]

6.2 Estil de codificació

Per a la segona fase de desenvolupament, que consisteix en la avaluació de l'estil de codi on es va començar amb una nova fase d'investigació de les eines proposades prèviament i un estudi de les normes d'estil de codi que se segueixen a l'assignatura de metodologia de la programació per tal d'identificar si és millor fer una adaptació d'aquestes normes, si es pot configurar les eines mencionades prèviament per tal de seguir les normes de l'assignatura o si convé més fer-ho de zero sense fer ús d'aquestes eines. Finalment, es va arribar a la conclusió que el millor era utilitzar cppCheck, que es pot cridar com a comand-line fent servir l'script i permet crear un fitxer de configuració amb normes personalitzades, i en cas que algunes no es poguessin abastar, discutir amb el tutor si el millor era adaptar la guia d'estil de l'assignatura, programar-les des de zero o deixar que simplement aquesta anàlisi de codi no abasti aquesta norma en concret. [16]

Després de realitzar diverses proves i investigar més s'ha determinat que els mètodes que es van trobar per poder configurar normes d'estil de codi personalitzades estaven deprecated pel que es va descartar aquesta opció. També es va investigar l'eina cpplint que permet comprovar si l'estil del nostre codi segueix correctament les normes de google. Es va investigar aquesta eina per confirmar si és possible personalitzar les normes d'estil que comprova l'eina, però només ens permet seleccionar quines normes d'estil que proporciona l'eina volem o no comprovar, en cas que no es desitgi comprovar totes les normes i,

en cavi, es desitgi obviar alguna d'aquestes. [17]

6.3 Codi duplicat

La següent mètrica a realitzar per a la tercera fase de desenvolupament és la detecció codi duplicat, per realitzar-la primer es van revisar les eines prèviament cercades per detectar codi duplicat, de les que es van escollir duplo i duplicate-code-detection-tool. Entre aquestes dues eines es va seleccionar fer servir duplicate-code-detection-tool executant-lo per command-line pel fet que duplo donava més complicacions per a la integració de l'eina al codi del projecte. Duplicate-code-detection-tool ens permet comparar els diferents arxius d'un directori per detectar la probabilitat en percentatge que hi hagi codi repetit en ambdós fitxers. [18][19]

6.4 Avaluació d'un alumne i de tots els alumnes

Per a la quarta i última fase de desenvolupament s'ha realitzat les dues últimes parts de la planificació, avaluació d'un únic alumne i de tots els alumnes, degut a que encara que es tracta de dues parts diferents, s'ha considerat que estan altament relacionades entre si. La primera tracta de establir els llindars i, principalment, la manera de avaluar cada mètrica per cada alumne ja què es possible modificar els llindars, i la segona es tracta de realitzar aquest procés iteradament i lliurar un últim fitxer *excel* que reflexa els resultats generals de cada alumne, indicant l'exercici de l'alumne passa o no cada una de les mètriques avaluades i la valoració final d'aquest.

Per realitzar aquesta part del codi el que s'ha fet ha sigut realitzar una classe de codi que executa en bucle els mètodes que avaluen cada una de les mètriques per l'exercici de l'alumne. Pel que el primer ha sigut decidir com reconèixer l'alumne que ha fet cada exercici, pel que finalment s'ha decidit que es realitzi amb el nom del directori en que es troba l'exercici, pel que el codi s'hauria de lliurar en un directori amb el nom del niu de l'alumne, que s'hauran de situar al directori del projecte nomenat "exercicis" i els resultats es lliuraran en una nova carpeta amb el mateix nom que de l'exercici que es crearà al directori "Solucions".

La principal feina de desenvolupament en aquesta fase ha consistit en decidir la forma d'avaluació de cada mètrica, ja que els llindars escollits son configurables al fitxer de configuració general del projecte.

Per la nombre de línies de codi i el compte condicionals i bucles s'ha seguit el mateix mètode d'avaluació, que consisteix en comparar les comptes totals del projecte de la solució proveïda pels professors amb les de l'exercici general de l'alumne, si la similitud es igual o major al percentatge estipulat al fitxer de configuració, es considerarà aquesta mètrica com aprovada.

Per a l'estil de codi s'ha decidit llegir el fitxer lliurat per l'eina cpplint i comptar el nombre d'errors detectats per l'eina en qüestió, si aquest nombre es superior al valor del llindar que apareix al fitxer de configuració es considerarà la mètrica com a suspesa.

En quant a la mètrica de codi duplicat també s'ha escollit llegir el fitxer lliurat al treball i comprovar els percentatges de similitud entre tots els fitxers del projecte, si algun d'aquests es superior al percentatge indicat al fitxer de configuració s'avaluarà la mètrica com a suspesa.

Finalment per cada mètrica aprovada l'alumne obtindrà un punt que es sumarà per obtenir la nota final de la seva avaluació de qualitat de codi i s'afegirà a una llista ordenada per aquest valor que retornarà l'alumne

7 Resultats

7.1 Explicació resultats

A continuació es presentaran els diferents resultats que s'han obtingut actualment durant el procés de desenvolupament d'aquest projecte, que consisteixen principalment en el codi del projecte i diversos arxius de configuració i de resultats de les mètriques.

El codi actual permet avaluar les diferents mètriques proposades durant l'objectiu i planificació del projecte juntament amb alguns fitxers de configuració i proveint diferents fitxers que retornen els resultats de cada una de les mètriques que es comproven.

Els fitxers de configuració actuals són, per una part un arxiu de configuració on es poden canviar els paths per defecte dels diferents fitxers de configuració, noms de fitxers ignorats o els paths en els que es troben els codis dels alumnes, la solució del professor o on es guarden els diferents resultats. L'altre fitxer de configuració en l'actualitat és un fitxer que ens permet indicar les diferents instruccions que volem cercar en el codi i ele-ments que poden resultar importants a l'hora de buscar-los per descartar casos on aquestes paraules apareguin sense ser una instrucció com per exemple un for o un if, com pot passar a comentaris per exemple.

Els fitxers de resultats que s'han obtingut són un fitxer que retorna el nombre de cops que apareix cada una de les instruccions que apareixen a l'arxiu de configuració, un altre que retorna el compte, per fitxer de codi i total, de les línies de codi, línies buides, comentaris i línies totals del fitxer.

Un altre dels resultats són un fitxer per cada fitxer de text que retorna tots els problemes d'estil de codi detectats per cpplint, per avaluar l'estil del codi.

I finalment es retorna un fitxer de codi que retorna la similitud entre tots els fitxers del directori de l'exercici, amb l'objectiu veure la probabilitat que hi ha de trobar codi duplicat en més d'un fitxer o classe de codi.

Entre els fitxers de resultats finalment trobem un fitxer excel que retorna l'avaluació final i de cada mètrica de tots els alumnes, indicant si l'exercici passa o no cada mètrica i la nota final de l'alumne.

7.2 Exemple resultats

En aquest apartat s'ha seleccionat un projecte simple de github anomenat per fer una execució de prova del projecte i poder mostrar a continuació exemples dels resultats lliurats pel projecte. [24]

Els primers resultats lliurats pels exercicis son uns arxius excel que entrega el nombre de línies de cada arxiu i el total de l'exercici i el nombre d'aparicions de cada tipus d'instrucció que apareix a l'arxiu de configuració.

A continuació es mostra l'exemple del resultat del comptatge de línies:

filename	lines	code lines	comments	empty lines
exercicis/11111111/PositionReporter	36	16	9	11
exercicis/11111111/MyProject	5	2	1	2
exercicis/11111111/MyProjectGameModeBase	8	2	1	5
exercicis/11111111/OpenDoor	61	42	8	11
Total	110	62	19	29

I l'exemple del comptatge d'instruccions:

filename	for	if	else	switch	while	do
exercicis/11111111/PositionReporter.cpp	0	0	0	0	0	0
exercicis/11111111/MyProject.cpp	0	0	0	0	0	0
exercicis/11111111/MyProjectGameModeBase.cpp	0	0	0	0	0	0
exercicis/11111111/OpenDoor.cpp	0	4	1	0	0	0
Total	0	4	1	0	0	0

El següent resultat lliurat pel projecte es un fitxer de text que retorna tots els problemes d'estil de codi detectats per cpplint.

A continuació es pot veure un petit fragment del fitxer dels errors d'estil de codi detectats:

```
Code duplication probability for /Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/PositionReporter.h
--------------------------------------------------------------------------------
                                    File                                    Similarity (%)
--------------------------------------------------------------------------------
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/MyProject.Build.cs           7.56
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/OpenDoor.h                   65.19
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/MyProject.cpp                6.33
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/MyProjectGameModeBase.cpp    3.02
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/PositionReporter.cpp        32.68
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/MyProjectGameModeBase.h     18.13
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/OpenDoor.cpp                18.31
/Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/MyProject.h                  8.04


Code duplication probability for /Users/hugo/PycharmProjects/CodeQuality/exercicis/11111111/MyProject.Build.cs
```

Altre dels resultats es el fitxer de text que retorna la probabilitat de que hi hagi codi duplicat a dos fitxers de codi diferents.

A continuació es mostra un petit fragment d'exemple d'aquest fitxer:

```
exercicis/11111111/MyProject.cpp:5:  Extra space after ( in function call  [whitespace/parens] [4]
exercicis/11111111/MyProject.cpp:5:  Extra space before )  [whitespace/parens] [2]
exercicis/11111111/MyProjectGameModeBase.h:13:  { should almost always be at the end of the previous line  [whitespace/braces] [4]
exercicis/11111111/MyProjectGameModeBase.h:15:  Line ends in whitespace.  Consider deleting these extra spaces.  [whitespace/end_of_line] [4]
exercicis/11111111/MyProjectGameModeBase.h:16:  Line ends in whitespace.  Consider deleting these extra spaces.  [whitespace/end_of_line] [4]
exercicis/11111111/MyProjectGameModeBase.h:17:  Line ends in whitespace.  Consider deleting these extra spaces.  [whitespace/end_of_line] [4]
exercicis/11111111/MyProjectGameModeBase.h:18:  Line ends in whitespace.  Consider deleting these extra spaces.  [whitespace/end_of_line] [4]
exercicis/11111111/MyProjectGameModeBase.h:18:  Redundant blank line at the end of a code block should be deleted.  [whitespace/blank_line] [3]
exercicis/11111111/OpenDoor.cpp:9:  { should almost always be at the end of the previous line  [whitespace/braces] [4]
```

L'últim resultat lliurat pel projecte es un fitxer excel que mostra una llista ordenada dels resultats de cada mètrica per cada exercici lliurat per l'alumne i el total.

A continuació es pot observar un exemple:

Estudiant	Compta linea	Compta instr	Estil de codi	Codi duplicat	Total
11111111	1	1	1	0	3

En aquest cas només es mostra un únic exercici perquè només s'ha avaluat un únic exercici per poder veure exemples dels resultat.

8 Conclusions

Després d'haver realitzat el projecte ha sigut possible entendre encara millor la importància de la qualitat de codi en vers al deute tècnic per tal d'evitar que un projecte no arribi a un punt on no sigui possible continuar sense perdre temps a refer parts del codi que es van realitzar sense tenir en compte el resultat final.

Una vegada arribat a aquesta fase del projecte ha sigut possible comprovar que les mètriques escollides durant la fase d'objectius han estat ben escollides tenint en compte el context d'ús de l'aplicació desenvolupada, encara que hi ha hagut certs problemes o falles en el desenvolupament de cada mètrica, en concret s'han trobat errors a l'hora de fer la comprovació d'estil de codi, ja que no ha sigut possible escollir les normes d'estil a plaer, per la qual cosa per poder aplicar aquesta mètrica s'hauria de prendre la decisió de canviar l'estil de codi proposat a l'assignatura de Metodologia de la Programació per les normes d'estil de google, encara que en un futur seria possible investigar eines noves que ens permeti escollir les normes d'estil personalitzades.

L'altra mètrica que no ha resultat de la millor manera possible és la detecció de codi duplicat, ja que busca codi duplicat entre diferents fitxers, a més que només dona una probabilitat que hi hagi codi duplicat pel qual no és massa útil en el context suggerit on es presenten exercicis on habitualment hi ha un únic fitxer modificat per l'alumne, l'ideal consistiria en un futur investigar com modificar aquesta mètrica per poder retornar els blocs de codi repetits o molt similars en un mateix fitxer de codi que podrien ser substituïts per un únic mètode que pot ser cridat des de diversos punts del codi.

9 POSSIBLES LÍNIES DE MODIFICACIONS FUTURES

En aquest apartat es llistaran algunes possibles línies de lectura possibles per fer modificacions o afegir noves característiques al codi del projecte.

Aquestes possibles modificacions futures son les següents:

- Afegir en algunes mètriques com el comptatge de línies i de nombre d'instruccions de codi la possibilitat d'avaluar-les per mètodes a part de per fitxer, així com tenir en compte aquests valors i els resultats per fitxers i no només els resultats totals de l'exercici.

- Acabar de modificar el codi per que sigui escalable completament per a altres llenguatges de programació a part de c i c++.

- Modificar la implementació de l'estil de codi per tal de que sigui possible avaluar-lo en funció de les normes d'estil de codi de l'assignatura i poder configurar normes d'estil personalitzades.

- Afegir una funció a la mètrica de codi duplicat per tal de poder detectar blocs de codi duplicat a un únic fitxer a part de buscar codi duplicat entre fitxers com s'avalua actualment.

- Afegir noves mètriques d'avaluació com podrien ser la cohesió, l'acoblament o la complexitat ciclomàtica.

- Optimitzar el codi per reduir la complexitat ciclomàtica de l'eina i reduir el temps d'execució d'aquesta.

- Millorar la forma en la que s'obtenen els projectes i es lliuren les solucions per poder fer-ho més ràpid i còmode per a l'usuari.

- Afegir una interfície gràfica per millorar l'experiència de l'usuari a l'hora de fer servir l'aplicació i llegir i accedir els fitxers de resultats.

BIBLIOGRAFIA

[1] Calidad del código: qué es y cómo verificarlo. (2021, 28 diciembre). Apliint Software Development. https://apliint.com/2021/12/28/calidad-del-codigo-que-es-y-como-verificarlo/ Referència 2

[2] A. (s. f.). Qué es la deuda técnica y cómo saldarla (con ejemplos) •. Asana. https://asana.com/es/resources/technical-debt

[3] Deuda tÃ©cnica: las posibles consecuencias de ahorrar en el desarrollo de software. (2020, 15 octubre). IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/deuda-tecnica-explicada/

[4] ¿Qué es la calidad del código? (s. f.). parasoft. https://es.parasoft.com/solutions/code-quality/#:~:text=Las%20métricas%20de%20calidad%20del,código%20es%20de%20alta%20calidad.

[5] M. (2022, 22 octubre). Cálculo de métricas de código - Visual Studio (Windows). Microsoft Learn. https://learn.microsoft.com/es-es/visualstudio/code-quality/code-metrics-values?view=vs-2022

[6] Juan V. Carrillo (jvprofe). (2016, 2 mayo). Complejidad ciclomática: cómo calcularla. YouTube. https://www.youtube.com/watch?v=iILl-n57IEs

[7] LCOM4 y la falta de cohesión en las clases. (2011, 10 diciembre). Jummp. https://jummp.wordpress.com/2010/05/22/lcom4-y-la-falta-de-cohesion-en-las-clases/

[8] A. (s. f.-a). Project Metrics Help - Cohesion metrics. https://www.aivosto.com/project/help/pm-oo-cohesion.html

[9] CBO (Coupling Between Object Classes). (2010, 30 junio). Jummp. https://jummp.wordpress.com/2010/06/12/cbo-coupling-between-object-classes/

[10] Code smell. (2022b, noviembre 21). IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/code-smell/

[11] Un repaso por los code smells más comunes. (s. f.). https://intive.com/es/careers/un-repaso-por-los-code-smells-mas-comunes

[12] Catalog of Refactoring. (s. f.). https://refactoring.guru/es/refactoring/catalog

[13] H. (s. f.-c). metrinome/src at develop · hmc-alpaqa/metrinome. GitHub. https://github.com/hmc-alpaqa/metrinome/tree/develop/src

[14] M. (s. f.-c). GitHub - MetricsGrimoire/CMetrics: CMetrics measures size and complexity for C files. GitHub. https://github.com/MetricsGrimoire/CMetrics

[15] C. (s. f.-c). Ubuntu Manpage: pmccabe - calculate McCabe cyclomatic complexity or non-commented line counts for C and. https://manpages.ubuntu.com/manpages/xenial/man1/pmccabe.1.html

[16] D. (s. f.-d). GitHub - danmar/cppcheck: static analysis of C/C++ code. GitHub. https://github.com/danmar/cppcheck

[17] google. (s. f.). styleguide/cpplint at gh-pages · google/styleguide. GitHub. https://github.com/google/styleguide/tree/gh-pages/cpplint

[18] P. (s. f.-h). GitHub - platisd/duplicate-code-detection-tool: A simple Python3 tool to detect similarities between files within a repository. GitHub. https://github.com/platisd/duplicate-code-detection-tool

[19] D. (s. f.-e). GitHub - dlidstrom/Duplo: Duplicates finder for various source code formats. GitHub. https://github.com/dlidstrom/Duplo

[20] M. (s. f.-h). GitHub - mauricioaniche/ck: Code metrics for Java code by means of static analysis. GitHub. https://github.com/mauricioaniche/ck

[21] os — Interfaces misceláneas del sistema operativo — documentación de Python - 3.10.11. (s. f.). https://docs.python.org/es/3.10/library/os.html

[22] os — Interfaces misceláneas del sistema operativo — documentación de Python - 3.10.11. (s. f.). https://docs.python.org/es/3.10/library/os.html

[23] Python RegEx. (s. f.). https://www.w3schools.com/python/python_regex.asp

[24] DavidRVi. (s. f.). MyProject/Source/MyProject at master · DavidRVi/MyProject. GitHub. https://github.com/DavidRVi/MyProject/tree/master/Source/MyProject