題目：基於DevOps之程式作業品質測試分析系統

(一) 專題內容製作動機與預期目標

在大一程式設計這堂課程中，使用了系上老師自己開發的線上自動評分系統，這套系統只能正確的判斷程式執行的結果，當時的我們也就直覺的認為程式只要結果對就好。到了大二，在軟體工程這堂課中，我們了解到了軟體程式品質的重要性，如果只在乎結果，程式中可能存在許多不必要的變數或結構，而不好的變數的宣告方式及糟糕的函式寫法都有可能產生安全性問題、影響可讀性，從而降低程式的品質。

藉由系上資訊專題課程的契機，我們預計開發出一套能判斷程式執行結果正確性，且能夠分析程式品質的教學輔助系統。對於這套系統，我們將重點放在如何檢測程式品質以及如何提升使用者程式碼品質上。檢測品質的部分，會利用SonarQube來檢測上面所使用的三項指標：安全性、可靠性以及可維護性。提升程式碼品質的部分，在檢測完後，系統會將可改進的部分標記起來，並提供範例給使用者參考，此外，系統也會將每次結果紀錄下來，生成程式碼的品質曲線圖，以便了解學習狀況。  
目前預計的系統功能：

(1) 提供單元測試以及品質檢測。

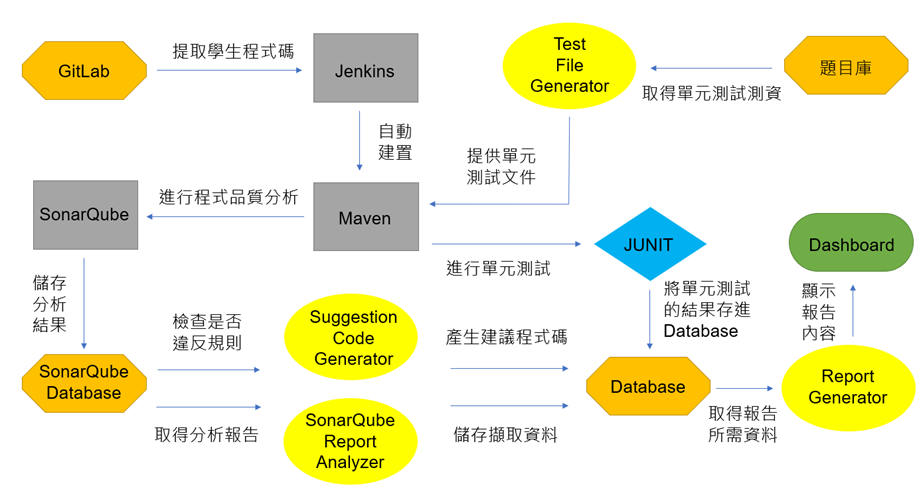
(2) 根據學生程式碼錯誤提供對應的建議。

(3) 老師在出題時可以自由選擇自己出題或是從題庫出題。

(4) 提供品質曲線圖以便學生了解進步狀況。

(5) 報告結果整合到儀表板上。

(二) 整體時程與工作規劃



系統資料流

本計畫的研究步驟分為三個階段，分別為：

階段一：基本環境建構(GitLab, Jenkins, SonarQube, Maven, Database, 題目庫)及Test File Generator 的製作

階段二：Suggestion Code Generator, SonarQube Report Analyzer的製作

階段三：加入前端，及Report Generator的製作

階段一時，會先建構基礎的環境，包含GitLab, Jenkins, SonarQube和Database的架設。其中GitLab會作為每個學生繳交作業的工具，為了防止學生互相偷看程式碼、抄襲作業，目前預計每一位學生每一題都建立一份專案，設定成只有該專案的成員才能進入該專案。

Jenkins的部分，我們會在Jenkins安裝GitLab的plugin，然後在GitLab拿到SSH的金鑰，再回到Jenkins配置token，最後設定當有程式碼提交時，就會觸發Jenkins的部署操作。如此每當GitLab有新的檔案上傳時，Jenkins便會自動建置專案並對程式碼進行單元測試及品質測試。

接著會將SonarQube加入系統中，架設SonarQube後與Jenkins連接，也會架設屬於SonarQube的資料庫，SonarQube會將程式碼掃描後的結果儲存到SonarQube的資料庫。SonarQube產生的報告內容將安全性、可靠性及可維護性各自分為A,B,C,D,E這五個等級，可維護性藉由評估Technical Debt Ratio來給予等級，其中Technical Debt定義為將所有code smell修正所需花費的時間，可靠性則是藉由程式碼產生的Bugs的嚴重程度來給予等級(Blocker Bugs, Major Bugs, Minor Bugs, Critical Bugs)，安全性藉由評估程式碼的Vulnerability issues嚴重程度來給予等級。另外還會提供覆蓋率，重複率，以及程式碼的Top Common Issues, Top Issues By Severity, Top Security HotSpots to Review。

題目庫的部分預計將整個題目分為題目敘述、測資、答案三個部分，且題目庫中會記錄題目的類型，名稱，敘述，編號，Input，Output，及圖片。老師在出題時，除了可以選擇題目庫中的題目外，也能自行出題。

單元測試則是透過Maven default生命週期的test階段綁定了Maven-surefire -plugin插件，該插件可以調用Junit完成單元測試，我們會製作一個Test File Generator，將題目庫中老師選取的題目的測資和答案，做成單元測試的程式。目前計畫系統會先產生單元測試程式的基礎樣本，透過這個樣本我們將資料庫中的測資和答案寫到樣本中，最後將程式移到Maven專案中。

而在專案自動建置的過程會需要pom.xml來設定專案的版本、定義要用到的插件還有專案自動建置時運作的順序(script)，這裡我們提供兩種方法來產生pom.xml，第一種是一開始就產生一個基本的pom.xml，具備基本運行規則，出題老師若須要而外的功能則須修改pom.xml來達到需求。第二種方法是在出題之前勾選系統列出的功能中所需要的部分，由系統自動生成pom.xml。

第二階段會製作SonarQube Report Analyzer，能解析SonarQube的報告內容，並把系統會需要使用到的資訊儲存到系統資料庫，以及製作能生成建議程式碼的Suggestion Code Generator，以下敘述各個元件的實作方式。

(1)Suggestion Code Generator：

目前計畫參考[4]的實作方式，在其研究中提供了SonarQube中最常被手動更正的11個靜態代碼分析規則，依據這些規則，我們會製作每條規則的代碼轉換模板(code-transformation templates)，利用這些模板套用到使用者的程式碼中，即可產生建議程式碼。而在系統確實產生建議程式碼前會先針對SonarQube檢測產生的結果進行檢查，若確實違反了其中某項規則，才會套用模板。

以下為Suggestion Code Generator運作方法：

a.針對SonarQube的報告內容偵測是否違反預先決定的11條規則。

b.若有違反規則，找出使用者程式碼中違反規則的程式碼片段。

c.套用代碼轉換模板，產生建議程式碼。

以下為論文[4]提供的C8規則的代碼轉換模板(code-transformation templates)：

refactoredExp = parse(#Expression, "<beforeFunc>.isEmpty()");

此代碼轉換模板能套用在違反規則C8的程式碼片段上，能將Collection.size()==0替換為Collection.isEmpty(),如此便可消除SonarQube品質檢測所產生的issues。

(2)SonarQube Report Analyzer：

目前計畫SonarQube Report Analyzer會擷取包含三部分:

a.三項指標(安全性，可靠性，可維護性)的評估等級

b.報告中較嚴重的issue(Major issues, Critical issues, Blocker issues)

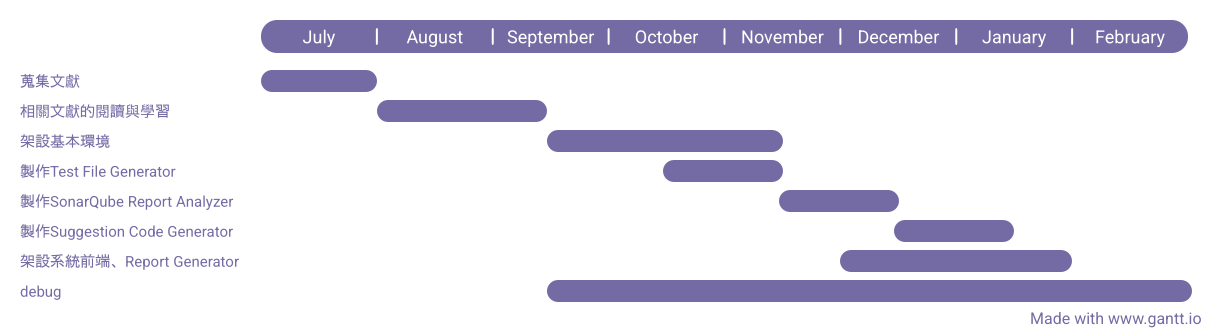
c.程式碼產生的Bugs數目，Vulnerability數目，及code smell的數目

目前預計利用SonarQube提供的的web API中的Get(內容位於api/measures/ component及api/issues/search中)將上述三項(a, b, c)的內容提出，並且利用Restful API中的Post，將需要的資料存入系統的資料庫中。

第三階段，我們會開始著手處理前端，及Report Generator的製作。

前端的部分會將重點放在儀表板上，儀表板會顯示該堂課老師出了哪些作業、各個作業所有學生的繳交狀況，點進該作業自己的名字能看到自己每次作業建置的狀況和報告。

Report Generator 會從系統的資料庫中，抓取每次繳交產生的Bugs數目，vulnerability數目及code smell的數目，分別對應到安全性，可靠性及可維護性，利用資料庫中各項目歷次的數目來繪製三張曲線圖，並且Report Generator 會將本次建置的分數，三張曲線圖，及程式碼的修改建議彙整成一份報告傳送至儀表板，讓學生了解作業的作答情況，老師也能查看學生作業繳交狀況。

以下為計畫的甘特圖:  


(三) 經費規劃及使用情形

此專案不須花費任何經費，故無經費規劃及使用情形。

(四) 可能遭遇之困難及解決方法

以下為目前所遭遇之困難：

1. Test File Generator的製作方式。
2. Suggestion Code Generator的實作過於複雜。
3. 系統帳號及GitLab帳號的整合。
4. SonarQube Report Analyzer的實作方式。

針對第一點，目前想出兩種解決方式，一為使用Jenkins 提供的Pipeline project，撰寫Pipeline Script，並在其中預處理的Stage中執行Shell Script，生成單元測試所需使用的測試程式。二是自行撰寫Jenkins的Plugins，在專案建置的過程中執行Plugin。

針對第二點，原先的預想是自行寫出可產生建議程式碼的Suggestion Code Generator，但經過資料的蒐集後發現過於複雜，因此我們決定轉為應用原先已有的專案內容，融入我們的專題中，以提供學生程式碼改進的建議。

針對第三點，我們有找到可以