Mock-up exam 2IMP25 Software evolution

This is a closed-book exam. No books, lecture notes, slides or any auxiliary material is allowed.

The exam consists of five questions; you should answer four of them. Should you decide to answer all five questions, only the first four questions will be graded.

When answering questions try to be brief but complete.

1. Software metrics.
    a. (5) Give a definition of cyclomatic complexity. Give a small example to illustrate the definition.
    b. (10) In the original paper by Chidamber and Kemerer LCOM(*C*) has been defined as *P-Q* if *P>Q* and *0*, otherwise; where *P* is the number of pairs of distinct methods in the class *C* that do not share instance variables, and *Q* is the number of pairs of distinct methods in the class *C* that share instance variables.
        i. (2) Identify the shortcomings of this definition.
        ii. (4) Propose an alternative that does not suffer from the shortcomings identified.
        iii. (4) Illustrate the difference between the original definition and the one proposed in 1-b-ii by means of an example.
    c. (10) The following figure from Businge et al. shows the mean and the standard deviation of the normalized distance from the main sequence *Dn* over a number of revisions of PyDev. Describe the evolution of PyDev using this figure.

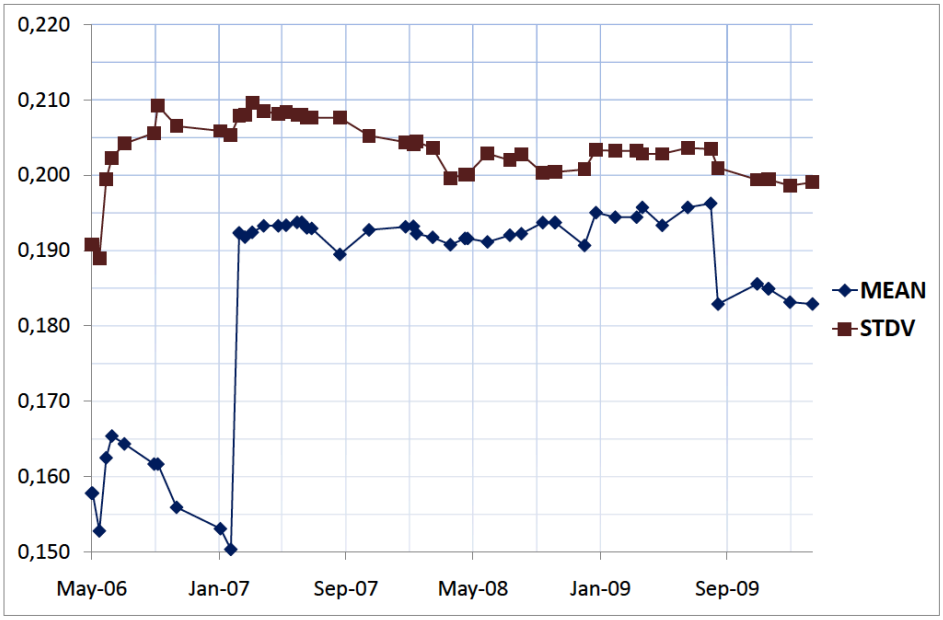


Figure 7: Growth Trend of $\bar{D_n}$'s and $\sigma(D_n)$'s for $PyDev$ plug-in

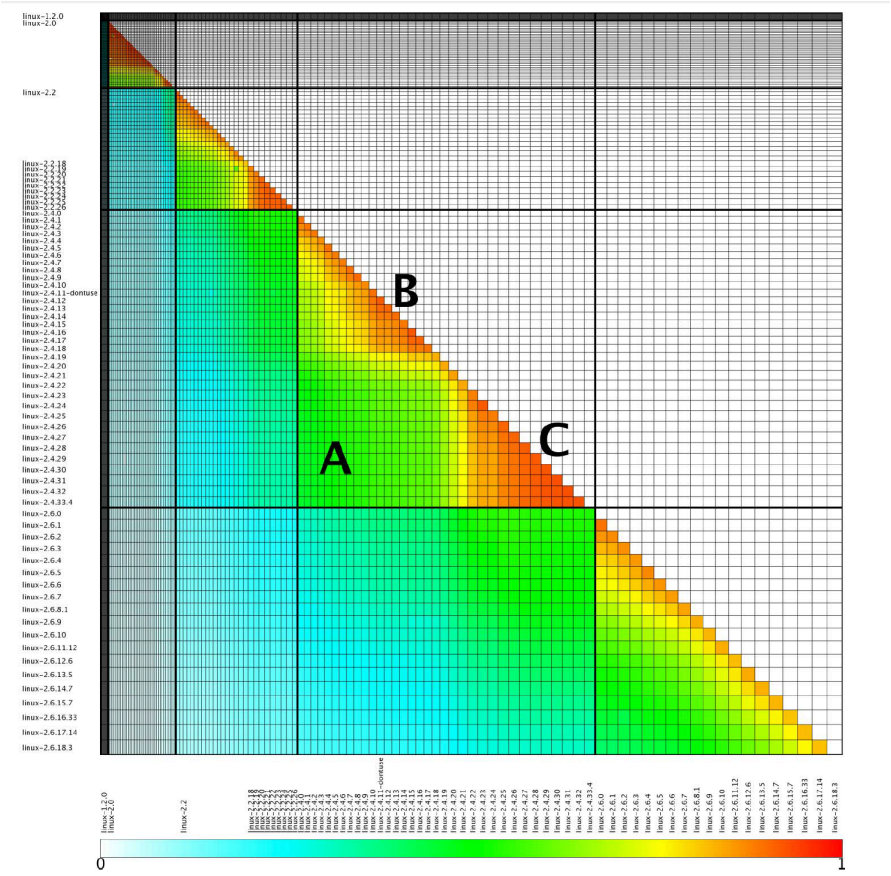
Here I would expect that you to at least make the following observations and interpret them:

- The mean is *lower* than the standard deviation. Explain what this would mean and how would this be related to the histograms of Dn distribution you have seen in the class.
- Combine the previous observation with the range of the values observed (0.15-0.19). What would this mean in terms of the "main sequence"?

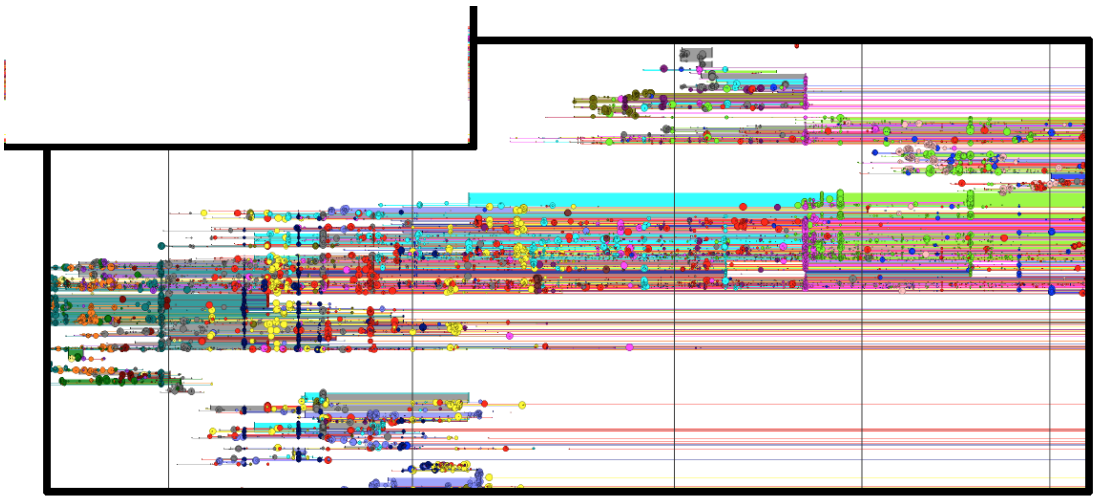
- Trends. In general standard deviation seems to decrease, while the mean is more or less stable between January 2007 and August 2009. How would you explain this observation?
- Singularities. For instance, in January 2007 the mean has greatly increased and around September 2009 it decreased while the standard deviation remained more or less the same. What could have caused this?

2. Code duplication.
    a. (5) Define the notion of Type 2 clones. Give a small example to illustrate the definition.
    b. (10) Compare the token-based approach of Brenda Baker's and the AST-based approach of Ira Baxter et al. What are the advantages and the disadvantages of each one of the approaches?
    c. (10) The following figure from Livieri et al. shows cloning between different versions of the Linux kernel. The figure is a heat-map with lower values being "cold" (blue) and higher values being "hot" (red). The value visualized represents the coverage of the source code of two versions by code shared (cloned) between those two versions; the higher the value the more similar those versions are. The figure has been obtained using D-CCFinder, the distributed version of the CCFinder tool discussed in the class. Please ignore characters A, B and C.

       Describe evolution of the Linux kernel based on this figure.



3. Repository mining.
    a. (5) Give a definition of a distributed version control system. Name at least two distributed version control systems.
    b. (10) While some version control systems such as CVS record information per file, some others such as Subversion record it per commit. What are advantages/disadvantages of each one of the decisions?
    c. (10) The following figure from Gîrba et al. represents the ownership map of JBoss (please ignore the cut-out part in the left upper corner). Discuss the evolution of the JBoss developers' community and system organization based on this figure.



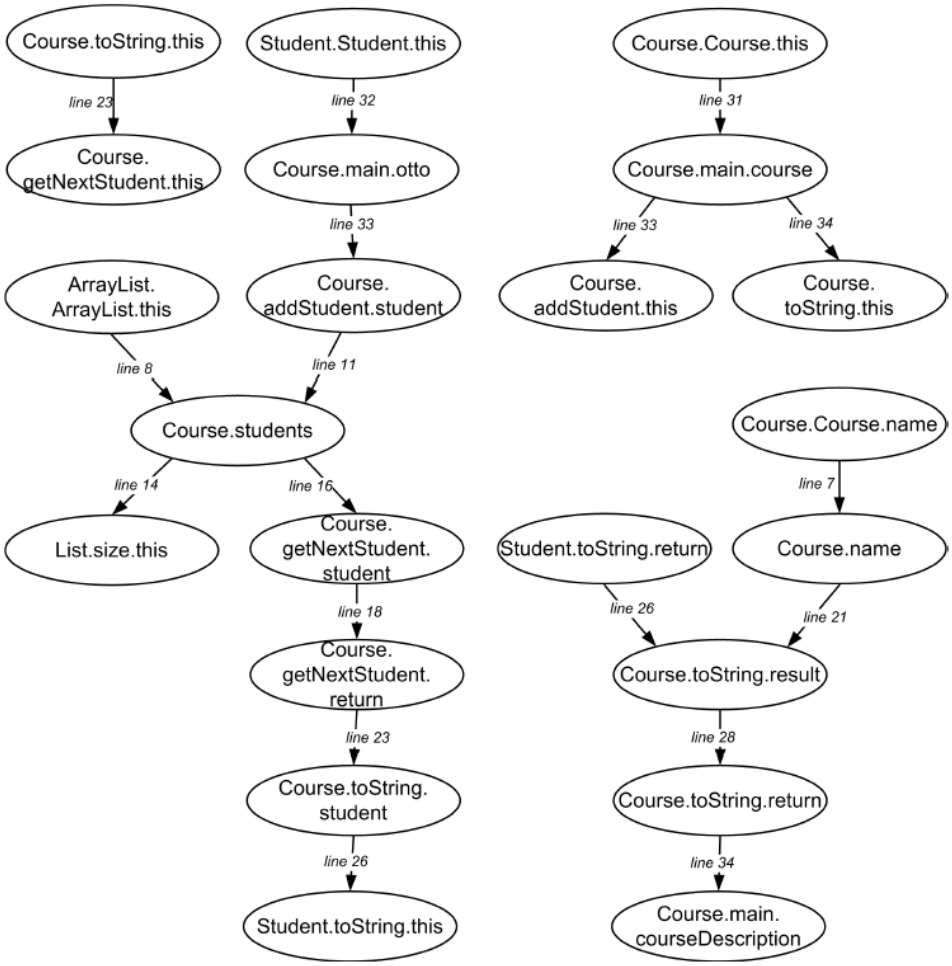
Here I would expect you to observe that (a) the color-mix changes with the time, suggesting that the developers' team composition has significantly changed over time (see, e.g., appearance and disappearance of the cyan developer), and (b) the changes in the team composition are accompanied by many files being added (new lines) and removed (lines stop) representing changes in the system organization---"new lords, new laws".

4. Architecture reconstruction
    a. (5) The term “reverse engineering” is sometimes used as a synonym to “architecture reconstruction”. Explain the difference between the two terms and give an example when one of the terms is applicable and another one is not.
    b. (10) Consider the following example from the Software Maintenance lecture notes by Franz Wotawa, Roxane Koitz and Birgit Hofer. Construct an Object-Flow Graph for this example.

```
1 class Course {
2   String name;
3   List<Student> students;
4   int iterator=0;
5
6   public Course (String name) {
7     this.name = name;
8     students = new ArrayList<Students>();
9   }
10  public void addStudent (Student student) {
11    students.add(student);
12  }
13  public Student getNextStudent(){
14    if(!(iterator<students.size()))
15      return null;
16    Student student = students.get(iterator);
17    iterator++;
18    return student;
19  }
20  public String toString(){
21    String result = name;
22    while(true){
23      Student student = getNextStudent();
24      if(student==null)
25        break;
26      result += student.toString();
27    }
28    return result;
29  }
30  public static void main(Strings args[]) {
31    Course course = new Course("Software maintenance");
32    Student otto = new Student();
33    course.addStudent(otto);
34    String courseDescription = course.toString();
35    System.out.println(courseDescription);
36  }
37 }
```

Solution (please ignore the annotations):

This is the resulting OFG:



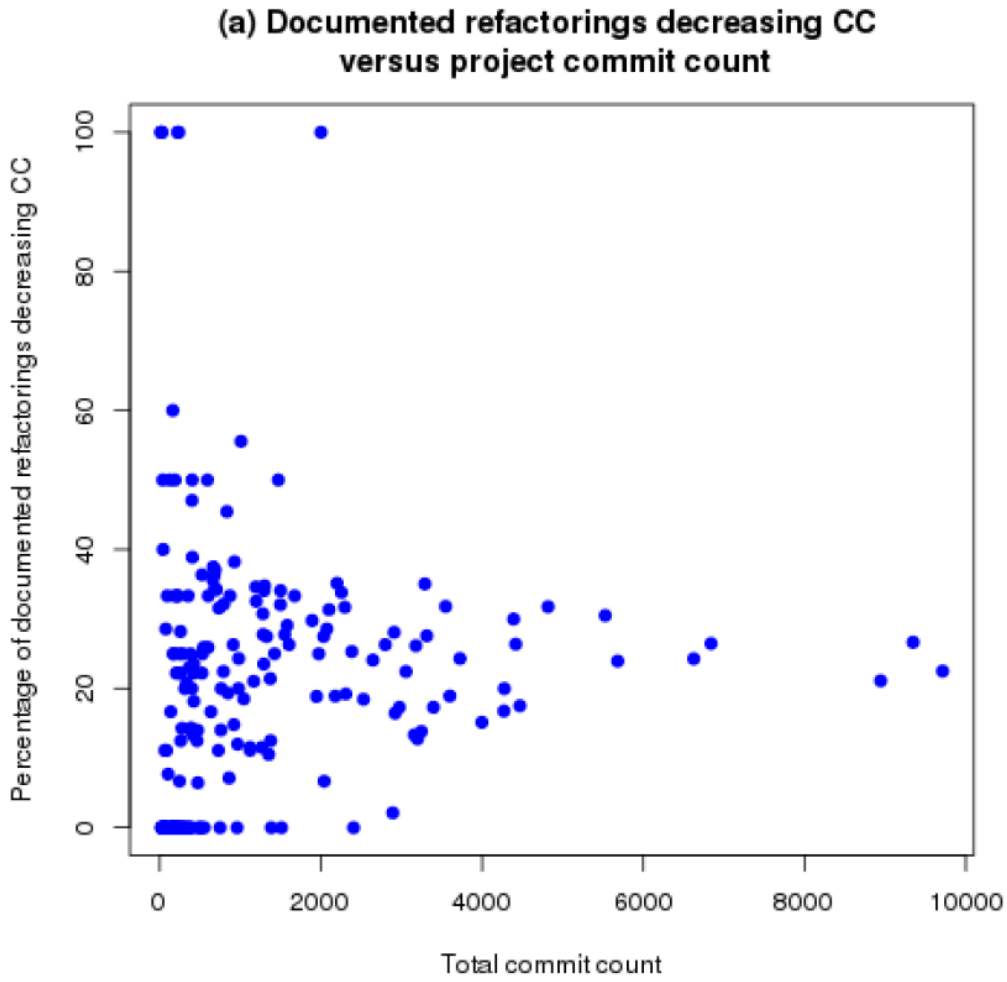
c. (10) One of the major problems related to analysis of the reconstructed class diagrams is the level of detail: automatically reconstructed class diagrams contain far too many details for humans to oversee. Present at least two different techniques to tackle this problem and discuss their advantages and disadvantages.

5. Refactoring
    a. (5) Give a definition of refactoring
    b. (10) Recall the classification of refactorings based on their impact on the interface as proposed by Moonen et al.: compatible, backwards compatible, can be made backwards compatible, and incompatible.
        i. (2*4) For each one of the classes: give an example of refactoring belonging to this class. Explain why the refactoring belongs to this class.
        ii. (2) Furthermore, Moonen et al. have identified a series of smells in the test code, such as Test run war (concurrent use of resources) and Mystery guest (dependency on an external resource). How can one refactor the test code to eliminate the Test run war and Mystery guest smells?
    c. (10) Sokol et al. have studied the impact of refactoring on source code complexity, measured in terms of the total cyclomatic complexity of a project (CC). They have distinguished between ”documented refactorings”, i.e., commits with an associated message containing words such as “refactoring”, “refactored”, “refactor”, and so on; and remaining commits. They have studied Java projects of the Apache Software Foundation and observed that 1504 documented refactorings decreased cyclomatic complexity, 1603 did not affect it and 3230 increased cyclomatic complexity.

        The following scatter plot shows the percentage of commits that decreased Cyclomatic Complexity by each project’s total commit count. Each dot represents a project and only the projects with documented refactorings were considered.

        Based on this study the authors conclude that no evidence can be found to support the positive effect of refactorings on the Cyclomatic Complexity.

        What kind of issues might have threatened validity of this conclusion?



Here at least the following issues should have been mentioned:

- Identification of "documented refactorings": the word "refactoring" might be misused by the developers, e.g., to denote any kind of change ("refactored XXX to fix bug YYY"), making the approach to identify spurious refactorings, and describe refactorings without using the word "refactoring" ("extracted method XXX"), making the approach to miss the refactorings.
- Code commits usually integrate multiple changes, some of them being related to refactoring and some not being related to refactoring, i.e., attributing change in CC to refactoring only might be preposterous.
- Cyclomatic complexity might not be an appropriate complexity measure for object-oriented programs; e.g., it does not take inheritance into account. Moreover, getters/setters usually have cyclomatic complexity of zero.
- Conclusions from the Apache Software Foundations might not be generalizable beyond the projects of this foundation. Similarly, conclusions from the Java projects might not be generalizable to other programming languages.

- The dataset shows high variance, i.e., large differences between different projects, i.e., conclusion based on the complete dataset cannot be applied to individual projects.