Student number:	2467273
Course title:	COMPSCI4015 Professional Software Development (H)
Questions answered:	ALL

1.

a) Sprint planning meetings have the objective of discussing goals, including new features, bug fixes and code improvements, with the customers to work on for the next few weeks. In contrast, release planning meetings consist of multiple sprints and have the objective of agreeing on a high-level roadmap with the customers for the most important and general features they want to be implemented.

b) The problems and their solutions:

- Problem: The customer seems to have very little influence on which user stories are done in which sprints, Lesley being the one who prioritises the 5 requirements from the 8 given.
  - Solution: Involve the customer in the sprint planning meetings as well to learn their priorities as it is their product that the team is working on.
- Problem: The user stories are allocated to developers only by Lesley, possibly not taking into account the developers' skills and desires.
  - Solution: Have a discussion on which user stories cater more to each team members' skills and which ones they would prefer to do.
- Problem: Assuming all sprints are of similar length, 10 user stories are chosen to be done in the coming sprint in contrast with the 5 done in the previous one, which possibly means double workload and incomplete tasks.
  - Solution: Prioritise tasks to be done in this sprint by keeping in mind how much the team were able to do before.
- Problem: User stories are all estimated to be the same by Lesley instead of trying to figure out why there is such a discrepancy between team members' estimates of the user stories.
  - Solution: Devote more time into planning poker or another estimation exercise instead of giving up. If there is no more time left in the day, start most immediate tasks but plan another meeting to discuss further estimation of the tasks.
- Problem: There is no discussion of the specific schedule of the tasks to be done.
  - Solution: Have all team members set individual deadlines for the tasks and establish a week-by-week schedule.
- Problem: The problem of broken builds being deployed is never addressed, only its fix is found.
  - Solution: Have a team discussion on how viable making a CI service would be and whether it would save time and errors in the long run. If found to be important, prioritise the CI service over some less important feature user stories.
- Problem: There is no discussion about how the work done in the previous sprint would influence work for the next one.
  - Solution: Discuss as a team how accurate the estimations for the tasks were in order to more accurately pinpoint what types of tasks are more appropriate for which team members and how quickly and effectively they can do them. Any problems encountered before could also be relevant for the next sprint.
- Problem: No theme set for the coming sprint.
  - Solution: Establish a theme based on the types of user stories worked on for the customer to better understand what is being done.

- Problem: The meeting is much too long (more than 3 hours), meaning that too much time is spent on talking about tasks than actually doing them.
    - Solution: Work more efficiently by discussing things in the group and not working "independently", and devoting time to more important issues, like discussing and fixing the broken builds and specific tasks to do.
- Problem: The meeting is set on a Friday, which means that a lot of tasks might not be done and forgotten about during the weekend.
    - Solution: Set the meeting time on a Tuesday/Wednesday when all team members have gotten into their work and will have time later in the week to work on their tasks.

c) I absolutely disagree with the proposal as starting to work overtime will damage the morale of the team and is mentally unhealthy for any software developer. Firstly, it can lead to a snowball effect where working longer but with the same inefficient methods will make the software process less efficient and lead to even more overtime. Secondly, there needs to be a root cause analysis of the reason why they have not achieved their user stories, as working overtime is only treating the symptom, when a much better alternative would be to reassess their process and be honest with their customers.

2.

a)

User story	Violating criterion	Reason	Revised user story
5.	Small	There are too many features described, which could be split into independent user stories without difficulty. Moreover, the character and time limit are too specific for a user story and belong elsewhere.	Split into 2 different user stories: 1. As a gym user, I want to provide constructive comments on videos that have been shared by my friends so that they benefit from my opinion. 2. As a gym user, I want to be able to add a Like, Hate, Love or Laugh emoji reaction to my friends' videos so that they benefit from my opinion.
4.	Independent	It is dependent on the previous user story as it mentions friends in the network the user has built "above".	As a gym user I want to share a recording of use of an item of gym equipment with my friends so I can get constructive feedback.
6.	Estimable	"Free automated feedback" from a video recording would involve very complicated technology, e.g., machine learning networks, which would be very hard to estimate how long it would take (I would assume – very long).	As a gym user, I want to get free instructions on how to use equipment I choose in the app so that I can follow them and thus exercise more effectively.
2.	Negotiable	There is no specificity of the users who would like to record videos, that is, there is no mention of the gym, which would lead readers to think that this app is made for more than just gym users.	As a gym user I want to be able to delete any of my videos so that I can prevent my account from becoming cluttered and remove recordings I don't like.
3.	Valuable	There is no value added to the phrase "my personal social network", e.g., why they want that network, as it is just a paraphrase of "select other users as my friends", i.e., circular reasoning.	As a gym user I want to select other users as my friends so that I can share my fitness progress with them and see how they are doing.

b) I would prioritise the user stories as follows:

- Must have:
    - 4. This user story is a prerequisite for any social aspect to work since without recordings there would be nothing else (that is mentioned) to share with friends. Sharing videos could also be done publicly if needed.

- Should have:
    - 3. The most attractive feature of a lot of apps nowadays is the social aspect of them, and this user story would allow users to make their gym experience more social than usual, which is unique enough to attract new users. It is a basic prerequisite to make the social aspect work.
- Could have:
- Would want to have:
    - 5. This user story adds more functionality to the social aspect of the app, which, although is not completely necessary for the app to be usable, is a very important feature to make the social aspect more appealing. If there is budget for 60% of user stories, this would be top priority.
    - 6. This user story would be extremely appealing to new users since it is free feedback on personal fitness, but it would likely be impossible/extremely difficult to implement (as mentioned in part a).
    - 2. Deleting videos is a valued feature, but it is not something that appeals to new users necessarily, only a mild annoyance for long-time users.

3.

a) An architectural pattern solves the problem of designing a whole system architecture with components that can be reused for different purposes. In contrast, a design pattern solves problems for the behaviour between classes and how they interact with each other, including inheritance, abstractness, etc.

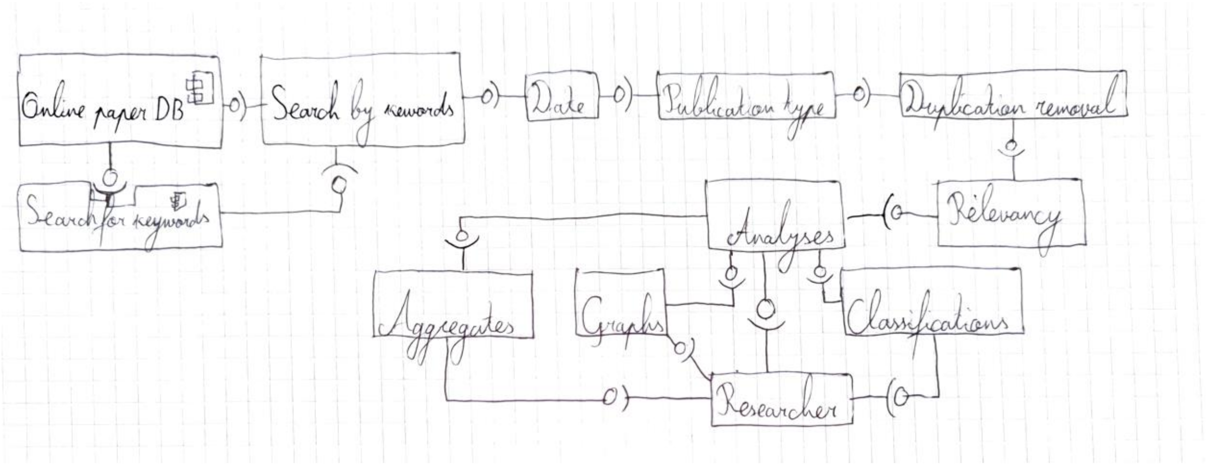
b) The appropriate architectural pattern for the SLR system would be a **pipe and filter** architecture because:

- filtering research papers based on different criteria directly translates to using the architecture with Filter components on information flow;
- modifications on the architecture allow implementing specific features mentioned in the requirements, like reorderability, etc..

c) The modifications:

- ***Pull*** data flow because the data does not change drastically enough to keep a continuous flow of all repositories and their changes. This way, filters can be more effective because they use output from other filters more without worrying about the original data source.
- A **hybrid** between concurrency and a sequential nature of the pipeline. Some filters can work concurrently, like duplication and irrelevancy removals, but others need to be executed only at the end, like aggregate analysis filters.
- A **hybrid** between reorderable and interchangeable filters to allow researchers to customise them as required, but the initial keyword search and PDF/BibTeX/graph output filters cannot be reordered.
- **Branching** filters to allow more potential analysis of the papers by the researchers and for different outputs from the filters, including graphs and formats (PDF or BibTeX).

d)



e) Other databases, like DBLP and IEEEXplore, could be added as parallel data sources to the architecture. In this case, more discussion should be led whether aggregate analysis (papers per year, etc) was to be made on all of them together, in which case all databases would be put into one pipeline, or to separate the pipelines. In the latter case, clones of the same filters could be reused to process papers from these alternate databases, and graphs could be compared only at the end.

f) I disagree because:

- it would be redundant behaviour since filters are already of a plug-in nature and they could be interchanged with the third-party tools proposed;
- results are meant to be consistent between researchers, and third-party tools would possibly violate this consistency;
- the project would be in danger of exhibiting the inner platform effect; thus, it would be better to use plug-in architecture for the extension of the current pipe and filter architecture.

4.

a) Firstly, there is a refactoring opportunity identified from bad smells in the code. Then, a test case is created to check for correct behaviour of the code fragment so that nothing is broken in the refactoring process. Then, the refactoring is planned and applied. Afterwards, the test is run with the new refactoring in place. If the test fails, the refactoring plan is changed and retried. If the test passes during any retrying (or first try), the refactoring is committed.

b)

Smell	Why smell makes it harder to maintain	Refactoring	How refactoring will improve maintainability
Primitive obsession in “String commentDate”, “String commentURL”, and others	Raw text is very hard to work with when there are numbers involved as they might be changed, the string needs to be checked for valid dates or URLs, there is no programmatic distinction between the different parts of the strings.	Replace the data values with a Date or URL object.	This would ensure that different parts of the concepts are kept separate, like the domain and protocol in a URL or day and month in a date. The code would then be explicit on what part of the object it is processing. Validation would be easier.
Data clumping in “commentDate” and “commentURL”	It is harder to establish their connection to the same concept (a comment) since they are independent variables. This difficulty can then result in confusion of attributes for different comments.	Extract variables as attributes to a new Comment class.	This would ensure that attributes do not get mixed up as well as improving developers’ understanding of the project since classes are more understandable than their attributes being on the same hierarchical level as all other variables.
Long parameter list for “sendNewComment-Notification”	It is hard to read the code with this smell, which in turn invites bugs and misunderstandings between developers. It is also harder to understand what the relevance of each of the parameters is.	With the earlier refactoring (Comment class), preserve the whole object.	With this refactoring, all comment-related attributes would be replaced with a Comment object, which is readable and understandable.
Duplicate code for both present methods	When one method is changed, the other needs to be changed the same way to ensure consistency, and keeping track of both is difficult and invites errors.	Make a template method that encapsulates functionality of both.	This ensures that consistency is kept, reduces code length and improves legibility of the code.

c) The new if clause would be a case of duplicated code since it would be exactly the same or very similar to all other instances of the clause. This repetition means that any change to one clause would be needed to be mirrored to all other instances of the same clause, which is very difficult to keep track of. The solution to this would be to form a general template method of sending a notification which could be specified with a parameter whether to send to a slack channel or to the email address. This would reduce repeated code and make it much shorter but still understandable to any developer reading the code. It would also mean that a change to the if clause would translate to all its use cases.