Project 2 Rubrics

Team 72 -

- Om Tandel
- Snehil Behar
- Devang Saraogi

Github Link - https://github.com/Software-Engineering-Folks/RecipeRecommender/tree/sfe-master

Sum - 144

Notes		Evidence
Workload is spread over the whole team (one team member is often Xtimes more productive than the others…	2	Evidence in GH
but nevertheless, here is a track record that everyone is contributing a lot)	2	evidence in GH
Number of commits	58	in GH
Number of commits: by different people	18	in GH
Issues reports: there are **many**	0	

Issues are being closed	0	evidence in GH
Docs: doco generated, format not ugly	2	in GH
Docs: what: point descriptions of each class/function (in isolation)	2	
Docs: how: for common use cases X,Y,Z mini-tutorials showing worked examples on how to do X,Y,Z	2	doc page entries
Docs: why: docs tell a story, motivate the whole thing, deliver a punchline that makes you want to rush out and use the thing	2	
Docs: short video, animated, hosted on your repo. That convinces people why they want to work on your code.	2	
Use of version control tools	2	

Test cases exist	2	dozens of tests and those test cases are more than 30% of the code base
Test cases are routinely executed	2	E.g. travis-com.com or github actions or something
Issues are discussed before they are closed	0	even if you discuss in slack, need a sumamry statement here
Chat channel: exists	2	Link or screenshots
Test cases: a large proportion of the issues related to handling failing cases.	0	If a test case fails, open an issue and fix it
Evidence that the whole team is using the same tools: everyone can get to all tools and files	2	
Evidence that the whole team is using the same tools (e.g. config files in the repo, updated by lots of different people)	2	
Evidence that the whole team is using the same tools (e.g. tutor can ask anyone to share screen, they	2	

demonstrate the system running on their computer)		
Evidence that the members of the team are working across multiple places in the code base	2	
Short release cycles	2	(hard to see in short projects) project members are committing often enough so that everyone can get your work
The file .gitignore lists what files should not be saved to the repo. See [examples]i(https://github.com/github/gitignore)	1	in GH
The file INSTALL.md lists how to install the code	1	in GH
The file LICENSE.md lists rules of usage for this repo	1	in GH
The file CODE-OF-CONDUCT.md lists rules of behavior for this repo; e.g. see example	1	in GH

The file CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up; e.g. see example	1	in GH
The file README.md contains all the following	1	in GH
Video	2	2min video of new functionality, showing a significant delta from prior.
DOI badge: exists. To get a Digitial Object Indentifier, regiser the project at Zenodo. DOI badges look like this:	1	in GH
Badges showing your style checkers	1	config files in GH showing your config, badges in README
Badges showing your code formatters.	1	config files in GH showing your this formatter's config, badges in README
Badges showing your syntax checkers.	1	config files iin GH showing this checker's config, badges in README

Badges showing your code coverage tools	1	config files in GH, badges in README
Badges showing any other Other automated analysis tools	1	config files in GH, badges in README
Is the name of your project/software unique?	1	You shouldn’t choose a project or software name that is shared by another group – especially if they are a competitor – but, sadly, few people spend enough time researching the uniqueness of their project or software names. It is, now, less important to have a domain name that mimics the name of your project or software, because most people will use a web search to find a website rather than manually entering a URL. However, it is important to check, using a web search, that there are no existing domains that include the name of your project or software that are owned by another group, because this can confuse your users. See our guide on Choosing project and product names (http://software.ac.uk/resources/guides/choosing-project-and-product-names).

Is your software available as a package that can be deployed without building it?	1	Building software can be complicated and time-consuming. Providing your software as a package that can be deployed without building it can save users the time and effort of doing this themselves. This can be especially valuable if your users are not software developers. You should test that your software builds and runs on all the platforms it is meant to support, which means you will already have created packages that can be distributed to your users! See our guide on Ready for release? A checklist for developers (http://www.software.ac.uk/resources/guides/ready-release). If you're interested in the consequences of ignoring your users' needs, see our guide on How to frustrate your users, annoy other developers and please lawyers (http://www.software.ac.uk/resources/guides/how-frustrate-your-users-annoy-other-developers-and-please-lawyers).
Do you publish your release history e.g. release date, version numbers, key features of each release etc. on your web site or in your documentation?	1	A release history allows users to see how your software has evolved. It can provide them with a way to see how active you are in developing and maintaining your software, in terms of new features provided and bugs fixed. Software that is seen to

		be regularly fixed, updated and extended can be more appealing than software that seems to have stagnated.
Question 7.1: Does your software allow data to be imported and exported using open data formats? (e.g. GIF, SVG, HTML, XML, tar, zip, CSV, JSON, NetCDF, or domain specific ones)	0	Supporting open data formats, data formats which are publicly available, as a complement to any proprietary ones you need to support, has many benefits. If data can be imported and exported in an open format, then it can be used with any software that uses this format. Software that supports open formats do not lock users into that software, because they will be able to use other software to access their data, if necessary. This can make it easier for users to swap between software that uses open formats. This means that users of other software may switch to your software, if your software is more efficient, robust, scalable or functional than that of your competitors'. Try and adopt open formats that are mature, ratified standards, if possible. Standards can go through many iterations, because they evolve as ideas are proposed and debated and the scope, remit and intent of the standards are agreed. If a standard changes, then any software that uses the standard needs to be changed to keep up to date. Most of the big changes occur early in the lifetime of a standard. Mature and ratified standards are less likely to change significantly or frequently, which reduces the risk of

		you having to modify your software in response.
Are releases tagged in the repository?	2	Every time a change is committed to source code held under revision control, a revision number or commit identifier is created. Though plain-text, these are not usually human-readable. Many revision control tools allow repositories to be tagged, whereby a useful, memorable name can be given to a specific version. Tagging releases in this way (e.g. release-1.0.1 or conference-09-2015) can make it easier for both you, and users, to get access to the source code that was included in a particular release, or used to create the results you reported in a particular paper, for example.
Question 10.4: Is there a branch of the repository that is always stable? (i.e. tests always pass, code always builds successfully)	1	Having a stable branch of the repository means that you and your users and developers always know where to get an up-to-date version of the code that is known to compile successfully and for which your tests pass. For example, users might want to use an up-to-date copy from the repository to benefit from bug fixes made since your last release. A policy on who can merge code into this branch, and under what conditions, can help preserve its stability.

Question 11.2: Can you build, or package, your software using an automated tool? e.g. Make (https://www.gnu.org/software/make/), ANT (http://ant.apache.org/), Maven (https://maven.apache.org/), CMake (https://cmake.org/), Python setuptools (https://pypi.python.org/pypi/setuptools), or R package tools (https://cran.r-project.org/doc/manuals/r-devel/R-exts.html)?	1	Typing in lots of instructions is both time-consuming and prone to error. An automated build/packaging tool can make building or packaging your software easier, and less error-prone. Automation is also useful for developers: it makes it easier for them to rebuild or repackage code after implementing extensions, enhancements or bug fixes.
Question 11.6: Does your software list the web address, and licences for all third-party dependencies and say whether the dependencies are mandatory or optional?	1	Users don't want to have to search the web for your third-party dependencies to find the information they need to package or deploy your software. You already know all the information that your users will need about suitable versions, licences and suchlike, so you should make it available to your users. In particular, licence information is very important, because users need to understand the terms and conditions of third-party dependencies so that they can determine whether they are legally permitted to use them, and, so, use your software.

Question 11.8: Do you have tests that can be run after your software has been built or deployed to show whether the build or deployment has been successful?	0	Tests give a user confidence that your software has built and installed correctly on their platform. If a test fails, the nature of the failure can help the identify why e.g. maybe the user forgot a configuration step, or provided an incorrect configuration option. For developers, tests contribute to a fail-fast environment, which allows the rapid identification of failures introduced by changes to the code such as optimisations or bug fixes. Tests are an important aspect of maintainable software. There are many frameworks available for writing tests in a range of languages, including JUnit (http://junit.org/) for Java, CUnit (http://cunit.sourceforge.net/) for C, CPPUnit (http://www.freedesktop.org/wiki/Software/cppunit/) and googletest (https://code.google.com/p/googletest/) for C++, FRUIT (http://sourceforge.net/projects/fortranxunit/) for Fortran, py.test (http://pytest.org/) and nosetests (http://nose.readthedocs.org/) for Python, testthat (https://cran.r-project.org/web/packages/testthat/index.html) for R and PHPUnit (https://phpunit.de) for PHP. Alternatively, you can provide a list of steps that a user can take to check the deployment of the software e.g. for a web-based application, this might just be

		checking that the application is accessible via a browser.
Question 12.5: Are all manually-run tests documented?	0	It may not be possible, or easy, to automate certain tests e.g. testing a browser-based application after it's been deployed. In such cases, you should document the list of steps that are to be done to test the software. Documenting the steps means that the tests can be run by anyone, not just the developer who usually does these tests.
Question 13.1: Does your project have resources (e.g. blog, Twitter, RSS feed, Facebook page, wiki, mailing list) that are regularly updated with information about your software? (e.g. release announcements, publications, workshops, conference presentations)	1	These resources are all good ways of showing that your project is active. If potential users see frequent posts, especially if they talk about new features and resolved problems, they know that your project is thriving, your software is useful and is under active development. This may encourage them to use your software, knowing that if they run into problems, there may be people who can help, and who they can share experiences with. These resources also give you a way of getting in touch with your current users, keeping them engaged, and asking for their input or help with problems. Don't know what new feature to implement next? Ask your users whether they think it would be useful.

Does your website state how many projects and users are associated with your project?	0	Where you have an active set of users and developers, advertising their existence is not just good for promoting the success and life of your project. If potential users see that there are a large number of users, they know that your project is thriving, your software is useful and is under active development. This may encourage them to use your software, knowing that if they run into problems, there may be people who can help, and who they can share experiences with.
Question 13.3: Do you provide success stories on your website?	0	A great way of showing off your software is to write case studies about the people who've used it and how they've used it. This helps potential users learn about the software but, more to the point, is a great advert for your software. If you can show happy users benefiting from your software, you are likely to gain more users.
Question 13.4: Do you list your important partners and collaborators on your website?	2	Providing a list of important partners and collaborators gives potential users valuable assurance that your software has a future. Also, the higher the scientific, academic or industrial reputation of those partners, the higher the perceived reputation of your software, and project, will be. Publicly recognising partners' efforts in improving or working with your software also increases the

		likelihood they will continue to use, or develop, your software in the future. Credit where credit is due!
Question 13.5: Do you list your project's publications on your website or link to a resource where these are available?	1	Listing your software publications provides an academic perspective on the value of your software. It can also help users, and other stakeholders (e.g. current and potential funders) to understand, in detail, how your software contributes to research, what scientific problems it has helped to solve. In addition, these can help to show where your software sits in relation to other software that fulfils a similar need, and what makes yours different, or better. These publications also give researchers something to cite when they write their own papers where your software has been used, which is of value to them and also increases your citation count for your papers, which helps you demonstrate your impact!
Question 13.6: Do you list third-party publications that refer to your software on your website or link to a resource where these are available?	2	Providing a list of third-party publications can show, academically, how the software is used by others, as well as promoting their efforts and successes. It also gives potential users ideas for how they may choose to use the software, as well as providing assurance that the software can be used by people other than its original developers to achieve

		something. Having such a list also means you can cite these publications in your own papers, funding proposals and reports to show or justify its value and the impact you have made! As a matter of routine, you should always ask people to cite your software if they've used it in their research for these reasons, and to inform you if they have included such a reference in one of their papers.
Question 13.7: Can users subscribe to notifications to changes to your source code repository?	0	Keeping information on these notifications as open as possible helps you present the impression that your project is open and inclusive. If users are actively developing using your software, keeping them up to date with on-going development on your source code (e.g. implementation of enhancements, extensions or bug fixes) enables them to factor these into their own development plans. For example, users might want to use an up-to-date copy from the repository to benefit from bug fixes made since your last release. If the notifications include information on the changes made (e.g. excerpts of the source code showing additions, removals and alterations) then this may allow for rapid identification of bug as any subscriber may notice an issue in the changes made. A lightweight, automated subscription process can reduce,

		even remove, from you the overhead of managing notifications. It also means that users aren't subject to long delays waiting for their membership to be approved.
Does each of your source code files include a copyright statement?	2	It’s easy to distribute source code files, and this separates the code from any copyright statement that might be on your web site or in your documentation. To cover this eventuality, and remove any ambiguity about ownership, it's good practice to include a copyright statement with each of your source code files, as a comment, or, if the language permits it, as a string constant.
Question 15.6: Does each of your source code files include a licence header?	2	It’s easy to distribute source code files, and this separates the code from any licence statement that might be on your web site or in your documentation. To cover this eventuality, and remove any ambiguity about what a developer can do with the source code, it's good practice to include a licence statement within each of your source code files, as a comment. This can also help to avoid confusion between source files that may have different licences, particularly if there are a number of third-party dependencies used within your software.
Question 15.7: Do you have a recommended	2	Asking that users cite your software, either directly or via its associated

citation for your software?		publications, provides you with credit for develop your software. It also provides a means, via harvesting of citations, of gathering evidence of the uptake and exploitation of your software. See, for example, Citing R (https://cran.r-project.org/doc/FAQ/R-FAQ.html#Citing-R) and Citing Taverna (http://www.taverna.org.uk/cite/). See our guide on How to cite and describe software (http://software.ac.uk/so-exactly-what-software-did-you-use) and examples of the citations recommended by various software packages (http://www.software.ac.uk/blog/2014-07-30-oh-research-software-how-shalt-i-cite-thee).
Question 16.1: Does your website or documentation include a project roadmap (a list of project and development milestones for the next 3, 6 and 12 months)?	1	A roadmap allows users to see when new features will be added and plan their project accordingly. It also has an important secondary benefit: one of the most important factors that will influence a user's choice of software is the likelihood of that software still being around – and supported – in the future. There are many ways in which a project can persuade a user of its longevity: regular announcements, regular releases, prompt replies to queries, information about funding and its plans for the future – a roadmap.
Question 16.2: Does your website or	0	Especially on academic projects, users will view the active lifetime of

documentation describe how your project is funded, and the period over which funding is guaranteed?		software to be the duration of the software's project funding. If you want to persuade users that your software will be around in the future, it is a good idea to describe your funding model and the duration over which funding is assured.
Question 16.3: Do you make timely announcements of the deprecation of components, APIs, etc.?	1	It's never a good idea to remove components or features without giving your users an advance warning first. It could be there are users who are dependent on the feature(s) you plan to change or remove. Announcing such planned deprecations well in advance means users and developers can respond if a given feature is important to them. If a feature is due to be superseded by a newer, better feature or component, including both for a suitable period within the software can allow your users to transition comfortably from the older version to the new version. You could also consider developing and publicising a deprecation policy, stating how and when features or components in general are deprecated. This gives your users assurance that features will not be removed without warning. see, for example the Eclipse API deprecation policy. (https://wiki.eclipse.org/Eclipse/API_Central/Deprecation_Policy).