Risk Assessment & Mitigation

We are using the risk assessment standard ISO 31000:2009 [1-2], which includes ISO 31010. We have used the Indian version of this, as neither the British nor International version of the standard were accessible to us.

The format that we shall be using for the risk assessment will be a table. This format will outline the risk and give it a score composed of the likelihood of it happening. We then multiply this by the impact that it will have on the project. It will also contain what we are doing to mitigate the impact of the risk on our project.

We have tried to identify as many risks as possible, however we have left out risks that would be included if this was being done in a software engineering firm. These include: high employee turnover, senior staff not having software engineering experience, the project affecting company profits/reputation, any cost based risks, etc. We do not believe that these risks are relevant for a group project at university. However, we have included risks that would not apply to a normal software engineering business but does apply to a university group project, a prime example is difficulty working over the Christmas vacation.

Risk assessment should be broken down into 3 steps:

- Risk identification: during this step we identify as many possible risks as we are able to do. We are using some of the methods outlined in appendix B of the standard, including: the swift technique, checklists (this includes adapting risks from literature [3]) and brainstorming.
- Risk analysis: in this step we look at each of the identified risks and decide what the likelihood of them occurring is, what effects would they have on the group project, and a general understanding of the risk. We are using a numerical scoring system to measure the risk on our project. This will take into account the likelihood of the risk happening and the impact it will have on the project. The impact is rated from 0, indicating no impact, through to 1, indicating complete failure of the project. The likelihood is rated from 0 to 1 (impossible to certain, retrospectively). These two numbers are then multiplied together to create a score that will determine the level of risk [4].
- Risk evaluation: during this process we determine the significance of the risk. We have colour coded this, red indicates a higher than is acceptable, orange is high but acceptable risk, green is very low risk. There are no red risks after mitigation.

After the risk assessment has been completed, we then move on to risk treatment. During this step we identify any measures that we have taken to mitigate the risks that we have identified. We then decide if the residual risk level is acceptable for our project (it is "non-critical software" so it is acceptable to have some residual risk.)

[1] Bureau of Indian Standards, *IS/ISO 31000:2009; Indian Standard; RISK MANAGEMENT - PRINCIPLES AND GUIDELINES*. New Delhi, India: BIS, 2011.
[2] M. Bin Mohamad Zain (2016, Jan. 30). *"MS IEC/ISO 31010 – Risk Management - Risk Assessment Techniques"*. mqm2016 [Online]. Available: https://mqm2016.files.wordpress.com/2016/01/awareness-and-application-of-ms-iec_iso-31010_2011-risk-assessment-techniques1.pdf. [Accessed: 31- Oct- 2016].
[3] T. Arnuphaptrairong, "Top Ten Lists of Software Project Risks : Evidence from the Literature Survey", in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2011*, IMECS 2011, Hong Kong, March 16-18, 2011, S. I. Ao, O. Castillo, C. Douglas, D. Dagan Feng, J. Lee, Eds. Hong Kong: Newswood Limited, 2011. pp. 732-37.
[4] B.W. Boehm, "Software risk management: principles and practices" *IEEE Software*, vol. 8, no. 1, pp. 32-41, January 1991.

Risk management should be done continuously, and not just to satisfy project reviews [5]. This is because otherwise some risks will always be missed. According to the ISO standard, the process should be “dynamic, iterative and responsive to change” [1].

Risk	**Score (likelihood * impact)**	**Mitigation**
One of our team becomes ill [6] and is unable to do their portion of the work (normal work).	0.02 (0.2 * 0.1)	There should be at least 2 people who can do any one task. Everyone should know what each team member is doing. We shall create a work schedule to keep track of who is working on what.
One of our team becomes ill and is unable to do their portion of the work (critical work).	0.12 (0.2*0.6)	All critical code should be stored online and should be committed and synced regularly. At least one other person should be informed when a person is working on critical code.
A computer component fails, destroying/deleting all the work.	0.001 (0.001*1)	All code should be stored on GitHub and should be committed and synced regularly. A local copy should also be stored. All documents should be stored on Google Drive where possible. We cannot find an example of Google Drive failing and losing data, so we think that this is a good way of mitigating this possibility.
The “customer” becomes unavailable to talk to, through illness, maternity/paternity leave, holidays etc.	0.0002(0.2*0.001)	The “customer” should be reachable through multiple contact means (email, phone not just in person). Also we should have backup contacts who we can talk to about the project.

[5] Sebokwiki, *“Risk Management SEBoK,”* Sebokwiki.org. [Online]. Available: http://sebokwiki.org/wiki/Risk_Management. [Accessed: 28- Oct- 2016].
[6] WebMD. *"What Are Your Odds of Getting the Flu?,"* www.webmd.com. [Online]. Available: http://www.webmd.com/cold-and-flu/flu-statistics. [Accessed: 25- Oct- 2016].

GitHub pages fail or become unavailable.	0.0006(0.001*0.6)	We have an alternate web server available for use. However, it would take about 24 hours to update the DNS records (we are hosting the website on GitHub). We will also have locally stored copies of the code.
Change of requirements for the software.	0.3 (1*0.3)	All software written should be as modular as possible, allowing easy updating of code. Code should be commented to help maintain it and documentation should be extensive.
The system requirements are not properly identified/incorrect requirements.	0.06 (0.3 * 0.2)	We have spent a relatively large amount time defining our requirements: the ambiguities have been discussed with the "customer" and settled. A report of the requirements has been sent to the customer for a final check before being implemented. We believe that this will mitigate the risk. We have also made sure that there are no conflicting requirements in our requirements documentation.
The project may use software that has not been used by any of the team members before.	0.18(0.6*0.3)	We are implementing the project in Java which all of the team have previous experience with, this was chosen over our alternative of using C# and/or JavaScript with Unity, as we felt that the extra experience would benefit the team and reduce the risk of us failing the project. We also all have experience with the Java IDE, Eclipse, as opposed to the alternative software, Unity, which nobody had used before.

Poor productivity.	0.2 (1*0.2)	We have made sure that the first thing that we did on the project was teambuilding. This will hopefully allow us to function better as a team and to make us more motivated for the project.
Difficult to work over Christmas.	0.3(1*0.3)	We have made sure that we stay in contact over the Christmas break. We are using Google Hangouts to speak to each other, and we also have a TeamSpeak server available to us if it becomes necessary. Furthermore, we are also using GitHub and Google Drive, so it is not necessary to meet up.
Customer forgets what they asked for and does not like the finished project.	0.2 (1*0.2)	We have made sure to record our interview with the customer as to protect ourselves against this possibility.