Risk Document

Human Resources

Thomas Falicone, Zachary DiPascale, Jarrod Cummings, Ephraim Gbadebo

**Risk Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Probability | # of Weeks | Exposure |
| Team Member Switch/Dropout | 5% | 1.5 | 0.075 |
| Switching Technologies | 20% | 3 | 0.6 |
| Schedule Interferences | 75% | 1 | 0.75 |
| Change in Requirements | 50% | 2 | 1 |
| Incompatible Integration w/ other Silos | 25% | 2 | 0.5 |
| Version Control Errors | 25% | 0.2 | 0.05 |
| Network Failure | 10% | 0.15 | 0.015 |
| Database Errors | 15% | 1.5 | 0.225 |
| AWS Crashes | 5% | 0.5 | .025 |
| Other Silo’s API Stops Working/Is not functional | 20% | 0.5 | 1 |

**Risk Descriptions & Mitigation Strategies**

1. **A** **team member switching teams** **or dropping out** in the middle of the semester is unlikely, but can still happen due to whatever personal circumstance may arise and/or rolling the corresponding daily risk. There would need to be an effort to assist the new team member in catching up with details concerning the project implementation, documentation and what would be required of them at that point. In the case of a dropout, the remaining work would need to be deliberated and redistributed to the other team members. Depending upon how much work is remaining, this could be a relatively minor to significant impact on development progress. One way to soften the impact of this when it happens would be for all members to frequently update each other of their own progress so the team is prepared for how much work will need to be picked up.
2. **Switching technologies** in the middle of production is relatively unlikely but is within the realm of possibility. If we discover that major requirements call for something that is beyond the capability of our current technologies then a significant amount of time would have to be wasted re-implementing functionality to accommodate the new requirements. Otherwise, rolling the risk dice and coming up with a change of technology (such as the server) could also cause this issue. We can circumvent this possibility by doing extensive research beforehand and making sure the technologies we use can be used to fully accomplish the task given.
3. Our team members are very likely to have various **schedule interferences** that include work for other classes and personal obligations. This will result in meetings that some members will miss and less time for development. Therefore, it is important that team members manage their time efficiently between classes and notify each other beforehand whenever they have conflicts to mitigate the effects of this risk as much as possible.
4. A **change in requirements** due to the resultant dice roll or new API requirements from other silos are likely to arise in the middle of development. Depending on the requirements, it could change other parts of the implementation, forcing us to re-code parts to accommodate the new functionality needed. There are few ways of preemptively handling this other than adhering to Agile principles as much as possible and making sure that we frequently communicate with each other and the project manager to keep them informed about the progress that is made with the implementing the changes and notifying them of any problems that may arise. Furthermore implementing a change board of sort will enable us to keep on top of any changes that are coming down the pipeline.
5. There is at least some chance that we will face some amount of difficulties in **integrating with the other silos**. While we are all part of the same organization, we are also broken into separate teams and focusing on completing our own individual tasks. There is the possibility of miscommunications between silos on what exactly was required by who and from whom, or there could be difficulties getting two different technology stacks to interact with each other as expected. This risk can be mitigated by communicating between the silos, both before and when problems arise, so that they can be addressed promptly. There is also the hope that as time goes on, the designs and implementations of each silo will become more set in stone. Once the silos are properly working with each other there will be less need to worry about a drastic design change that will break the integration.
6. When multiple people are working on the same project, there is a small chance that **version control errors** may happen. This would be the result of an unexpected or careless merging of multiple versions of code. If this were to happen, in the worse case scenario, depending upon the gap between the most recent push and the previous one, we would need to take time to fix whatever errors were generated by the merged push and ensure that a clean revision of the code is the most recent one on the branch before proceeding with development. We can decrease the likelihood of this happening by always keeping a separate copy of the modifications made while working and always making sure to pull before pushing any new code.
7. Considering that the majority of us working on the project utilize the campus-wide network to communicate and work asynchronously with version control, **network errors** are a possibility we must consider. Albeit, not often occuring, they can happen at random and one to two full days of work can be lost due to loss of access to version control. This risk can be particularly fatal if it happens towards the end of development time with one or two days remaining before the deadline. It is important that team members and silos have alternate ways of communication through methods like phone or text and that the most important components of the system have been implemented a fair amount of time before the deadline so that this doesn’t affect the project too negatively if crunch time is necessary before the deadline.
8. When utilizing a database to keep track of models for data manipulation, the possibility of **database-specific errors** is something that must be considered. This usually goes hand-in-hand with the requirements changing, or possibly, a poor base design for the system during the planning phase. Insufficient model representation, poor naming conventions and error-prone communication between the domain layer and data layer will negatively impact the validity of the entire system and make API integration with other silos impossible. Making sure to document and communicate all requirements and changes to the model, and properly running unit tests on the system will ensure that all information is being represented accurately and that errors are found as early as possible.
9. In the case where **AWS crashes** our main concern would be the relational database that hosts both our dev and production databases, and elastic beanstalk that enables our program to function. If this does occur the obvious alternative to fix this would be to revert back to hosting on the RIT Windows VM. The main concerns with that issue would be compatibility. For example, getting .Net and mysql relational database on a Windows 2008 Server. As such, constantly updating the VM with the necessary software ahead of time would prevent any extended periods of time where the application is not available.
10. Since HR is reliant on API endpoints from Sales and Customer Support, it is necessary to consider the risk that their **endpoints may go down** for a short period of time. The impact of such an event will result in Salaries not being calculated and Reviews not being seeable in an employee profile. For this reason, coming up with a mitigation strategy will be important to ensure the continual success of the project. Mitigation strategies would heavily rely on our silo being able to safely handle failure situations. For example, a possible solution for dealing with this issue will rely heavily on alerting the user, via the UI, about the outage of functionality. This will give some indication about the status of the API in a digestible manner.