SE 329 - Risk Analysis Report

Michael Parker, Jacob Caithamer, Alex Dana, and Cuong Nguyen

1) Personnel shortfalls

This is a common risk for almost any generic project, not just those in the software industry. In any small business or project team, there is a always a chance that a key team member could leave, or there simply aren't enough individuals to handle the large amount of work that needs to be done in a given time period.

For our project, we have a very small team, which means that each member has a lot of responsibilities and expectations to uphold. If any team member leaves, then we've just lost 25% of our project, and it would be difficult to get a new person trained quickly.

There are several things that we could do to mitigate this risk. One option would be to focus more heavily on team building and morale building exercises and activities. Doing so would emphasize that we appreciate our team members as well as the business value they bring to the table. Another thing we could do is focus on cross-training employees, so there is never one "key" employee that is responsible for holding everything together. Cross-training will allow us to make sure that there are several employees responsible for every working part, making it more difficult for an employee to leave us high and dry.

2) Unrealistic schedules and budgets

This kind of risk is usually present when there is a fair amount of miscommunication between upper management and the software developers. Sometimes, a new feature that management thinks will be simple to implement may take longer than expected, and sometimes software developers might plan for more or less time than is actually needed for a particular task.

In our case, this risk would be present if we attempted to create a rigid schedule for the entire development of our project with strict expectations for what features need to be done in certain timeframes. It will be hard for us to estimate these kinds of timeframes for large projects such as this, which is what might create this risk.

This kind of risk can be avoided by good software development methodology as well as better communication between the developers and management, which could be handled by the project's product owner. Agile software development will help by promoting adaptive planning and continuous improvement to the project, which makes the task of scheduling a lot easier. It

is much more realistic to plan a long project iteratively as progress is made, as opposed to drawing out one single plan to be used for the rest of the product's lifecycle.

3) Developing the wrong software functions

This risk affects the viability and the worth of the finished product when it first hits the marketplace. If the product does not do what the users need it to do, then there is a good chance that sales will be low and the product will become shelfware.

This risk definitely applies to our project, because none of us are very familiar with the Hotel business, and thus, we don't know what kind of functionality that our users would expect, or what kind of functionality that we could implement to set ourselves apart from any competing solutions.

We can mitigate this risk by conducting thorough user surveys throughout our development process. If we adhere to Agile, then we will have a continuous feedback loop that we can use to show prototype examples to users and assess their feedback. This will ensure that the team stays on track with developing and expanding upon the features that users are looking for, and nothing else. Another way that we can work to reduce this risk is to release early documentation and user's manuals and provide a mechanism to leave feedback. This can help prospective users get an early glimpse of the product as well as give them a way to voice any concerns they might have regarding functionality.

4) Developing the wrong user interface

Like #3, this risk also affects the viability of the finished product, as well as its usability. While the needed functionality might be present, it matters very little if the software benefits are not obtained, especially if users are reluctant to use it. If a bad user interface is developed initially, it will require extensive rework farther down the road, so it's important to try and mitigate this risk as early as possible.

This risk applies to most software projects, ours included. Since we are a small team, it's quite easy to create a bad user interface and continue to expand upon it instead of reworking it, because as the developers, all of the software's functionality is obvious to us, where it might not be so obvious to the end user.

We should be able to manage this kind of risk using the same techniques that we would use for #3. Agile software development will allow us to use a continuous feedback loop where we can assess the quality of the user interface before it gets to a point where it's too late to refactor it. Another thing that would help us is to utilize user characterization, which will assist us with creating Agile stories that pertain to certain users or groups of users in terms of functionality, style, and workload.

5) Gold plating

Gold plating refers to adding additional features to the project that only marginally useful or not useful at all. This kind of risk typically happens when management are out of touch with the end users that they're trying to sell to. A feature that management thinks is a "must-have" may be completely ignored by end users. Software developers may also think of innovative ideas that simply don't work or don't make sense to the end users, which usually results to a significant increase of the project's technical debt.

This risk applies to our project because, as a potentially new product on the market, we will be focused on adding innovative features which can help set us apart from similar software that has been around for a bit longer. Sometimes these innovative features may take a while to implement and still offer little to no benefit to the end user.

We can work to reduce this risk by performing a cost-benefit analysis for each feature that we plan to add to the project. We can try and plan for the amount of time we think a feature will take to implement, and compare that to the amount of perceived benefit that users will see, which we can assess by demoing prototypes to potential end users. We can also make sure that our early stages of development focus on completing a minimum viable product, which has the features that are required to make the product a worthwhile addition to the market. Once this MVP is finished, the team can focus on innovations without having to worry about sacrificing essential functionality.

6) Continuous stream of requirement changes

This kind of risk is common in projects which have few or no counterparts in that are presently on the market. Management may decide to target a completely new focus group, or add a group of users to account for when implementing certain functionality. In any case, this usually means that part of the project needs to be re-written with the new requirements in mind, which is definitely a drain on business resources.

This risk may not be so apparent in our project as the other risks presented in this document. Given the fact that we are a small team, and we have a clear idea of which direction we need to move to get our project on the market, it is unlikely that there will be even a few requirements changes, let alone a continuous stream of them.

This kind of risk can be mitigated by adhering to some of the principles of Agile software development. By working on the project in separate iterations, the development team can verify the software requirements or receive new ones between pre-defined sprints. This will allow the

development team to not lose focus of their goal during the current spring, while allowing management to make a small amount of changes to the requirements when necessary.

7) Shortfalls in externally performed tasks

This risk is typically present in projects that depend on an API that is offered by a different company. Perhaps the API lacks some key functionality that your project needs to function correctly, or a particular endpoint takes too long to respond, which leads to unreasonable wait times in your software. In any case, this kind of risk is difficult to manage, as it pertains to entities that are typically out of your control.

This risk does not really apply to our project, as our project doesn't deal with externally performed tasks, at least for the time being. Perhaps this risk may be introduced if we decide to add some functionality to the project that depends on an API controlled by another company.

This risk can be minimized by performing pre-award audits, which will allow us to evaluate a prospective external party's financial, operational, and organizational capabilities. Using this information, we will be able to assess the contractor's eligibility for being included on this project.

8) Shortfalls in externally furnished components

This risk is present in large projects that require external contractors to work on different parts of the project. When dealing with external contractors, both parties need to be aware of the project requirements and expectations, as well as the deadlines for different features or components that are supplied by the contractors, or there is a good chance that this risk will be present.

This risk does not really apply to our project, since we will not be depending on contractors to help us build different parts of the project.

This kind of risk can be minimized by monitoring the receipt and integration of the externally furnished components. After comparing the projected project dates with the completion dates of each component, we can issue alerts if there are any big differences, or modify the project requirements / schedule. Like the previous risk, this risk can also be mitigated by performing pre-award audits to assess a particular contractor's ability to meet the specifications of a required component on time.

9) Real-time performance shortfalls

This risk is present in any project that needs to take some time to perform some sort of algorithm or computations before returning results to the end-user. This could have any number

of effects on your product. For example, if your business logic takes too long to process, users may be reluctant to use your product at all in favor of something similar that works more quickly. A more significant example would be something like the anti-lock braking system on a car, which needs to work as quickly as possible to be effective.

This risk could apply to our project if we rely on complicated and time-consuming business logic. At worst, end users would opt not to use our product in favor of a pre-existing one that offers the same functionality but at a better speed. In order for our product to be successful on the market, it's important to reduce these kinds of shortfalls whenever possible.

This risk can be minimized by consistently performing simulations and benchmarks to make sure that our product meets or surpasses the expectations created by other products on the market. By running these kinds of tests on a consistent basis, it will be easy to catch smaller problems and fix them before they turn into larger, more time-consuming problems.

10) Straining computer science capabilities

While not a very common risk, this risk is still possible in some projects that are taken on by larger software companies whose business model revolves around providing technological innovations to end users. For example, Apple is always looking for ways to make computer hardware smaller tin order to produce more aesthetically appealing products and fit more hardware in a laptop shell than previously thought possible.

Given the fact that there are already several products on the market that do something very similar to the planned features for our project, the chance that we will be straining any computer science capabilities is very, very small.

Mitigating this kind of risk is not very easy, because it may be next to impossible to overcome some of the existing thresholds of computer science. One thing that can be done is a thorough cost-benefit analysis to determine if it is really worth implementing some functionality that is simply not possible at the present moment. Sometimes it's not worth investing lots of money into a feature that may not be very sought-after by end users.