SE423 Software Project Management

Activity 2: Classic Mistakes: Giga-Quote Case Study

Group Members:

SHOUG FAWAZ ABDULLAH ALOMRAN

ROSE SAEED RAKAN AL RAKAN

YARA FARIS FAIHAN ALBUGAMI

HATOON ABDULLAH

1. Case Study Overview

The Giga-Quote project was a program suggested by Mike, the technical lead for Giga Safe, which is a medical insurance company. Originally, the purpose of the project was to offer automation for generating medical insurance quotes via a desktop application. Later on, the project was enhanced by the executive committee when it added the function of uploading daily generated quotes to the head office so that the most current sales leads could be accessed online. Bill, Mike's boss, told the idea to the executive committee and had successfully gotten the funding approval. This usually would be good news, however the original proposal had no ability to communicate with the head office as it was a standalone desktop application. Initially, the plan was for the entire project to take about 12 months, however, the executive committee wanted to transfer the field quotes into the main server automatically. At the same time that the project scope was expanded, the executive committee reduced the development schedule from approximately 12 months to only 6 months. In response to the shortened deadline, Mike began exploring different ways to reduce the development schedule and complete the project within six months. Although Mike attempted to shorten the schedule, the project plan did not adequately account for potential risks and significant requirements changes that could occur during development. As development continued, requirements changes, technical problems, and increasing numbers of defects resulted in repeated schedule delays. The repeated schedule pressure and extensive overtime also negatively affected team morale and productivity. Ultimately, Giga-Quote was completed in 13 months rather than the planned 6 months and required 98 staff-months of effort compared with the planned 36 staff-months.

2. Classic Mistakes Analysis

In this section, we will analyze the different mistakes that Mike and his team made from different dimensions, such as people, process, product, and technology mistakes.

2.1 People Mistakes

Mistake 1: Wishful Thinking

Definition:

It's a kind of optimism that is hoping for something to work while having no reasonable or logical basis for thinking it will. It undermines meaningful planning and could be the root of more software problems.

Evidence:

After Bill reveals the project deadline, he tells Mike: "I know it will be a challenge, but you're creative and I think you can pull it off…"

Impact on Project:

This, per the definition, has to be one of the most crucial reasons why the project ended up being delayed the way it was. This kind of thinking also led to employees rushing, which only brought up more issues and bugs.

Mistake 2: Unrealistic Expectations

Definition:

Defined as one of the most common causes of friction between developers and their managers, it is setting unrealistic expectations regarding the project.

Evidence:

From the start of this case, it seemed that many stakeholders involved have had unrealistic expectations. The strongest evidence to that is the deadline they choose, Mike had

made it clear that this project could need around a year. Instead, the executive committee gave a strict time of 6 months. They expected a fully functional, bug-free, and fully tested system in that time, which is not realistic or feasible.

Impact on Project:

Similar to the wishful thinking mistake, this caused a panic in selecting employees and even resulted in some tension. For example, one of the team members, Chip, was not the easiest to work with. He wouldn't allow Jill or Tomas to get near his code, even though, with the due date approaching, team work is much needed now more than ever.

Mistake 3: Adding people to a late project

Definition:

Although it is not the most vital mistake, it still is an issue. Being already late to the deadline, adding a member later in this project was not a good choice. In fact, adding people later can take more productivity away from those who have been in the team from the beginning.

Evidence:

Regardless of how much of a team player Chip was, he had actually accepted a contract to a different company. This meant that Mike needed to get someone else to help out instead, so he got Kip.

Impact on Project:

After analyzing Chip's code, Kip found deep conceptual flaws that meant it would never work. Kip had to unfortunately redesign and reimplement a major part in the communications link, which ultimately added to the already delayed project.

2.2 Process Mistakes

Mistake 1: Shortchanged Upstream Activities (#20)

Definition:

Shortchanged upstream activities occur when a project under time pressure cuts or reduces activities that should be performed before or during development, particularly requirements analysis, architecture, and design. These activities may not directly produce code, but skipping them can create significant rework later.

Evidence from the case:

The Classic Mistakes reference explicitly identifies the Giga-Quote case as an example of this mistake: a design hack was used for the bar-chart report instead of quality design work. The reference states that the hack eventually had to be discarded and the higher-quality work had to be completed anyway. The case itself shows the consequence of this shortcut. When testing required the bar chart to be placed on the right side of the report, Tomas explained that he would have to rewrite the report from scratch and estimated that this would take at least 10 days.

Impact on the project:

The shortcut created additional rework instead of saving time. The team had to spend valuable development time replacing the earlier implementation while already under severe schedule pressure. This increased the workload and contributed to further project delay.

Mistake 2: Shortchanged Quality Assurance (#22)

Definition:

Shortchanged quality assurance occurs when a project cuts corners on important QA activities, such as design reviews, code reviews, test planning, and thorough testing, in an attempt to meet an aggressive schedule.

Evidence from the case:

As Giga-Quote fell behind, Mike and Bill extended the schedule to December 5 but required the developers to work 12-hour days. The revised schedule required testing and field-agent training to take place concurrently. Stacy warned that this would not give QA enough time to test the software, but Bill overruled her.

The consequence was severe. After the feature-complete build was handed to testing, the testing team identified more than 200 defects in two days, including 23 Severity-1 (Must Fix) errors. Stacy concluded that the software could not realistically be ready for release by January 1.

Impact on the project:

Reducing QA allowed serious defects to remain undiscovered until very late in development. Fixing them required substantial additional effort and delayed the project further. Therefore, the attempt to gain schedule time by reducing QA ultimately created more rework, more delay, and greater pressure on the development team.

Mistake 3: Insufficient Management Controls (#23)

Definition:

Insufficient management controls occur when a project lacks effective mechanisms for monitoring progress, identifying schedule problems, and providing timely warnings when the project begins to fall behind. The reference specifically states that Giga-Quote had few such controls and that the controls that existed at the beginning were abandoned when problems arose.

Evidence from the case:

The project lacked reliable information about its actual progress. By February, Claire stated that she had not received a reliable schedule estimate from Bill for months, that the project was already more than three months late, and that the bug statistics showed the team was not closing the gap.

After stopping the project, Charles reviewed the project's bug statistics and recommended focusing on error-prone modules, introducing design and code reviews for bug fixes, and having the team work regular hours so management could obtain an accurate measure of the effort being spent and the effort still needed to finish.

Impact on the project:

The lack of effective management controls prevented Giga Safe from recognizing and responding to the project's true condition early enough. Management continued operating with unreliable schedule expectations instead of taking corrective action based on objective progress and defect information. By the time effective controls were introduced, the project was already severely delayed and overloaded with defects.

2.3 Product Mistakes

Mistake 1: Feature Creep (#29)

Definition:

Feature creep occurs when additional requirements or features are introduced after a project has already been planned or development has begun.

Evidence from Giga-Quote:

Giga-Quote's original proposal was for a standalone desktop application. However, the executive committee later added a significant requirement for field quotes to be automatically transferred to the company's main server. During development, the actuarial department also introduced an entirely new rate structure that required information about exercise habits, smoking habits, recreational activities, and other factors.

Impact on the Project:

These changes significantly increased the team's workload. The new requirements forced the team to modify the input dialogs, database design, database access, and communications objects. This resulted in substantial rework and contributed to further schedule delays.

Mistake 2: Push-Me, Pull-Me Negotiation (#31)

Definition:

Push-Me, Pull-Me Negotiation occurs when management approves a schedule extension because a project is behind schedule, but then introduces additional tasks or demands that make the revised schedule difficult or unrealistic to achieve.

Evidence from Giga-Quote:

During the entire process, the management kept changing the deadlines but failed to revise the requirements for the project as well. Since the deadline of November was not met, the new one was set for December 5; however, this time the developers had to work 12-hour days. The testing and training of agents were supposed to take place simultaneously in order to be able to adhere to the deadline of January 1. Despite the new issues coming up, the management kept setting more strict deadlines.

Impact on the Project:

Rather than having the schedule reflect the true amount of work left to do, these choices kept adding pressure to a project that was already behind schedule. The people working on the project got tired out, less work got done, and there was less time available for testing.

Mistake 3: Requirements Gold-Plating (#28)

Definition:

Gold plating of requirements involves having requirements for a project which involve unnecessary complexity or effort that is not required to achieve the primary objective of the product.

Evidence from Giga-Quote:

The Giga-Quote reports included specific presentation requirements that added complexity to the product. The sales group required the quote summary to contain both text and a bar chart on the same page. Later, testing required the bar chart to appear specifically on the right side so that the report would match existing materials and the format familiar to the company's agents.

Impact on the Project:

These detailed presentation requirements created additional development effort. The report-generation tool could not easily satisfy the required layout, which initially led Tomas to implement a temporary workaround. When the bar chart was required to appear on the right side, Tomas determined that the report would have to be rewritten from scratch using low-level formatting and graphics code, requiring at least 10 additional days of work.

2.4 Technology Mistakes

Mistake 1: Silver- Bullet Syndrome (#33)

Definition:

The term "Silver- bullet syndrome" implies to relying on new tools, solutions, and technologies and too much dependency on them when solving huge project problems, for example shortening the development schedule.

Evidence from the case:

Carl and another technical lead had assumed that technology could have shortened the development time, they chose to use object- oriented design C++ and Mike agreed to it. As well as a, report generation tool believing it would reduce development time in half.

Impact on the project:

The team concluded that C++ didn't save them from changing requirements for instance input dialogs, database, database access, and communication objects but using these technologies gave no improvement. This contributed to delays and made it harder for the team to meet the deadline.

Mistake 2: Overestimated Savings from New Methods and Tools (#34)

Definition:

This error happened when the team anticipated new methods to considerably decrease time more than it reasonably can. Forgetting to account for learning time, limitations, and risks. This case clarifies that new tools usually don't produce huge productivity improvements because the benefits can be lessened to by learning curves and unforeseen risks.

Evidence from the case:

The team thought that using C++ could save 1 or 2 months, by utilizing the report-generation tool they believed that they could cut report development time in half. They utilized these unanticipated savings to shorten their estimated timeline for around 12 months to meet the 6 month deadline.

Impact on the project:

The predicted time saving didn't happen. The report generator later was not capable of producing the necessary report containing both text and bar chart on the same page. Thomas

constructed a short-term solution. Later, for modifying the bar charts position they had to rewrite the whole report, which Thomas predicted would take at least 10 days.

3. Recommended Project Approach

People Mistake Approach:

Since 2 of the people related mistakes are more at an organizational level, and therefore will be dealt with according to the company policies and structure. We will tackle the third mistake which is about adding a member to a late project. Contrary to popular belief, adding more people to an already behind the schedule project could actually do the opposite of the intended goal. This mistake could end up taking away productivity from the team, which in result slows down the operation. To avoid this, we can either opt for boosting team morale or recruiting more than the estimated team members from the beginning. In a project like this, where the deadline is already short and not fitting for the project size, having many employees from the start can be helpful. Even though Bill wasn't always helpful, he made sure that Mike knew that he could recruit anyone he needed to achieve the project delivery on time. Many could argue that having a lot of employees from the start could become messy, however, the point is to have skilled members at least follow and be aware of the project progress in case of any additional help that could be required later on. This could have fixed the issue when Chip left and Kip had to take over, resulting in a major change in the system.

Process Mistake Approach:

If our group had been responsible for Giga-Quote, we would have established a structured development process that protects essential design, quality assurance, and project-control activities even when the schedule becomes difficult.

First, to avoid Shortchanged Upstream Activities (#20), we would complete sufficient requirements analysis, architecture, and design before major implementation begins. Important design decisions would be reviewed before development proceeds, and the schedule would

include adequate time for these activities. This would reduce the likelihood of relying on temporary design shortcuts that later require significant rework, as happened with the report-generation problem in Giga-Quote.

Second, to prevent Shortchanged Quality Assurance (#22), QA would be integrated throughout development rather than being left until the end. We would conduct design reviews, code reviews, test planning, and incremental testing for completed components. Critical defects would be tracked and resolved before release, and the project would not sacrifice necessary testing simply to meet an unrealistic deadline. This would help prevent the large number of late defects experienced by Giga-Quote.

Third, to address Insufficient Management Controls (#23), we would establish measurable milestones and regularly compare actual progress, remaining work, and defect levels against the plan. A formal status review would be conducted at each milestone. If progress significantly deviated from the plan, management would reassess the schedule, remaining work, scope, and release date instead of relying mainly on overtime or increasingly aggressive deadlines. This would provide early warning and allow corrective action before problems become critical.

Overall, this approach would prioritize proper design, continuous QA, and objective project monitoring. It would reduce rework, detect defects earlier, and ensure that management decisions are based on actual project progress rather than optimistic assumptions.

Product Mistake Approach:

To address the product-related mistakes in Giga-Quote, our group would establish stronger requirements and scope control throughout the project. Any new requirement introduced after development begins would first be evaluated based on its impact on the schedule, cost,

design, and testing effort before being approved. This would help control feature creep and prevent major changes, such as the new rate structure, from being introduced without adjusting the project plan. If a schedule extension becomes necessary, the revised deadline should reflect the actual remaining workload rather than adding further pressure or overlapping major activities such as testing and training. Finally, detailed requirements that introduce significant development effort should be evaluated based on their actual value to the product. These approaches would reduce unnecessary rework, control the project scope, and create a more realistic balance between product requirements and the available development time.

Technological Mistake Approach:

Giga- quote had deepened heavily on new technologies and methods relying on C++ could save and cut 1 or 2 months and in addition report generators could reduce development time. In our group, we would test and assess the technologies with prototypes before fully using them. This would help us deeply understand limitations, risks, and learning time and allows the team to make a prediction of a practical timeline. For instance, testing earlier would have displayed how the report generator couldn't create a report with correct requirements and layout results; this could have helped avoid writing the whole report from scratch. These approaches could help stop silver-bullet syndrome and overestimated saving for new methods and tools.

4. Quality and Cost Tradeoffs

4.1 Quality Tradeoffs

The approach would be to prioritize software quality regardless, if more time was needed during development. Investing more time on design and requirements would enable the team to spot problems beforehand, and produce a more dependable system instead of rushing the solutions to meet the deadline. Early on and continuous testing could also help identify defects

before becoming more complex and costly to fix. This is what occurred during Giga- Quote: “In two days, the testing group had identified more than 200 defects in the Giga- Quote program including 23 that were classified as ‘Severity 1’- ‘Must Fix’-errors” all this occurred after the team had assumed the development was complete.

Although prioritizing these quality activities would require additional time and cost during development, our group would accept this tradeoff because identifying defects earlier would reduce the risk of significantly more expensive rework and schedule delays later in the project.

4.2 Cost Tradeoffs

Hiring people from the start could definitely have its financial burden, but we believe that the final results payoff is more worth it. In this case study, time was a big focus and determinant for how things went. Due to the time limit given, members of the team were expected to rush which ended up costing more since that birthed more bugs in the system. The late change of one of the developers resulted in a huge financial loss since there was someone expected to replace him. This, however, wasn’t the cause of the cost increase, rather the fact that all his work needed redesigning and reimplementation was the true cause for it. Important design decisions could have also cut time short, thus preventing the errors that came up from the teams rush. One of the project managers, Claire, actually took action that should have been done from the start to cut all the unnecessary cost of fixing everything. She demanded a report of the realistic time schedule of all the bugs and issues that needed fixing to achieve the correct program. Once it was reviewed and approved, they started seeing a decrease in the amount of bugs, which ended up saving cost since there was no need for people to fix them in such a small time crunch.

5. Conclusion

From this case study, it is possible to conclude how a range of errors in the people, process, product and technology areas of a software project may affect its success. In the Giga-Quote project, the reasons for frequent delays, overload and high number of defects included unrealistic planning, changing requirements, poor quality assurance, management issues and over-reliance on technology. These errors might be avoided by more careful planning of the project, requirements control, quality assurance, realistic planning and effective management decisions. Thus, the whole Giga-Quote case is an example that illustrates how trying to fasten a software project without taking into account its true scope, requirements for quality, risks and available resources may lead to increased time and efforts needed to develop the project.