

Copyright
by
Vedhapriya Raman
2017

The Report Committee for Vedhapriya Raman
Certifies that this is the approved version of the following report:

**A Dashboard-based Approach for Efficient Requirements Change
Management**

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Suzanne Barber

Sarfraz Khurshid

**A Dashboard-based Approach for Efficient Requirements Change
Management**

by

Vedhapriya Raman, B.Tech.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2017

Abstract

A Dashboard-based Approach for Efficient Requirements Change Management

Vedhapriya Raman, M.S.E.

The University of Texas at Austin, 2017

Supervisor: Suzanne Barber

Requirements gathering and documentation are important first steps for a successful software engineering project. The documented requirements act as a guideline for design and development of software products. Requirements also represent customer expectations for the end product. Since these documented requirements serve important purposes for many stakeholders, managing requirement changes effectively plays a major role in the overall success of any project. Changes in requirements are very common in software engineering and can occur during any phase of software development lifecycle (SDLC). Though the impact of requirement changes differs depending on the SDLC phase in which it occurred, there is almost always a setback that happens in terms of the project timeline. This scenario is common in projects that follow both Agile/Scrum methodology and ones that follow the more traditional Waterfall model. In this report, I will first present two case studies of how requirement changes impacted the timelines of two projects (one following Agile/Scrum methodology and another following Waterfall

methodology). In the second part of this report, I will propose and design a user-friendly dashboard, which could be used to speed up the delays caused by changing requirements.

Table of Contents

List of Figures	vii
Introduction	1
Background	4
Case study	8
Case study 1	8
Organizational structure	8
Software development methodology	9
Change in requirements	10
What could have been different	10
Case study 2	11
Organizational structure	11
Software development methodology	12
Change in requirements	13
What could have been different	13
Summary	14
Proposed dashboard	15
Design specifications	15
User interface	16
Key Advantages	21
Future work	25
Conclusion	27
References	28

List of Figures

Figure 1: Organizational structure chart of company A for project A.....	9
Figure 2: Organizational structure chart of company B for project B	12
Figure 3: Home page of proposed requirements dashboard	16

Introduction

Requirements gathering is the first and critically important phase of software development lifecycle (SDLC). Usually after the inception of a software development project, an organization's executives decide on the budget to be allocated for the project and the time within which the project is to be completed based on its importance for the organization. After this initial executive meeting, the project idea is passed down to an appropriate software development manager and the SDLC begins. At the end of SDLC, the software product becomes ready for being delivered to the customer.

Organizations strive to achieve customer satisfaction by delivering projects on time, within budget and according to customer expectations. A major challenge faced by organizations in achieving the above-mentioned goals is requirement changes. More than 70% of software projects are delivered late due to change in requirements that happened during the SDLC. Is it often impractical to avoid requirement changes after the initial requirements phase of SDLC. Especially for projects that have a long timeline, changes are bound to happen to customer expectations, which in turn change the initial requirements. Hence, many organizations have a change management process that dictates how requirement changes will be handled.

There are many shortcomings associated with existing change management processes and tools. Change management processes are usually lengthy and time consuming and in most cases, few of the stakeholders are left unaware of the changes happening to the project. Many organizations do not have a requirements engineer or change manager to take care of requirement changes. These changes are handled by the development team who are not aware of the best practices in requirements engineering. They also have project development deadlines and hence are unable to allocate sufficient

time for managing requirement changes. There are some web based project management tools that could be used for initial requirements gathering and for the other phases of SDLC but not many of such tools have effective change management options.

In the first part of my report, I present two actual case studies of software projects that were impacted by requirement changes. One of these projects followed the waterfall model of software development and the other one followed the more modern agile development practice. The case studies show how the organization was structured, how requirements were gathered and show how changes in requirements were handled. Finally, the case studies show how customer satisfaction and project timelines were affected by the requirement changes.

After analyzing these two case studies, I identified the problems associated with requirements changes. Specifically, I took into account the following factors:

1. Nature of changes
2. Phase of SDLC during which the changes happened
3. How the organization was structured
4. Any changes associated with respective requirements

In the second part of my report, I present a user interface design of a dashboard to manage requirement changes efficiently. This dashboard is a web-based collaborative environment that is not only a space to document requirements, but also provides means to efficiently capture requirement changes and showcase the changes to all the stakeholders involved. The major functionality of this dashboard includes requirements documentation, stakeholder accountability, break-down of each requirement to compositional tasks, effective tracking of these compositional tasks until completion, ability to change requirements to the level of individual tasks impacted by change,

prediction of time required to accommodate changes based on tasks impacted and requirement change approval and notifications.

I conclude the report with suggestions of additional functionality that could be added to the dashboard in order to personalize it for each project. There is also more work to be done to effectively predict the time delay that a requirement change would cause based on prior experiences in the project.

Background

Software development lifecycle (SDLC) refers to the various phases that a software development project goes through from inception to completion [1]. It usually starts with requirements phase. During the requirements phase of SDLC, various stakeholders like the customer, software development team including the manager, testers meet and come up with the requirements document. This requirements document usually lists the requirements along with their dependencies and priority. Once the requirement document has been approved by various stakeholders, the next phase of SDLC, software design and development begins. During this phase, the requirements captured during the previous SDLC phase acts as a guideline. After development of the software product, the next phase of SDLC, testing is done. Testers usually write test cases to determine if the requirements are met. Based on feedback from the testers, changes are made to the software product in order to meet customer requirements.

In the waterfall model of software development [1], the steps of SDLC happen sequentially. Once each step is completed, the next step is initiated. Since there is usually no going back to previous steps, implementing a waterfall software development model requires meticulous planning and strong documentation. The obvious disadvantage of this method is that there is no room for change in requirements to happen directly. If any change happens, then it takes the project team back to the first phase of SDLC and all the steps have to be repeated. This sets the project back in time based on how impactful the impending change is.

The agile method of software development [1] was introduced as a solution to the disadvantages caused by waterfall method. This method follows an incremental approach instead of a sequential approach. The requirements are broken down into small tasks and

collections of requirements are packaged as modules. Team members work on these individual modules in weekly or monthly sprints. More work can be done by working parallel on independent modules. At the end of each sprint, the end results are evaluated and the teams are assigned new modules. Many agile teams meet daily to understand what each team member is working on and to tackle any obstacle the team members might have. This method is obviously more flexible in that it allows for change after initial planning. During any sprint, the module priorities can be changed and additional requirements can be evaluated and added. However, a major disadvantage of this approach is that it needs meticulous tracking of the sprints. Otherwise, projects can take on too many changes and evolve to an entirely different product from the product that was planned initially. Improper tracking can also lead to chaos in the project.

Irrespective of the SDLC methodology being followed, it has been observed that requirement changes always cause a delay in project completion. The impact that requirement changes have on project timeline depends on the severity of the change or the number of requirements impacted and the phase of SDLC in which the change occurs. Following are some common scenarios when requirement changes happen.

1. Poorly defined requirements in requirement gathering: This is one of the most common causes of change in requirements later on in the SDLC. If there is no designated requirements engineer, usually team members lacking requirements engineering training such as developers or marketing team members are held responsible for defining and documenting requirements. This often leads to poorly define requirements. Another cause could be that some of the stakeholders could be missing in the requirements gathering meeting and hence requirements impacting their groups might need modification in the future. Poorly defined requirements could be averted by

having a designated requirements engineer in the meeting and planning the requirement gathering sessions in order to take into account the inputs from all impacted stakeholders.

2. Dependencies: Another leading cause of requirement changes is dependencies. For example, if the project is dependent on a set of tools and technologies, and if few of the updated versions of the tools do not work with existing versions of other tools, then this would create changes in multiple requirements dependent on the tools impacted.
3. Changing needs of customer: Almost all projects are required to consider the changing needs of the customer and how it impacts the initially documented requirements. This is especially the case for projects on a long timeline, during the course of which their customer needs might change.

There are many software tools currently available that assist in project management tasks. Some examples are Microsoft Project and Project Kickstart. These tools help in task scheduling, resource management, project budget planning, etc. However, these tools are designed for the use of project managers to keep track of project timelines, budgets and other areas of project management. They offer a much broader perspective of projects as against being specifically designed for managing requirements.

This project presents a tool which addresses this problem by being designed specifically for the use of requirements engineers to document requirements and requirement changes. Requirement changes are inevitable in both Waterfall and Agile development models and hence it would be immensely helpful to have a tool which could act as a requirement logging tool and document requirements and their constituent tasks. It would be all the more beneficial if the tool could act as a platform for collaboration

between all the project stakeholders and send customized notifications whenever documented requirements are modified or task schedule slippage happens.

Case study

The following case studies demonstrate how changing requirements impact project timelines and ultimately, the customer satisfaction for two real-time projects. Each case study details about the company's organizational structure, the software development methodology followed by them specifically explaining how requirements were maintained, how and when change in requirements happened, impact of requirement changes and what could have been done differently.

CASE STUDY 1

Project A was an internal research project. The project plan was to create an internal tool to help a section of the company's employees monitor their applications remotely. Thus, the company was the customer, its employees were the end users who would use the application developed by this project. Project A was planned as a six month long project initially including the time for initial requirements gathering and research.

Organizational structure

Project A had a team size of six members reporting to a project manager. The team included a team leader and five team members working from the same office location. The project manager's (PM) base office location was the same as the team members but the PM also used to travel to other office locations frequently. He would work remotely during his official trips and was available over phone and email. As mentioned earlier, this project had no direct external customers. End users were also employees of the same company. Figure 1 shows the organizational structure of company A specifically for project A. Of the five team members, three were software developers and two were testers.

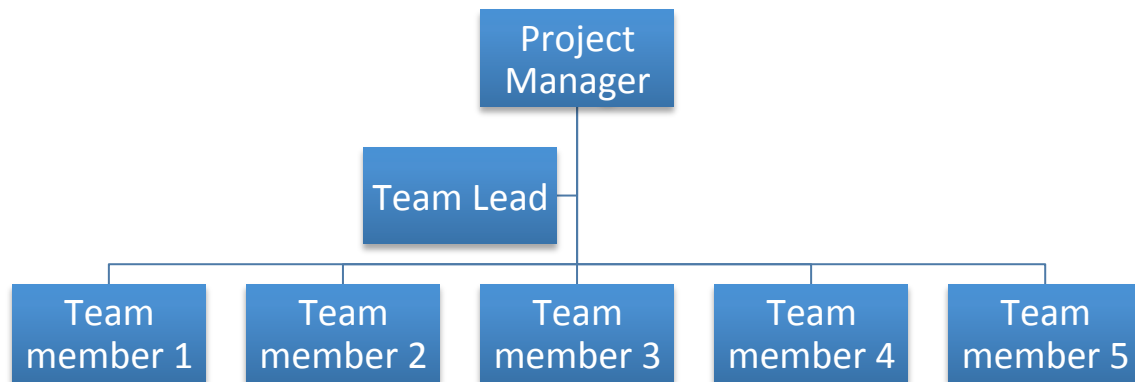


Figure 1: Organizational structure chart of company A for project A

Software development methodology

Project A followed the agile development lifecycle and held a stand-up meeting every day to keep track of their work. Also, there was no designated requirements engineer on the team. Two requirements meeting were held at the beginning of the project with the end users and the project team. However, there was no formal business requirements document created. The team started working based on their understanding of the requirements. Though the team consisted of highly experienced Software Engineers, the team had a tough time designing and developing the tool due to the lack of a clear and documented requirements. The only instances of documentation that happened throughout this project were the application design documents created by the development team and the unit test plan created by the testing team.

Change in requirements

During the two requirement gathering sessions that were held at the beginning of the project, some of the end users were absent. As a result, throughout the project lifecycle, those end users kept changing the initial requirements which caused further confusion and lead to constant change in the design and architecture of the tool. Adding to the challenge was the fact that the project team had no concrete requirements document to start with so they had nothing to go back and reference in times of changing requirements. Ultimately the project could not be completed in 6 months as originally planned. Only a prototype version of the tool was released at the end of six months, which recieved a lot of negative feedback from the end users.

What could have been different

Following are some of the key shortcomings of project A. Had these processes been done differently, the project might have succeeded.

1. There was no designated requirements engineer on the project team. The project team consisting of developers and testers were given the sole responsibility of maintaining requirements. Hence, there was no formal documentation of requirements. There was no way to compare to baseline requirements when requirements were changed. Documenting and maintaining requirements is the first step in order to make a project successful. The team would have greatly benefited by having a designated requirements engineer on the team.
2. Some of the key end users were absent for the initial requirements gathering session. This lead to partial and ignored requirements for the project team to begin with. Clearly, in order to have their expectations for the tool met, all key

end users should have been present in at least one among the two requirement gathering sessions.

CASE STUDY 2

Company B is an IT services company and project B was an ongoing maintenance and support project for one of its clients. Company B and its client are in a geographically distributed setting and their locations fall in different time zones. As part of the maintenance work, company B made enhancements to the legacy applications of the client. In addition to the maintenance work, the company also provides 24X7 production support for the same applications.

Organizational structure

Project B had a team size of five members reporting to a client manager. The project team included a team leader and four team members working from the same office location. Only the client manager worked from the client location. Since the client was geographically separated from the project team, they met the team virtually every month in order to go over the tasks that the team was working on and discuss and prioritize the upcoming work to be done. The same project team provided production support for the client applications. The client would raise tickets for any issues that arose in their application and the project team would take care of the tickets using break fixes or code fixes on a daily basis. Apart from regular support, the team also provided weekend and month end support on rotational basis. There were no deemed developers or testers in this project. All four team members and the team leader assumed tasks on an ad-hoc basis. The tasks included development, testing and production support fixes. Figure 2 shows the organizational structure of company B specifically for project B.

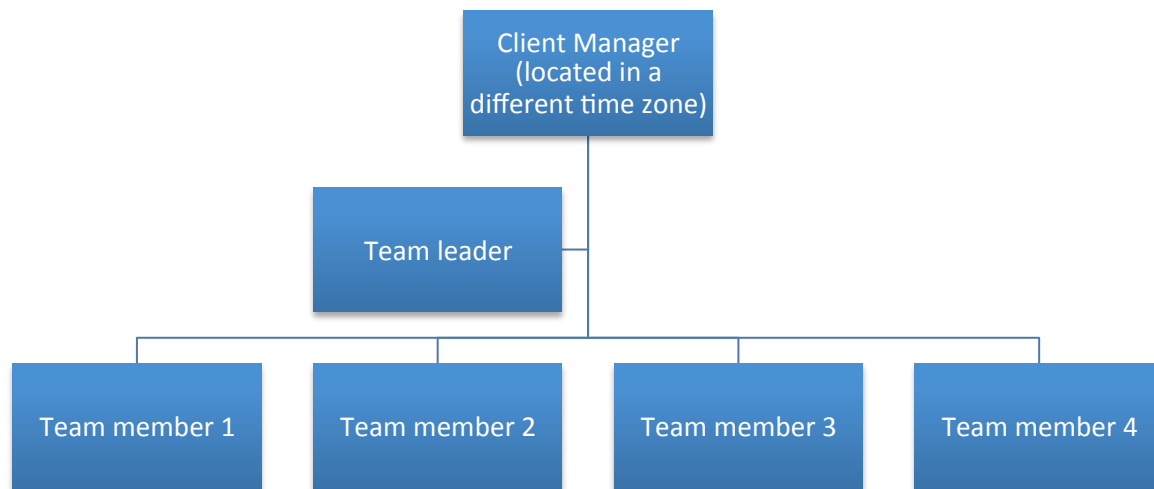


Figure 2: Organizational structure chart of company B for project B

Software development methodology

Project B followed the waterfall model for software development. This project also did not have a designated requirements engineer. The client manager would set up a project planning meeting with clients monthly to set forth a plan for project requirements to be completed that month. They also addressed spillovers from the previous month during that meeting. Based on the priorities assigned by the clients, the project team would start working on requirements. In most cases, there was no formal requirements specification document. The client manager would send the project team an email describing what feature in a specific application needs to be fixed/tweaked.

With respect to the production support tasks, these were largely ad-hoc and the tickets usually had very little information about the ongoing issue. The team usually

contacted the appropriate customers on phone to understand what the issue was and offer fixes accordingly. There was rarely any documentation of these production support tasks.

Change in requirements

In cases when the client changed requirements for maintenance tasks, there was no formal change management process and the team was just expected to give in to the needs of the client. This not only delayed the delivery of the modules impacted by the changed requirements, but also other modules which were of lower priority for that month. This regularly caused task spillovers to consecutive months resulting in low customer satisfaction.

The situation was made even worse for production support tasks. Since there were less details about requirements on the customer tickets and no documentation was created by the project team for resolved tickets, if any ticket was reopened with changed/ added requirements, it always caused confusion. Additionally, since the requirements were primarily given to a single team member over phone, there was no way other team members could know the requirements and this process increased dependencies in the project team.

What could have been different

Following are some of the key shortcomings of project B. Had these processes been done differently, the project might have been successful.

1. There was no designated requirements engineer on the project team. The project team was given the sole responsibility of maintaining requirements. Hence, there was no formal documentation of requirements. In particular, there was no way to know the actual requirements for production support

tickets. The team too, would have greatly benefited by having a designated requirements engineer on the team.

2. Project team was following waterfall development lifecycle for maintenance tasks and hence suffered from the inherent disadvantages of this method. It would have benefitted the team to meet more often and break down requirements into group of small tasks.
3. There were a lot of dependencies in this team. The work done by each of the team members was not transparent to each other and hence, the team members were alone in resolving their obstacles. Everyone was keen on completing only the tasks assigned to them and hence the team lacked team spirit to come together to resolve issues. Following an agile development process would have been greatly beneficial since each team member would then know exactly what the other team members are working on. It is also the best way to resolve obstacles and move forward as a team.

SUMMARY

In summary, the case studies presented above illustrated how important it is to manage requirements and requirement changes for the successful completion of a project. One of the major deficiencies of both Project A and Project B is that there was no designated requirements engineer on either teams. The projects also lacked requirements documentation and a collaborative platform for the stakeholders to communicate. Had there been better collaboration amongst project teams and proper documentation of the requirements, both projects could have been completed successfully on time

Proposed dashboard

Considering the shortcomings of the previous projects, this project designed a dashboard that could be used to effectively document, track and maintain requirements and any changes in requirements in a transparent manner. Following are the design specifications for this interactive dashboard.

DESIGN SPECIFICATIONS

1. Ability to add project team members and their profiles. All project team members must have access to view information on the dashboard.
2. Ability to maintain requirements log according to release numbers of project.
3. Each requirement in the requirements log should be expandable to its constituent tasks and information about priority of each task and number of days required to complete each task.
4. Each task should be assigned to a project team member.
5. Each task should show percentage of work completed in the form of a slide bar.
6. Ability to capture dependencies between requirements (and in turn the constituent tasks).
7. A graphical representation of project's timeline goals and whether the team is on schedule.
8. Ability to set up email notifications in case of missed schedule, requirement changes, etc.
9. In case of change in requirements, ability to identify the requirements and specifically, tasks impacted.

10. Ability to predict the delay any requirement change will cause based on the project's historical data.

USER INTERFACE

Following the design specifications outlined above, the user interfaces for the proposed dashboard was designed. The balsamiq desktop tool was used to design these mock-ups. Figure 3 shows the home page of this proposed dashboard. The dashboard home displays important aspects of the project in a consolidated format.

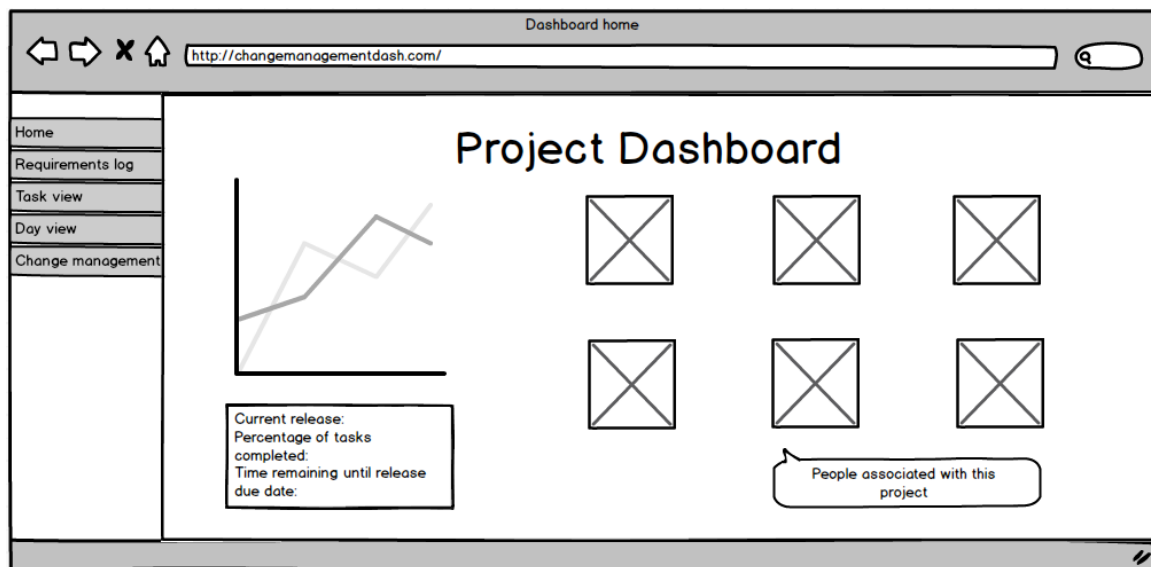


Figure 3: Home page of proposed requirements dashboard

The home page has a line graph showing the expected rate of task completion to meet the project deadline as planned against the real-time task completion rate. This graph can be used to get an insight of how the project team is doing with respect to the project timeline goals. There is also a list of project statistics displayed below this line graph detailing the percentage of tasks completed, time remaining until release due date,

etc. This data could be used to send email notifications to project team in case of lapse in schedule. The remainder of the home page displays the photos of the project team and customer representatives. Each of these images is a clickable link to that person's profile showing their role in this project. Additionally, this page could be configured with ways to set up email notifications for various events like missed schedule, change in requirements, etc.

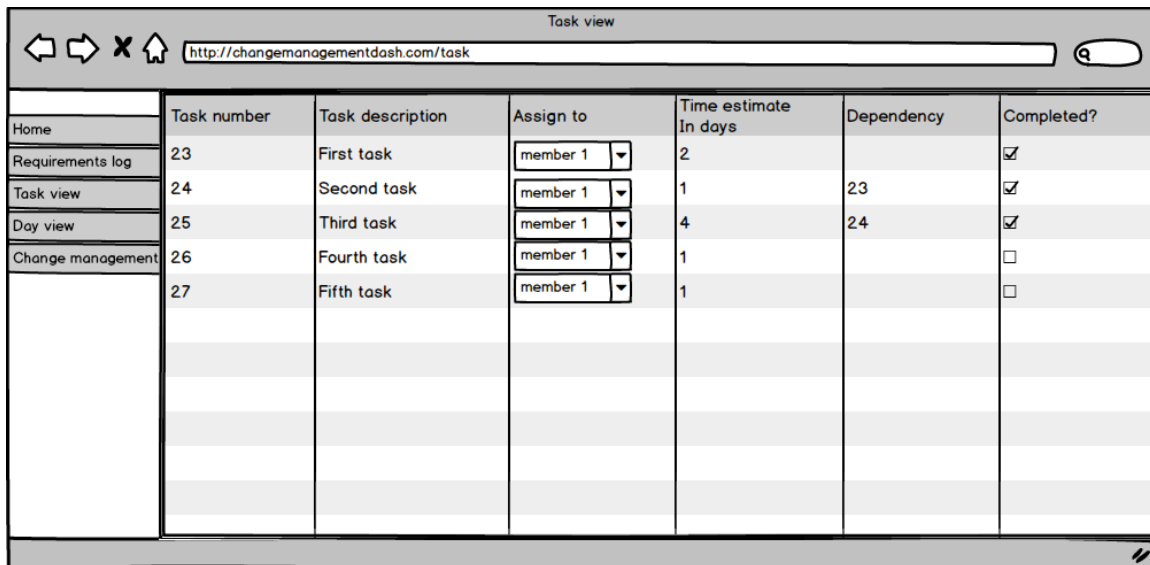
The requirements log page maintains all the existing requirements for a project release in a tabular format. Figure 4 shows the mock-up of requirements log page. The columns of the table include requirement number (to be used for mapping dependencies), description, number of constituent tasks, estimated time to complete all tasks associated with this requirement in days, priority and an indicator of completion.

	Requirement #	Requirement description	# of tasks	Time estimate In days	Priority	Completed?
Home	1	First requirement	7	7	High	<input checked="" type="checkbox"/>
Requirements log	2	Second requirement	10	10	High	<input checked="" type="checkbox"/>
Task view	3	Third requirement	2	2	Medium	<input checked="" type="checkbox"/>
Day view	4	Fourth requirement	4	5	Low	<input type="checkbox"/>
Change management	5	Fifth requirement	8	5	Low	<input type="checkbox"/>

Figure 4: Requirements log page

The requirement description column can be expanded to show each requirement's constituent tasks. Finally, there is a checkbox to indicate whether each requirement is complete or not. This checkbox can be marked to show completion when all the constituent tasks for a requirement are completed. This checkbox will be reset every time a requirement change is submitted.

Each task in the requirements log page is in the form of a clickable link, clicking which opens task view page. Figure 5 shows the task view page showing tasks constituting a particular requirement.



The screenshot shows a web browser window titled "Task view" with the URL "http://changemanagementdash.com/task". On the left is a sidebar menu with options: Home, Requirements log, Task view (selected), Day view, and Change management. The main content area displays a table with the following data:

Task number	Task description	Assign to	Time estimate In days	Dependency	Completed?
23	First task	member 1	2		<input checked="" type="checkbox"/>
24	Second task	member 1	1	23	<input checked="" type="checkbox"/>
25	Third task	member 1	4	24	<input checked="" type="checkbox"/>
26	Fourth task	member 1	1		<input type="checkbox"/>
27	Fifth task	member 1	1		<input type="checkbox"/>

Figure 5: Task view page

The task view page is also in a tabular format with columns comprising of task number (for mapping dependencies), task description, assignee, estimated time to complete task in days, dependency and an indicator of completion. The project team members' names appear on the dropdown and an assignee for each task can be chosen.

The dependency indicates the requirement numbers and task numbers on which any particular tasks depends.

This dashboard was designed to work well with agile software development methodology. As outlined earlier in this report, in agile methodology, project team meets every day to discuss their individual tasks and the overall progress of the project. In order to work with this model, a day view page was designed, which can be used during the daily stand-up meetings. Figure 6 shows a sample of this day view page.

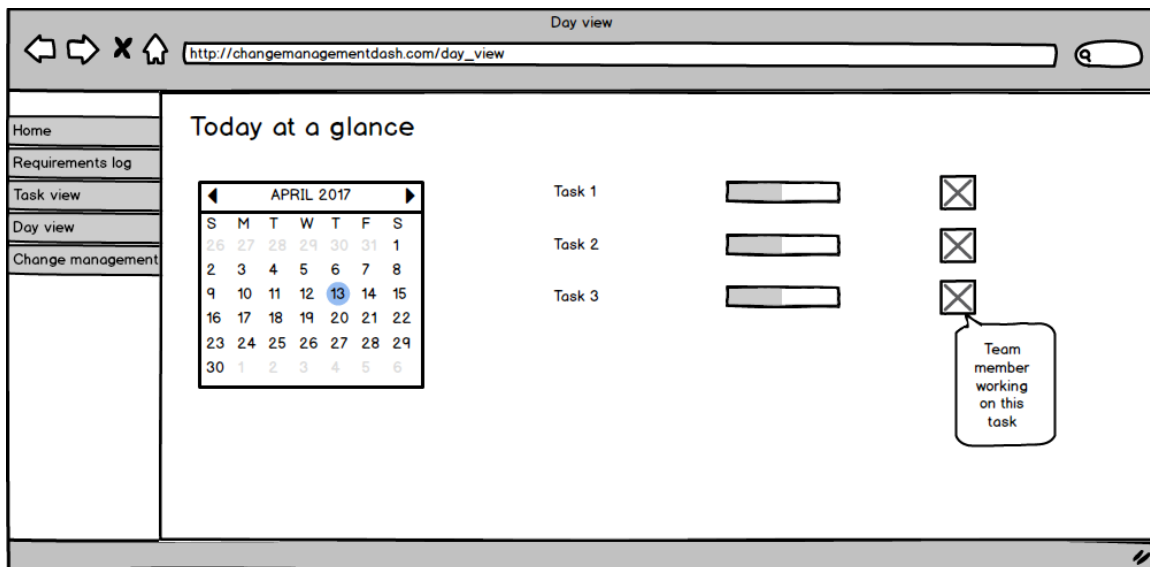


Figure 6: Day view page

The day view page includes a calendar view showing today's date and the progress of various tasks. Each project team member is assigned a task for the day. The tasks are displayed along with photos of the assignees, so that everyone attending the stand-up meeting has an idea of what every other person is working on. There is also a progress bar for each task showing percentage of task completed. During stand-up

meeting, teams can also discuss potential obstacles and their impact on specific tasks. Tasks can be marked as completed by the team members as needed.

Finally, a change management page was designed which can be used to change, add or delete existing requirements. Figure 7 shows the change management page designed for the proposed dashboard.

Figure 7: Change management page

In order to change an existing requirement, first the constituent tasks are to be considered for identification of tasks impacted by this change. The requirement number and impacted task numbers are to be entered followed by a brief description of changes to the requirement. Clicking on the estimate button would return an alert with a precomputed delay that this change would cause. On accepting the alert, the requirement change would be submitted along with an email notification to the project team. In order to add a new requirement, all constituent tasks should be added using the “add task” button. The information required to add a new task includes a task number (for mapping

dependencies), task description, assignee, estimated time to complete task in days and dependency with other requirements/tasks.

In order to estimate the delay caused by a change in requirement, the dashboard could include a backend API. The API can be designed to analyze historical change data for the project along with the information about tasks impacted in order to come up with a delay estimate. The change request would then go through approvals of certain members of the project team in order to be added to the release. The release due date would be adjusted accordingly. The permissions for various team members can be controlled to allow edit access to just a few members of the project team, while the dashboard would be viewable by all members of the project team.

KEY ADVANTAGES

Since the proposed dashboard was designed taking into consideration the case studies presented earlier in the report, dashboard has specific functionality to address the major shortcomings of those projects. Following are some of the key advantages of using this dashboard.

1. The dashboard acts as a common platform for all stakeholders of the project. Each stakeholder can be added to the dashboard along with their role in the project. Hence, the possibilities of any of the stakeholders staying out of loop from the project plans and actions are avoided.
2. The ability to set up email notifications is another great way for the project stakeholders to stay in loop. A default notification is triggered to all stakeholders whenever a requirement change request is submitted. This ensures that everyone in the project team is made aware of changes happening

to requirements, so that if any team member has concerns regarding the change, they can take it up with rest of the project team.

3. The dashboard homepage also contains line graphs showing planned and actual project timelines. Apart from the graphs, this page also displays project statistics like percentage of tasks completed, number of days until project due date, etc. These numbers and the line graphs are updated in real time based on status updates from project team members. A look at this graph and the project statistics is enough for the project team members and other stakeholders to understand if they are on schedule. Automated email notifications are triggered whenever there is schedule slippage in the project, so that the issues causing schedule slippage can be addressed immediately.
4. The project requirements are clearly documented in the requirements log page. The documented requirements act as the guidelines for the project development and testing teams. The requirements are broken down into constituent tasks so that the dependencies can be addressed effectively. In addition to this, each requirement has attributes such as priority and number of days required to complete all tasks pertaining to a requirement. All these details reduce confusion and are essential to the project team to plan and execute the project effectively.
5. There is also a task view page that displays the constituent tasks of any given requirement. There is ability to document dependencies and assign tasks to any of the project team members. The assignees can mark the tasks as completed as needed. Once all the constituent tasks of a requirement are completed, the requirement itself is marked as complete. The project statistics in the homepage are updated according to the percentage of requirements

marked as complete. All these functionalities help ensure schedule tracking. This dashboard is designed to help manage requirements, change in requirements and work in progress in an agile development setting.

6. Another way to track work in progress using this dashboard is the day view page. This page displays any given day at a glance. It shows all the tasks that are currently in progress and the person working on each individual task. The task assignees have the ability to update the progress of their tasks using the slider next to their tasks. This functionality would be especially useful for daily stand up meetings in an agile development model. During stand up meetings, the project team can pull up the day view page and will be immediately shown the list of tasks in progress and the assignees. Each team member can take turns to go over their assigned task and move the task progress slider to indicate what portion of their task has been completed. This would also enable the team to analyze and resolve any obstacles any team member might face in order to finish a task.
7. Finally, this dashboard also provides ability to add new requirements or change existing requirements. In order to add a new requirement, a stakeholder has to enter requirement description in detail, break down the requirement into its constituent tasks, and specify attributes like number of days required to complete tasks, dependencies with other requirements/tasks, task assignees, priority of requirement, etc. Having to add all these details ensures that any new requirement that gets added after project initiation is thoroughly scrutinized. Also, an automated notification is triggered to all stakeholders and project team members informing them of this new additional requirement. The project timeline is also adjusted accordingly. This makes

sure that the project team has enough time to complete all requirements and is not forced to complete the added requirements within the original project deadline.

8. Changes to requirements also go through a similar structured process to ensure there are no disruptions to the project schedule. In order to submit a change request, a stakeholder must enter the details of the requirement that needs to be changed, like dependencies, description of change, etc. The dashboard would then analyze historical data and come up with a time estimate to account for this change. This estimate can be changed later for individual tasks as needed. The dashboard only allows one change at a time so that the requirement dependencies can be addressed better. Hence, the change management page of this dashboard prevents potential disruptions to the project and keeps the project team up-to-date with the project requirements.

Future work

Potential future work could involve improvements to the existing functionality of the requirements dashboard. In addition to day view page, it would be nice to be able to see tasks in progress in weekly and monthly views. Since each task has a set number of days estimated, the tasks could span across days according to their estimated length. Being able to see weekly and monthly views would help the project team plan their work around holidays and their schedule. It would also be beneficial to have the ability to tag resources used to complete individual tasks to the tasks page. For example, if a particular programming language and database were used to fulfill a requirement, then the names of that programming language and database could be tagged to all the tasks constituting that requirement. This would be helpful when the team reflects about the work they did for a given project. They would be able to get insights like which tools and technologies were easier to use and which were causing delays in task completion based on historical data. It would also be a great way to document resources needed to complete a given task. The project team members can refer to an old project to find out exactly what resources they need if they encounter similar tasks in the future.

Another area in which future work could potentially be done is the implementation of the features presented in the dashboard design. In addition to creating a web-based application with a user interface similar to the features presented in the design, it would be beneficial to develop a mobile application with similar features. The mobile application could be connected to the same back-end web services as the web-based application so that these applications can share data and updates. Using a mobile application would be much easier for the project team. The notifications for schedule

slippage, requirement changes, addition and deletion of requirements, etc. would have much more reach if sent from a mobile application. The combination of web and mobile based applications can be utilized effectively by project team members who work remotely.

Conclusion

In this report, I have analyzed how requirement changes impacts project timelines. I first presented case studies of two different projects and explained how they were negatively impacted by change in requirements. One of the projects followed the waterfall model of software development and the other followed an agile methodology. However, both projects lacked a requirements engineer and had a multitude of other issues impacting the project schedule.

In the second part of my report, I proposed a requirements dashboard which was designed based on the shortcomings of the two projects presented in the case studies. This dashboard was designed as a common platform for all stakeholders of a project to collaborate and has the ability to document requirements, requirement changes and the impact of those changes. The dashboard has many advantages like being transparent to all stakeholders and project team members, having the ability to set up email notifications, sending automated email notifications in case of requirement changes, etc. This dashboard would provide realistic delay estimates in case of requirement changes and would be immensely beneficial to project teams.

References

1. Kute S., Thorat S. A Review on Various Software Development Life Cycle(SDLC) Models, IJRCCT, 2014
2. A.Aurum, C.Wohlin. Engineering & Managing Software Requirements. Berlin, London: Springer, 2005.
3. Leffingwell, Dean. Managing Software Requirements: Use Case Approach- 2003. Boston: Addison-Wesley, 2003.
4. Yana Selioukova. Business Process Modeling in Software Requirement Engineering. 2002.
5. Peter Oriogun. Innovations in Teaching and Learning in Information and Computer Sciences. University of North London, UK.
6. Jain Wang. Object – Oriented Analysis Methodology. University Of Missouri, St.Louis, 2001.
7. Ilia Bider. Choosing Approach to Business Process Modeling-Practical Perspective. Research Report, Ibis soft, 2002.
8. E. Smith. Re-Engineering a Trash/Recycling Collection Vehicle Based on Challenging Customer Requirements. Clemson University College of Engineering and Science, Clemson, South Carolina, 2010.
9. E. Hull, K. Jackson, J. Dick. Requirements Engineering, 3rd ed., London, UK: Springer, 2011.
10. J. Terninko, Step-by-step QFD: Customer-driven Product Design, Boca Raton, Florida: St. Lucie, 1997.

11. Linda Westfall. Software Requirements Engineering; What, Why, Who, When, & How.
12. Roland Kaschek, Heinrich C Mayer. Characteristics of Object Oriented Modeling Methods. University of Klagenfurt.
13. Dragan, Milicev. Model Driven Development with executable UML. Indianapolis, IN: Wrox/Wiley, 2009.
14. Jones, Caper. The Economics of Software Maintenance In The Twenty First Century, 2015.
15. Anton, Annie L., and Colin Potts. The Use of Goals to Surface Requirements for Evolving Systems, 2015.