Senior Thesis

Change Management

WITS Shira Neuberger Summer 2020 Etti Sherman August 11, 2020 Tehilla Schwartz

[Change 4](#_Toc48074125)

[Background 4](#_Toc48074126)

[Introduction 5](#_Toc48074127)

[Larger Picture of Change in Software Systems 5](#_Toc48074128)

[Reasons to Change 7](#_Toc48074130)

[Lehman’s Laws 8](#_Toc48074131)

[Types of change 9](#_Toc48074132)

[Adaptive vs. Transformational 9](#_Toc48074133)

[Built-to-Change Systems 10](#_Toc48074134)

[Moving Parts 10](#_Toc48074135)

[Metadata 10](#_Toc48074136)

[Change Management Process 11](#_Toc48074137)

[Five General Steps of Change Management 11](#_Toc48074138)

[Change Management Models 14](#_Toc48074139)

[Olsen’s Change Management Model 15](#_Toc48074140)

[V-Like Change Management Model 16](#_Toc48074141)

[Ince’s Change Management Model 17](#_Toc48074142)

[The AMES Change Management Model 18](#_Toc48074143)

[Spiral-Like Change Management Model 18](#_Toc48074144)

[Overview of Change Management Models 20](#_Toc48074145)

[Urgency of change 21](#_Toc48074146)

[Emergency Change 21](#_Toc48074147)

[Expedited Change 21](#_Toc48074148)

[Latent Change 21](#_Toc48074149)

[Normal Change 21](#_Toc48074150)

[Minimal Impact Change 22](#_Toc48074151)

[Approval Process Depending on Urgency Level 22](#_Toc48074152)

[Smooth changes 23](#_Toc48074153)

[Agile vs Waterfall 23](#_Toc48074154)

[Parts Error Explosion 24](#_Toc48074155)

[Push vs Pull 25](#_Toc48074156)

[Implementation of Changes 26](#_Toc48074157)

[Acquisition of Resources 26](#_Toc48074158)

[Implementing the Change 27](#_Toc48074159)

[Version Control 27](#_Toc48074160)

[Importance of Version Control 27](#_Toc48074161)

[Centralized Version Control vs. Distributed Version Control 28](#_Toc48074162)

[General function of version control 30](#_Toc48074163)

[The Evolution of Version Control Systems 30](#_Toc48074164)

[Git Commands 32](#_Toc48074165)

[Creating a Central Repository on Git Hub 33](#_Toc48074166)

[Branching and Merging 35](#_Toc48074167)

[Caution Regarding Change 38](#_Toc48074168)

[Example of Change in a Database 38](#_Toc48074169)

[Testing 42](#_Toc48074170)

[Regular Testing vs Regression Testing 42](#_Toc48074171)

[Formal Regression Testing vs Informal Regression Testing 42](#_Toc48074172)

[Reasons for Regression Testing 43](#_Toc48074173)

[Implementing Regression Testing 44](#_Toc48074174)

[Change Management for Large Business Critical Purchased Software systems 44](#_Toc48074175)

[Overview 44](#_Toc48074176)

[Up Front Preparation 45](#_Toc48074177)

[Purchased Software Components and Customizations 45](#_Toc48074178)

[Communication with the User Community 46](#_Toc48074179)

[Evaluation of Additional Changes 46](#_Toc48074180)

[Updating the User Community 47](#_Toc48074181)

[Interactions with Other Systems and Companies 47](#_Toc48074182)

[Making a Go or No-Go Decision for Upgrading to the New Version 47](#_Toc48074183)

[Evaluating Costs of a No-Go Decision 48](#_Toc48074184)

[Develop Back Out Method in Case of Failure 48](#_Toc48074185)

[Prepare for Failure of the New Version 49](#_Toc48074186)

[Updated Migration Plan 49](#_Toc48074187)

[Test New Version 50](#_Toc48074188)

[Test Software Interactions with the System Being Changed 51](#_Toc48074189)

[Phase New Version into Production 51](#_Toc48074190)

[Support for the new version 52](#_Toc48074191)

[People Consideration 52](#_Toc48074192)

[Conclusion 53](#_Toc48074193)

# Change

## Background

Change Management is a vital process for implementing changes and updates within software development. Changes are necessary because software programs are constantly changing and increasing in functionalities. Changes must be made to fix bugs or errors and to update and improve written code. The process of the change will depend on the nature of its priority - taking into consideration whether it is regular, emergency, or major. Once it is determined that a change must be made, one must evaluate if necessary resources are available. This is contingent on having a team to carry it out, having the proper funding, and having software to handle it. After ascertaining that the change is worthwhile, it can now be implemented. The team assigned to carry out this change will need a platform to collaborate, this is generally done using version control. There are centralized version control systems, which are more prevalent, and distributed version control systems, which are less common. The most common version control platform used today is Git, or in its visual representation - GitHub. Version control systems in general provide numerous functionalities; such as, synchronizing, branching, merging, and tracking changes made by each team member. Besides for new code that was written to implement the change, it is important to consider the effects the change will have on other pre-existing features of the system. Testing is a recurring step throughout the entire process to ensure that the change is producing the expected results and the change did not negatively affect any pre-existing features. Additionally, the system must be set up to accommodate potential failure, one way is to have access to backups of previous versions, making it possible to migrate back to the older version of the system if necessary. Besides for change of smaller systems, one must also consider the change management for large business critical purchased software systems. In this thesis we discuss an area of computer systems management - change management. We discuss the importance of change management, the difference between types of changes, and how to properly implement these changes, both small and large.

# Introduction

Change. The very word causes many people to shudder and turn away as humans crave stability. However, change is vital as it fuels our lives and betters our society. As Machiavelli wisely described change in his 16th century political treatise, The Prince, “There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of the new order of things[[1]](#endnote-1).” Although change causes uncertainty and instability, by embracing the knowledge that change is necessary and understanding why it is so vital to every system, one can focus on the progress and improvement being accomplished rather than the stress involved.

## Larger Picture of Change in Software Systems

When developing a software system, the developer must build the system with the knowledge that it will be changed in the future. When starting the development process, it is important to keep in mind both the initial cost of development and the long-term cost of the system. Many times, when creating a software system an organization will forget to evaluate, calculate, and plan the long-term cost of the system, leaving them in detrimental position when confronting change without proper planning. When designing a software system, an organization must look at both Figure 1.1[[2]](#endnote-2) and Figure 1.2[[3]](#endnote-3) to get the larger picture of software systems. Figure 1.1 depicts the initial cost and life cycle of a software system. However, Figure 1.2 more accurately depics the overall long-term cost and effot distribution of a software system. As can be seen on Figure 1.2, the bulk of the devlopment process and allocation of resources is used for maintenance – “small changes” to keep the system running effeciently.[[4]](#endnote-4) Many organizations conclude that more than 80% of their funds and employees are allocated for maintenance. When developing a new software system, one must keep both initial and long-term effort distribution and costs, in order to properly prepare their software system for long-term use. With proper planning and big picture outlook, it is not unlikely for a system to last more than twenty years. Interestingly, the longer a system is used, the percentage of time and effort spent on maintenance increases. This is simply because there are usually more change requests as the need to change increases as the system ages.[[5]](#endnote-5)

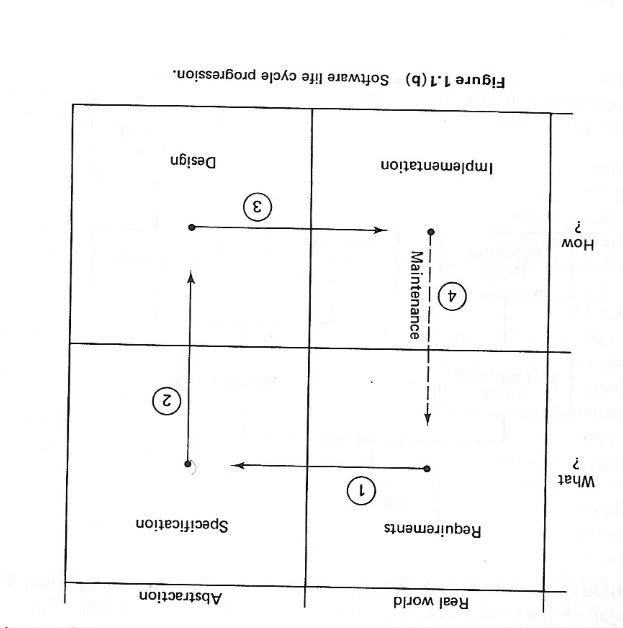


Figure 1.1 Software life cycle

## 

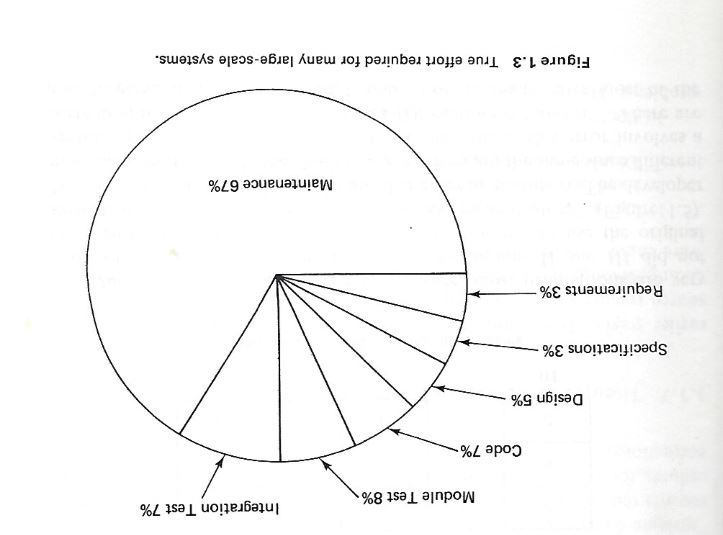


Figure 1.2 The effort required for many large-scale systems

## Reasons to Change

Change is a major component of software systems. Every business, whether big or small, must constantly be changing its software system. Maintenance is a type of change which is typically a small change but can have a significant impact. There are numerous reasons why companies are forced to change their software programs or databases on a regular basis. Some of these reasons may be because of the software itself and some reasons may be because of external factors.

Some examples of external factors resulting in a need for change, is an added or

expanded business line. Expanded business lines and services can occur because of internal growth as well as acquisition of another company. Additionally, companies must be aware of new or changed government regulations which will affect them. Furthermore, there are often changes in industry standards that require adjustments to be made in a system [[6]](#endnote-6). Because the line market is more popular than ever, competition is greater than ever before. This requires companies to constantly be advancing their system to keep up with the competition[[7]](#endnote-7). Other reasons for change include customer dissatisfaction and suggestions. This can range from simple complaints like “the website is a hard color to read” to more severe issues like “the system is slow”. Changes in technology, desire to reduce costs, and improving quality of software are all reasons why companies may choose to change their software system[[8]](#endnote-8). Bugs and errors found in programs would be an example of a required change because of the internal software system.

### Lehman’s Laws

Lehman’s Laws of Software Evolution include eight laws which changes should follow in order to be successful. As mentioned previously, business software systems must constantly be advancing and changing due to new requirements, changed regulations, and many more to ensure customer satisfaction. Because of this, the size and complexity of the software system will increase overtime, and the quality of the system will decline if the changes are not monitored properly[[9]](#endnote-9).

The first of Lehman’s law, Continuing Change of 1974, suggests that software systems must constantly be changing to ensure user satisfaction. This can be ensured by keeping up with the competition and improving the quality of the system.[[10]](#endnote-10)

Increasing Complexity, the second law invented in 1974, claims that the complexity of a system will continue to increase unless it is monitored and maintained properly[[11]](#endnote-11). Creg and Walsh state that the process of the business must be simplified to provide better quality service to the user community[[12]](#endnote-12).

The third law of 1974 states that a system evolution process must be self-regulating.

Conservation of Organizational Stability is the fourth law stating that a business should have an invariant work rate[[13]](#endnote-13).

The fifth of Lehman’s laws is Conservation of Familiarity, which claims that for a system to be successful, all associated with it must “maintain mastery of its content and behavior.” As stated previously, the system must constantly be updating and changing to ensure user satisfaction, known as Continuing Growth of 1980. In contrast to the sixth law, the Declining Quality of 1996 declares that software systems will decrease unless the changes are maintained. The eighth and final of Lehman’s law is the Feedback System of 1996 asserting that feedback must be taken seriously to successfully improve[[14]](#endnote-14). Lehman’s laws create a basis of the principles of change management.

## Types of change

### Adaptive vs. Transformational

When discussing change, one must discuss that variety of changes that may take place. There are two main categories - and most if not all changes fall under one or the other. The first change type is adaptive change, this includes changes that are small and more gradual. As can be seen from the “adaptive” keyword, adaptive change is primarily used to adapt your software or database to new requirements or technology. Most adaptive changes fall under the category of maintenance and usually take place as a form of updating a system that is already in place – these minor changes are commonly referred to as micro changes. [[15]](#endnote-15)

The second type of change is transformational change, these are changes that are much larger and dramatic. For example, when a company must revamp their entire system and relaunch their software or database, would be considered transformational change. Transformational changes are encountered less frequently, however when encountered they involve an abundance of work and funds. Some vendors may mislabel an adaptive change as a transformational change to excite their user community; therefore, a user should evaluate the change for themselves and verify that the change is truly transformational. [[16]](#endnote-16)

## Built-to-Change Systems

### Moving Parts

Since change, whether adaptive or transformational, is inevitable, one must create a system with built-in mechanisms that will facilitate smooth changes when they are necessary - primarily evaluating moving parts and metadata regarding those moving parts.

Moving parts include any programs, databases, and user interfaces that are involved in the systems functionality. Besides including the internal frame of a system, moving parts also includes any potential input and output that a system may encounter - generally data. When considering the potential input and output of a system, one must consider data that may be sent or received from within their organization and data that may be arriving from other companies. Many companies choose to begin closely following and documenting their moving parts from the time of system creation in preparation for change that will be encountered in the future of their system using metadata.[[17]](#endnote-17)

### Metadata

Metadata is the documentation and comments that a programmer includes within his programs, databases, and user interfaces to make his code readable and adaptable by himself and others. Metadata is so important for two main reasons. Firstly, a programmer may be coding and maintaining multiple systems simultaneously and through his metadata he is able to reacquaint himself with the system he is currently involved with and alter it efficiently. Secondly, developers almost always work as a team and they may not be the ones who are regularly updating the system that they originally coded as that task may have been given to another team member. Using metadata all team members are easily able to acquaint themselves with any system that they may be assigned to update - contributing to the agile environment of the organization. [[18]](#endnote-18)

Metadata not only includes documentation of changes that were made on a given program or database, it also includes a description of the data - what that particular data is used for, which table that data is stored in, how this data is stored, and relationships that exist between processes and data. Additionally, Metadata is especially necessary when dealing with relationships between processes and data, as all the relationships must be analyzed and considered before a change is implemented, as changes can impact not only the data but also existing queries and reports. In fact, many database administrators will create a matrix of relationships to follow all relationships in the database to be able to easily identify which programs are currently accessing which data. Although seemingly tedious, Metadata is necessary for facilitating proper system functioning and change management. A common problem that many companies encounter regarding documentation is that many developers will only concern themselves with implementing the change and leave very little time to include metadata and external documentation - both which are necessary for the system’s function.

## Change Management Process

### Five General Steps of Change Management

After closely analyzing the preparation of change that is embedded in the makeup of a system, one must understand the general steps involved in the change management process (Figure 2.1)xxi. The first step in the change management process is the preparation step, this includes the preparation that is specific for this change and not the groundwork that is embedded in the system. This includes understanding why this need is vital and explaining to the necessary authorities and team members why this change is so critical. After clearly understanding the need for the change to be implemented, step two involves creating a vision to plan for the change. This step includes evaluating, calculating, and planning the logistics that would be involved if and when the change would take place. The next step, step three, involves both step one and step two, the explanation of the importance of this change and the logistics of the change, being presented to the powers that be to decide if the change is to be implemented now, never, or in the future. This is also the time to go over change requirements for the specific change being discussed and envision the expected outcome. Assuming that it was decided to advance forward with this change, step four is to implement the change following the logistics that were evaluated and decided upon in step three. There are three main components of change implementation - the team to complete the necessary work, necessary funding, and necessary software. A change management team usually includes a change manager, who heads the change of a system when necessary, a change initiator, who initiated this particular change, and a change coordinator, who coordinates all small components of the larger picture. Lastly, there is a change implementer, who implements the change. Each of these roles can be filled by multiple team members as each role is generally too large to be completed alone. It is critical for the entire change management team that is completing the work to be motivated to implement the change for the change to meet success. Step five is to test the change appropriately and then after passing necessary testing, embedding the changes into the company’s existing system. Lastly, the change’s progress is reviewed, and the results are analyzed and documented for future reference.[[19]](#endnote-19)

The Change Management Framework that is described above and depicted below (Figure 2.1) is the general structure of a change to a system but this process may be altered when necessary to meet specific change requirements.

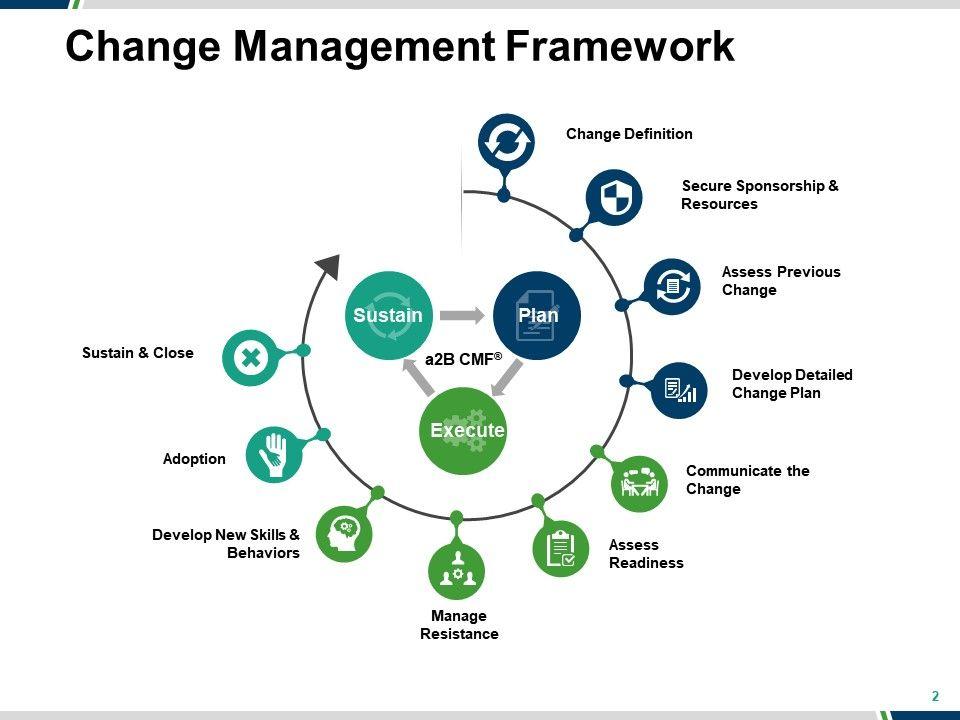
[[20]](#endnote-20)

Figure 2.1 Change Management Framework

### Change Management Models

The five general steps of the Change Management System outline the fundamental steps of any change; however, the steps may be altered based on the model that is being used to implement that change. There are five primary Change Management Models, and although the models do overlap, they each have key pointers that characterize that specific model’s agenda. Some organizations choose a specific change management model to follow but as an employee, one should know that although they should try to follow the organizations chosen model as much as possible, if necessary they can alter it to fit their specific needs and requirements.

#### Olsen’s Change Management Model

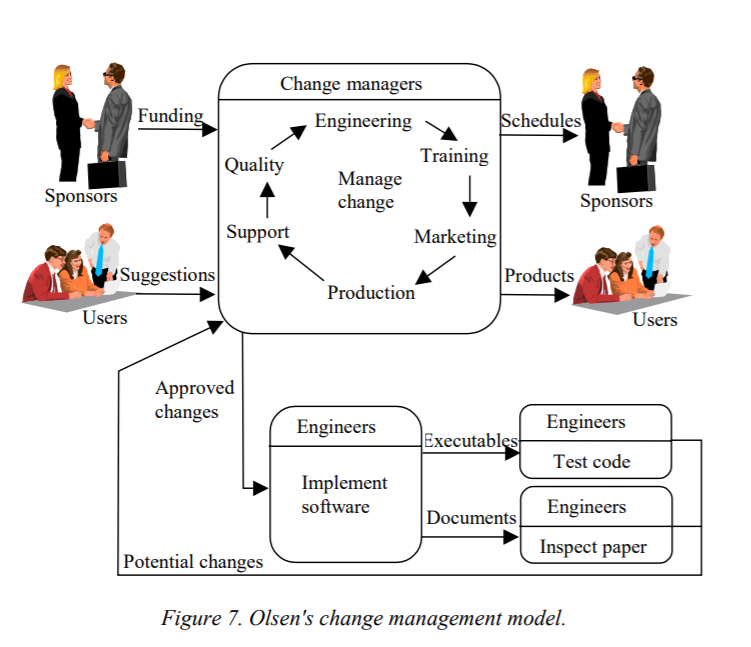
[[21]](#endnote-21)The first approach, Olsen’s approach, is an approach that was established over twenty years ago, but his definition of change is still relevant today. Olsen defines change as any work that is done on the software end of a system - whether it includes adding new features to a system or maintaining existing features. Olsen’s definition of change does not differentiate between different types of changes, making it not life cycle dependent and useful for both software development and maintenance. Olsen understands the change source as suggestions from users and change managers or verification activities. As can be seen in Figure 3.1, Olsen’s model assumes sponsors to provide funding for the change and monitor the scheduling of the change. Although Olsen’s change model provides a simple and clear definition of change, however; by bundling all activities under the same definition of change, it loses the ability to evaluate and alter existing features of software and necessitates new software development.

Figure 3.1 Olsen's change management model

[[22]](#endnote-22)

#### 

#### V-Like Change Management Model

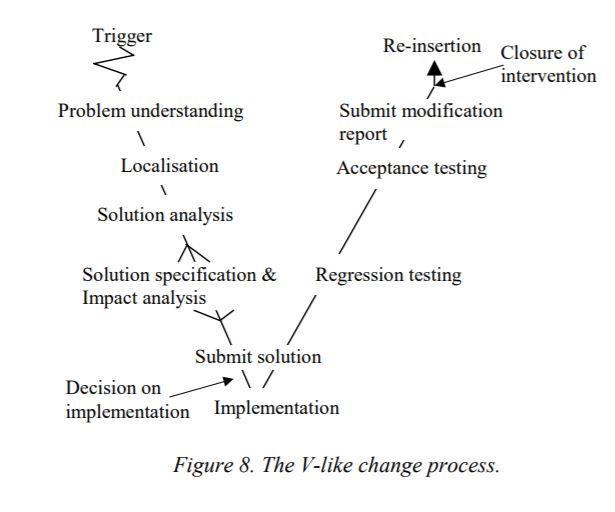
The second change management model is the V-Like Model. This model was constructed solely for maintenance and cannot be used for adding new features to an existing system. The V-Like model deals with all maintenance concerns, including but not limited to user support, corrective change, preventative change, and anticipative change. As can be seen in Figure 3.2, the V-Like model begins with the receipt of a problem. Once the problem is received, the change management team works to find the cause of the problem and evaluate all that will be affected if the change is implemented. The team then carries out a mini-development cycle of the product or feature where the change is necessary. This mini- development cycle is followed by regression testing to make sure the change will not negatively affect other features of the system and the software. Once all testing is completed, the software is then re-inserted in its operational environment and the problem is resolved.

Figure 3.2 V-Like change process

[[23]](#endnote-23)

#### Ince’s Change Management Model

The third change management model is Ince’s model. Ince’s model is similar to Olsen’s model in that the model can be used for both maintenance of existing features and creation of new features when necessary. Ince identifies the two main sources of change - external or internal requests. An external request is received from a customer whose requirements have changed and he is therefore requesting changes in the system to accommodate the new requirements. Internal requests are received when the system’s development team identifies an issue through the validation process that necessitates evaluation and modifications.

Once a request is received, whether it is internal or external, it is sent to the Change Request Board (CRB) who decides if the change is to be rejected, batched, or allowed. If the change is allowed, the request, along with its evaluation and modifications, is sent to the implementation team to implement the change. It is important to notice that Ince includes modification of the systems documentation following implementation of the change. The documentation of a change is many times overlooked in the bustle to implement the change, yet it is necessary for both the system’s current function and future changes.

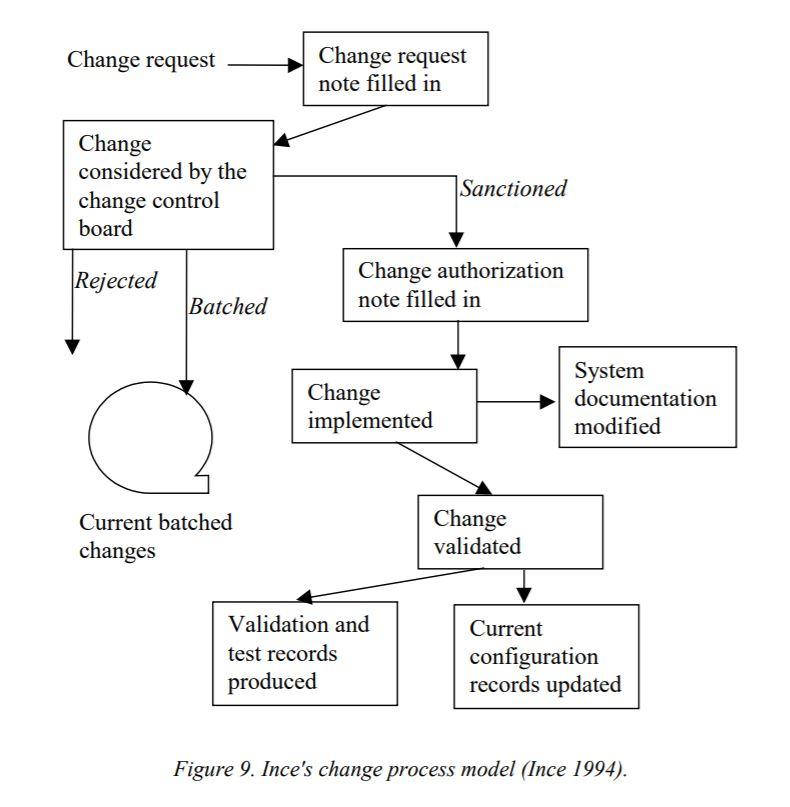


Figure 3.3 Ince's change management process

[[24]](#endnote-24)

#### The AMES Change Management Model

The fourth model, the AMES model, is unique in that it is intended to be used solely for maintenance, however; when necessary, the AMES model can be applied to the development phase as well. The AMES model (Figure 3.4) includes three levels, strategic, management, and technical. The strategic level involves the conservation and commercial aspects of the change including marketing, budgeting, and process improvement. The second level, the management level, is in charge planning and organizing all the aspects of change implementation from change initiation until closure. The final level of the ACME model, the technical level, is left to implement the change according to the management level’s evaluation and preparation.

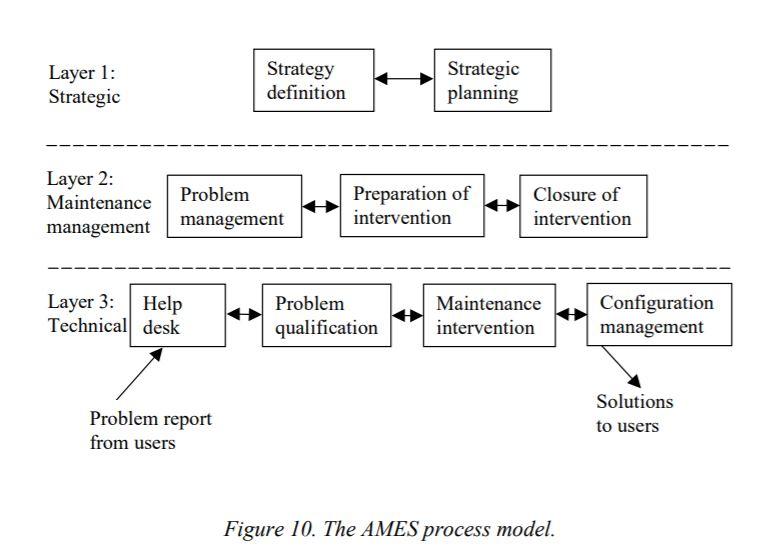
[[25]](#endnote-25)[[26]](#endnote-26)

Figure 3.4 AMES process model

##### 

##### Spiral-Like Change Management Model

The final model and most recent of the five is the Spiral-Like model. The Spiral-Like model, which is seen in Figure 3.5, has a spiralized presentation and is composed of four independent rounds. Within each round the same four main tasks are performed, but the viewpoint while performing those tasks changes based on the round. The four main tasks that are involved in each round are problem understanding, solution and risk evaluation, development, and preparation for the next round. The first round is the “problem owning cycle” which involves the “owner” of the problem, the identifier of the problem, deciding when and how the problem must be resolved. The second round, the problem-solving round, is optional. If it is necessary to examine the problem and solution from a non-technical viewpoint then this round is necessary, but if the solution was clarified in the first round then the second round may be skipped. The third round is the system engineering round, in which the change is evaluated from the system point of view and a detailed implementation plan is formulated. The fourth round, the technology-specific cycle, is when the change is implemented and verified - the change is documented, the results are delivered, and the problem is resolved.

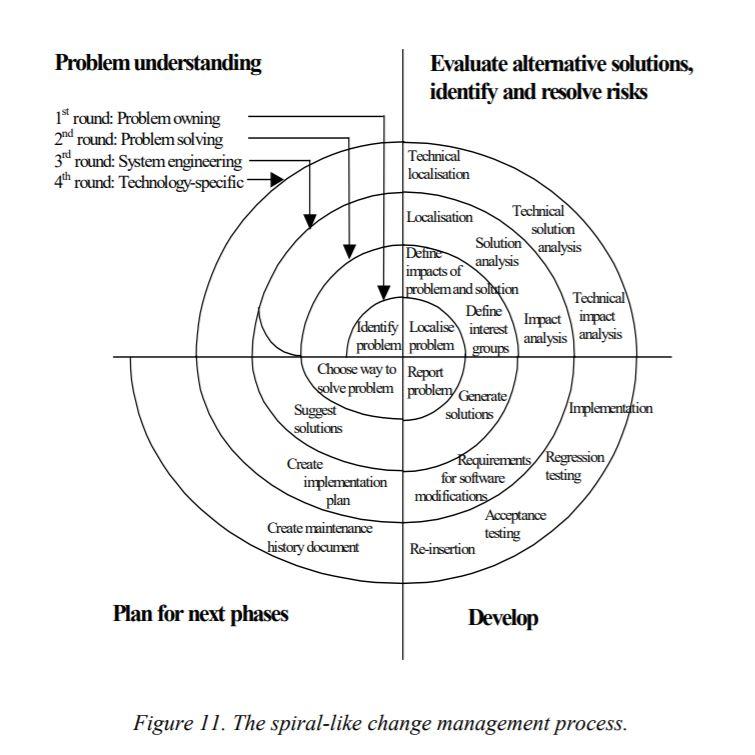
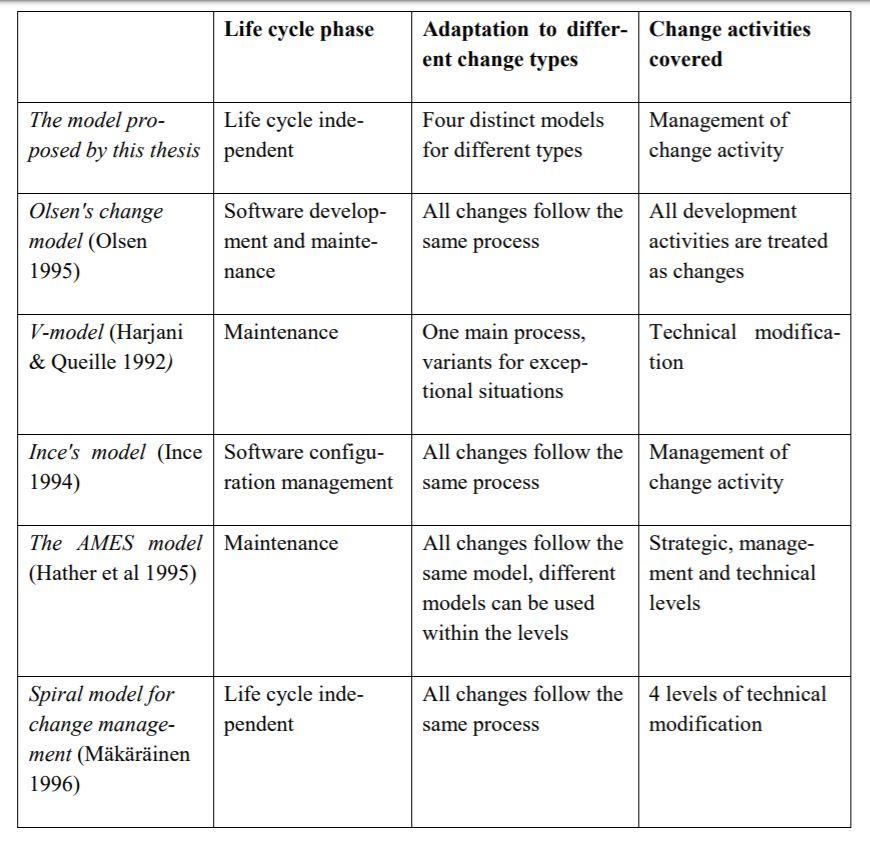


Figure 3.5 Spiral-Like change management process

#### Overview of Change Management Models

After contrasting the five change management models, one can see that there are quite a few similarities between the models. Some organizations choose to implement a specific change management process to properly suite their requirements and lead employees in the right direction. Although employees should try to follow the change management model that their organization has implemented, it is okay to bend the model’s rules a bit when necessary to fit suit one’s needs. As depicted below in Figure 3.6[[27]](#endnote-27), there are minor differences between the five models; however, there is much overlap between them all.

*[[28]](#endnote-28)Figure 3.6 Overview of Change Management Models*

### Urgency of change

#### Emergency Change

The time frame that a change must happen will depend on the urgency of that specific change. Emergency changes are needed to resolve problems which are crucial for the continuity of the business. Emergency changes are the highest priority and need urgent attention. It requires a quick approval by the change advisory board before the change can be implemented[[29]](#endnote-29). Emergency changes should be done as soon as possible at a low activity time, for example lunch or overnight. At least half the team should be present while implementing this change to be aware of any effects the change may have. Emergency changes are the most difficult because it must be done in a short amount of time causing it to be a high-pressure environment[[30]](#endnote-30). Many companies choose to do a quick patch-up in an emergency and then return to better fix the problem when the emergency has passed.

#### Expedited Change

Expedited changes are considered high priority changes which have a significant impact and are associated with a high risk[[31]](#endnote-31). After a quick approval with the change advisory board, at least half the team should gather to implement these changes. They must be aware of any impacts these changes may have on the system and be ready to resolve them[[32]](#endnote-32).

#### Latent Change

Latent changes are changes that have been performed already. These types of changes require approval from the CAB retroactively. If a change is latent, the request status is set to completed after the change requests are saved[[33]](#endnote-33).

#### Normal Change

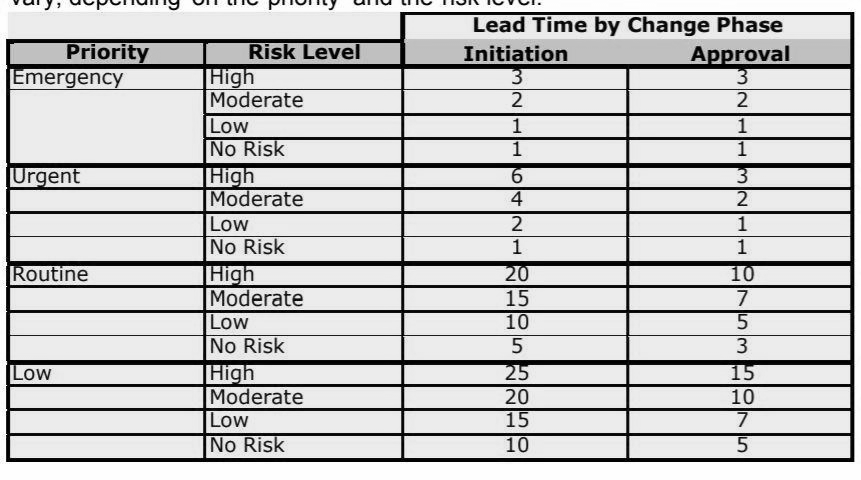
Normal change is the default value of a change which is required to go through a regular approval process by the CAB. There is typically a set time for when normal changes occur[[34]](#endnote-34).

#### Minimal Impact Change

Minimal impact is the lowest priority change. This is reserved for only pre-approved change requests.

#### Approval Process Depending on Urgency Level

The approval process for each change varies based on the urgency of that change. However urgent the change, it is necessary that the change request be submitted in time for the documentation, initiation processing, and approval to be received and leave enough time to implement the change by the intended completion date. Lead time, which is measured in days, is the amount of time the change request must be submitted ahead of the intended implementation date. As can be seen in Figure 4.1[[35]](#endnote-35), the priority level and risk level of the change directly affects the lead time. Within the priority level, the lead time rises as the urgency decreases. However, within each priority level, the lead time increases along with the risk level. This is based on the phenomenon that the larger the risk, the more time needed to evaluate potential dangers and decide on a correct implementation approach

***[[36]](#endnote-36)Figure 4.1 Initiation and Approval time based on Urgency Level*

### Smooth changes

#### Agile vs Waterfall

Agile and waterfall are the two common software development methodologies. Each method has several advantages as well as disadvantages. Although the agile approach is thought of as the new method for software development, it is actually an old concept known as prototyping. Evolving prototypes is the idea of building a small part and constantly growing the system; this is the concept of agile [[37]](#endnote-37). Agile is meant to be a continuous process which is flexible and allows for change at all steps. This method is based on the concept of coordination and integration, meaning the teams are smaller, high functioning, and collaborative. Additionally, customers tend to be satisfied with the agile approach as opposed to the waterfall approach because they are more involved. Some disadvantages with this approach include potential organization restructure and lacking proper groundwork for the project[[38]](#endnote-38). Another drawback to the agile method is that it demands more work on developers and clients. Furthermore, due to the lack of linear structure, it is harder to get proper documentation for the projects.

The waterfall approach on the other hand, is based on the idea of separating the processes. There must be intense planning and documentation from the beginning because it is structured in linear stages. Several advantages of this method include clear linear structure, focus on end goal, easy transfer from one step and team to another. Along with these conveniences come drawbacks such as not so much room for feedback, hard to learn from teammates, rigidness in the system, difficult, expensive, and slow[[39]](#endnote-39). The three main disadvantages with the waterfall system is that it does not allow for changes to be made at any step. This is because all planning must be done in the first two steps of ‘finding requirements’ and ‘design’ Additionally, because customers are less involved, they tend to show dissatisfaction. Thirdly, due to the rigid structure of this approach, all testing is done at the end of the project.[[40]](#endnote-40)

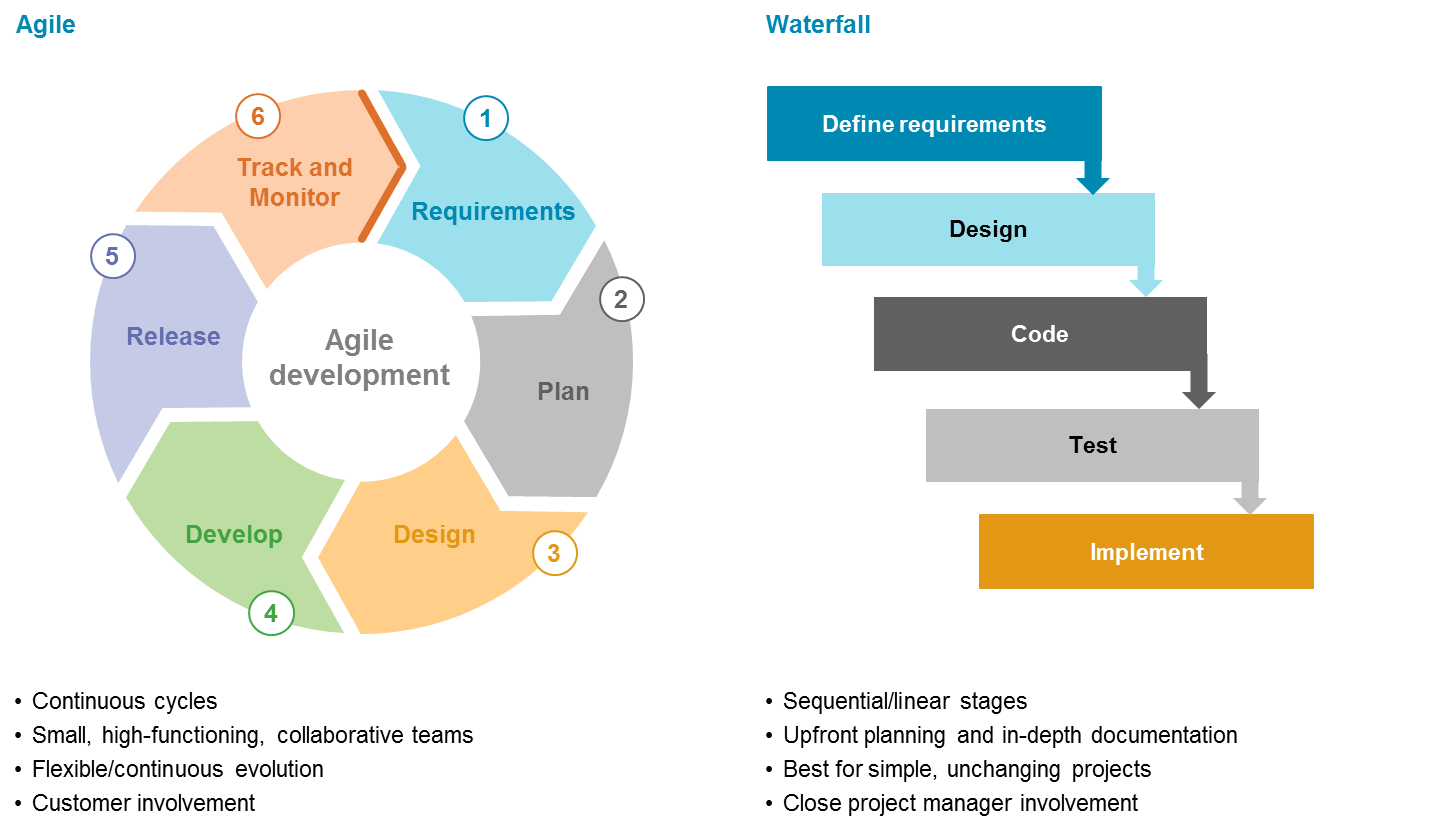
[[41]](#endnote-41)

Figure 5.1 Agile vs Waterfall

#### 

#### Parts Error Explosion

Parts error explosion is a major nightmare for many companies, and many companies put policies into place to protect themselves and their employees from this nightmare. A parts error explosion can occur when a company purchases a packages software from a company and then customizes the software to fill their requirements. However, a problem occurs when the vendor updates their software package, either by adding new features or updating existing ones. When the new version is released the company must go back to the basics and customize the software to fit their requirements again. Ideally, a company will establish procedures that track each change and customization to keep the software as much in sync as possible. At times there may be different groups making different changes to the system but if they work in a managed environment and employees are not stepping on each other’s toes, they can see success.[[42]](#endnote-42) In order to prevent parts error explosion, the software should be updated and maintained in releases, allowing different team members to customize and update different features without doing each other in. Another solution that many companies implement is the use of a systems database. A systems database details the characteristics of all installations and releases of the software system and stores the software package in an organized fashion[[43]](#endnote-43). Although, part error explosions are quite the pain, if the software package is meticulously maintained and updated in an organized fashion they can usually be prevented.

#### Push vs Pull

Changes can be approached with either a push or a pull framework. Many times, companies use fragmented and push-based approaches to change by dictating what the changes should be opposed to viewing changes as a flexible and collaborative effort. This push change model can also be called “the burning platform model” because it is often implemented when a system’s survival is contingent on the change. It got this name from the idea that when a platform is burning the best option is to jump- despite that fact that in normal circumstances jumping from a platform is not recommended. This is because when life, or in this case the life of a system, is endangered drastic measures must be taken[[44]](#endnote-44). With the push method, the change is imposed and the one implementing it does not have a choice in the matter. This is often used when the programmers have a laissez-faire attitude about the running of the system and need to be told what to do for the system to survive.

The push system of changes often causes employees to resist the change. Since change is dictated, this causes the employees implementing the change to feel resistant as they do not feel they were involved in the change and it was just forced upon them. This idea holds true regardless if the customer is an internal or external customer. Even if one is told to implement a change for their own benefit and to improve a system that they are using- they oftentimes still resist. Additionally, the resistance may come from a lack of trust towards the one that is demanding the change. Other reasons for resisting can be because they are under pressure and choose to trust the familiar or replacement tools are not proven- or in some instances don't exist. This causes the one charged with the task to implement the change to become overwhelmed at its seeming impossibility[[45]](#endnote-45).

An alternative approach to change would be a pull framework. With this, instead of the change being imposed, Employees have a say in the process of the change and contribute to which changes should be made and how to best go about their implementation. Since employees feel that they own the change, instead of resisting it they embrace the challenge. Additionally, it gives primacy to the people that will eventually use the new system after the change occurs. The pull approach is a more ‘positive’ change that is often used when everything is running relatively smoothly. However, Pull is not always better, push sometimes is what is needed to get the job done[[46]](#endnote-46).

### Implementation of Changes

### Acquisition of Resources

After acquiring approval for a necessary change, the implementation process can begin, beginning with the acquisition or building of necessary resources. There are three main category of resources that must be evaluated and calculated in this step of the change management process. Firstly, there must be a team that is available to efficiently carry out the change. Secondly, proper funding, whether internal or external sponsors, must be found. Lastly, software that can properly handle the change must be customized, built, or purchased

### Implementing the Change

After determining that adequate resources are available for the change to make sense, and a vision for the change is created, it is time to implement the change. This step includes implementing the change into production through a series of stages – giving both users and employees an opportunity to adjust to the change that was made.

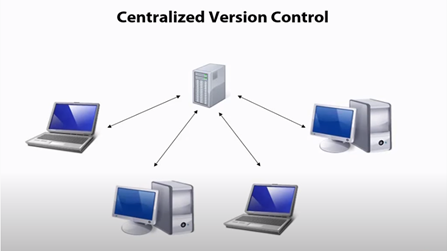
### Version Control

#### Importance of Version Control

Change management efficiency is contingent on using a version control system. Its importance can be explained by understanding that “coordinated management of changes to software assets for emergency hot-fixes, routine maintenance, upgrades, and new features with potentially overlapping development timeframes[[47]](#endnote-47).” Meaning to say that with a proper system of version control routine maintenance as well as debugging and emergency changes can happen simultaneously by different members of the team. More sophisticated version control systems are not dependent on a physical central server, but rather are web based, opening the doorway for worldwide collaboration. This is beneficial not only for remote workplaces but also for independent projects are able to flourish with combined international resources. Another reason why version control is vital is that it allows members to see what changes other members of the team made. This eliminates duplicate work and allows individual members to claim ownership for their contributions or mistakes.[[48]](#endnote-48) Additionally, a proper system for version control will have a system set up for storing and organizing previous versions. This is a major tool for any systems development because it allows for the flexibility to try new ideas without fear that previous, working versions will be lost.

#### Centralized Version Control vs. Distributed Version Control

For a team to efficiently carry out a change the team members must be able to collaborate, update, and change efficiently. There are many different systems of version control that aid with this process. Most major systems either fit into the category of centralized version control (CVC) or distributed version control (DVC). With a CVC system, everything is stored in one repository. When a developer wishes to change something, they need to pull a file or file segment from the central server, make the appropriate changes on their local machine, and then they put the file back into the larger system[[49]](#endnote-49). However, with a DVC instead of the server being the only repository, each client has a copy of all the versions saved on his local machine. This allows for each client to work locally on his machine and not rely on one central server. To work on a collaborated project, one downloads a copy of the project to his hard drive and makes changes there. When he is finished, he must push the changes that are saved on his local repository to the master repository[[50]](#endnote-50). The major difference is that code and data is able to be shared between individual repositories before it is committed to the central server. This is depicted in Figure 6.1 and Figure 6.2 below[[51]](#endnote-51). Since the DVC system allows for more flexibility and collaboration, it is considered the more efficient system[[52]](#endnote-52). However, a CVC system is still used 35% percent of the time because often the changeover to a DVC is simply not worth it[[53]](#endnote-53).

**

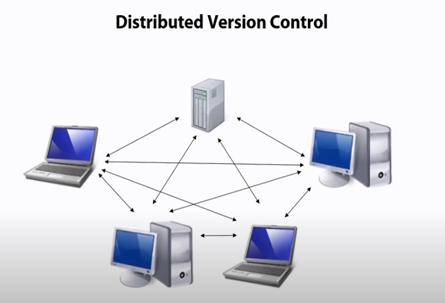
*Figure 6.1 Centralized Version Control*

Figure 6.2 Distributed Version Control

#### General function of version control

A robust version control system will include features that allow for changes to be easily documented, implemented, and organized. Documentation is important for the purposes of collaboration- it provides transparency of why and how the changes were made. Additionally, it helps future programmers understand the system that is in place. Another aspect of a version control is its portability and ease of deployment. Depending on the system that is being worked on, these aspects may or may not be important. It is also important to consider the file and directory renaming, moving, and copying. An efficient system will allow for these steps to be done easily. However, the most important aspect of version control is propagating the changes between the repositories. It is vital to update each repository so that redundant or unnecessary changes are not made. An additional feature of a robust version control system would be to track the file changes line by line, so it is clear what was changed, deleted, or added.

#### The Evolution of Version Control Systems

In 1990 Dick Grune and two of his students attempted to collaborate on a project, however their scheduling differences made it difficult to all be available at the same time. This was Grune’s inspiration and motivation to create the first system for version control, called Concurrent Versions System (CVS)[[54]](#endnote-54). The two main features of this system were version history and merging[[55]](#endnote-55). However, CVS was a CVC system and in 2000 Subversion was introduced and quickly gained popularity. Subversion is, like CVS, an open source software. Also, it has the same basic functionalities as CVS. The main difference is that while CVS is CVC, Subversion is DVC- which allows for more flexibility and manageability. Other systems began to develop as well, such as monotone, mercurial, and GNU.

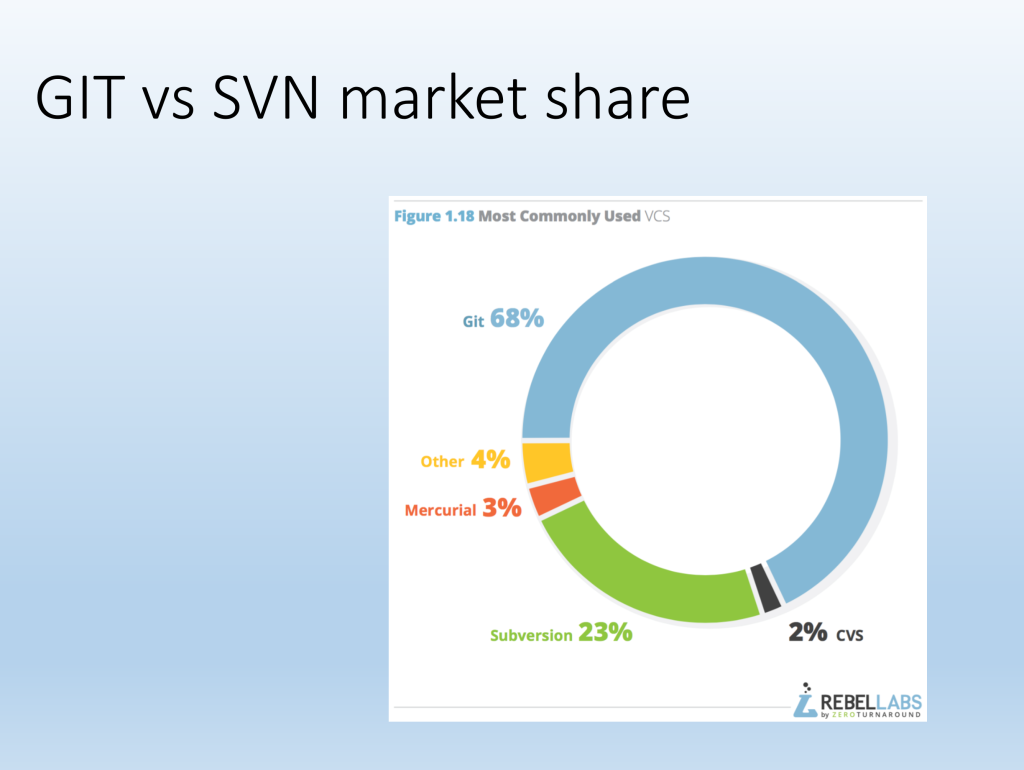
The next major development in this arena was with the creation of git. Git came about when the only viable options for Linux were proprietary. Linus Torvalds- git’s creator- wished to make a flexible, user friendly, and efficient system for version control. He set out to combat many of the problems that existed in the Linux based system. Therefore, Git is designed to be able to handle thousands of different developers that are operating simultaneously by eliminating the central depository- so there would not need to be a constant need for synchronization. Efficient performance was also something that was vital- in addition to utilizing space saving techniques and using DVC opposed to CVC, they also made objects in a database immutable to promote efficient comparisons for equality. The two features that are most recognized and appreciated are the encouragement and support of branched development as well as maintaining complete repositories. These two features allow developers to experiment with new ideas, collaborate, and merge with the security that previous versions are easily accessible[[56]](#endnote-56). As can be seen in Figure 6.3, due to these features, it is not surprising that the popularity of Git surpassed that of any other version control platforms.

Figure 6.3 Most Commonly Used VCS

##### Git Commands

|  |  |
| --- | --- |
| **Git Command** | **Function** |
| Add | Add file contents to the index |
| bisect | Find the change that introduced a bug by binary search |
| branch | List, create, or delete branches |
| checkout | Checkout and switch to a branch |
| clone | Clone a repository into a new directory |
| commit | Record changes to the repository |
| diff | Show changes between commits, the commit and working trees, etc. |
| fetch | Download objects and refs from another repository |
| grep | Print lines matching a pattern |
| init | Create an empty git repository or reinitialize an existing one |
| log | Show commit logs |
| merge | Join two or more development histories |
| mv | Move or rename a file, a directory, or a symlink |
| pull | Fetch from and merge with another repository or a local branch |
| push | Update remote refs along with associated objects |
| rebase | Forward-port local commits to the updated upstream head |
| reset | Reset current HEAD to the specified state |
| rm | Remove files from the working tree and from the index |
| show | Show various types of objects |
| status | Show the working tree status |

Git is essential software that allows for efficient version control. It includes many simple and easy to remember commands in order to accomplish what it does. These commands help to facilitate a host of functions including: creating new branches off of work repositories (branch), comparing versions (diff), committing changes (commit), merging different branches (merge), and many others shown in Figure 6.4[[57]](#endnote-57).

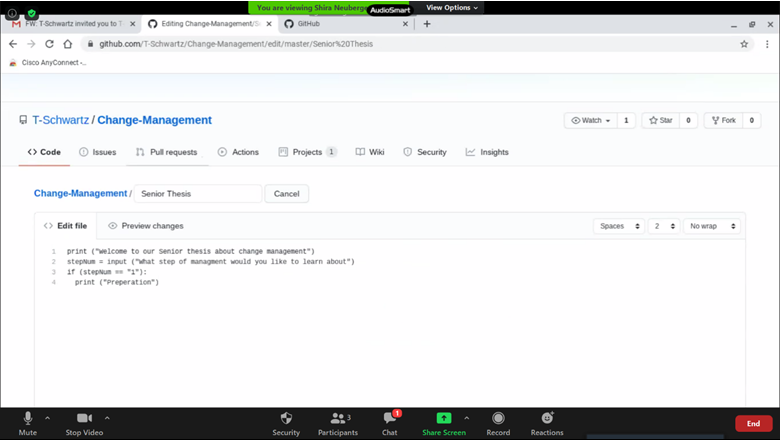
*Figure 6.4 Git Commands and Functions*

In 2008 GitHub was introduced as a web-based graphical interface for projects managed with git. Instead of git running from the command line, GitHub is run from a web browser and stores the data on the cloud opposed to on the local computer. Everything accomplished with Git can be accomplished through GitHub. GitHub allows for further collaboration as repositories can be made public and anyone can access them[[58]](#endnote-58).

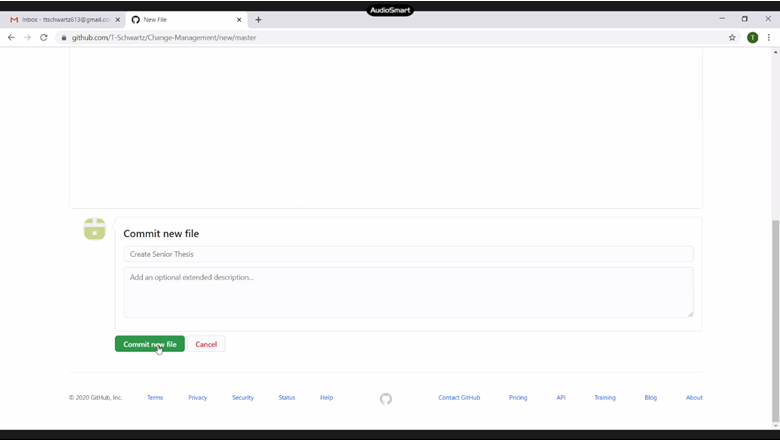
##### Creating a Central Repository on Git Hub

Creating a new central repository with Git Hub is a user-friendly process. Below are the steps that were taken to create a new repository with three collaborators.

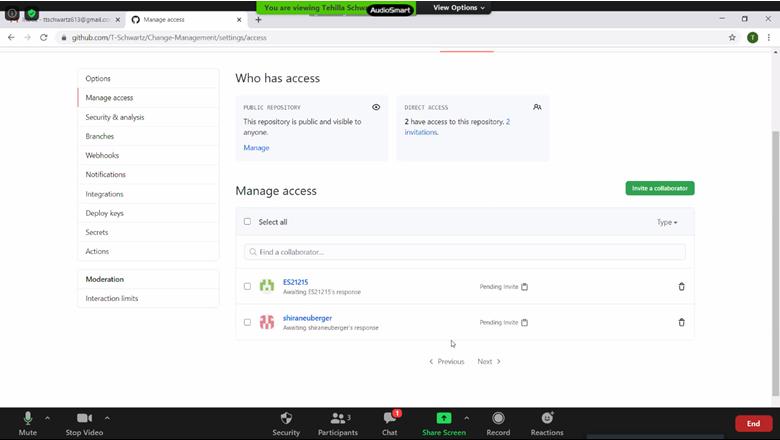
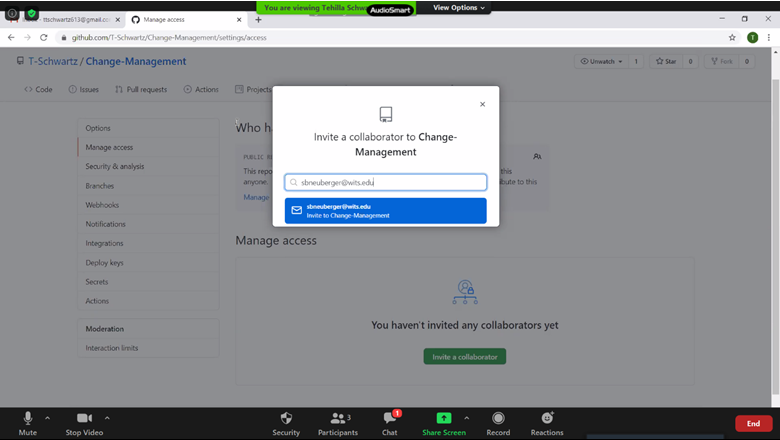
1. One team member creates the repository and adds some code. In this case, it is the beginning of a python program.



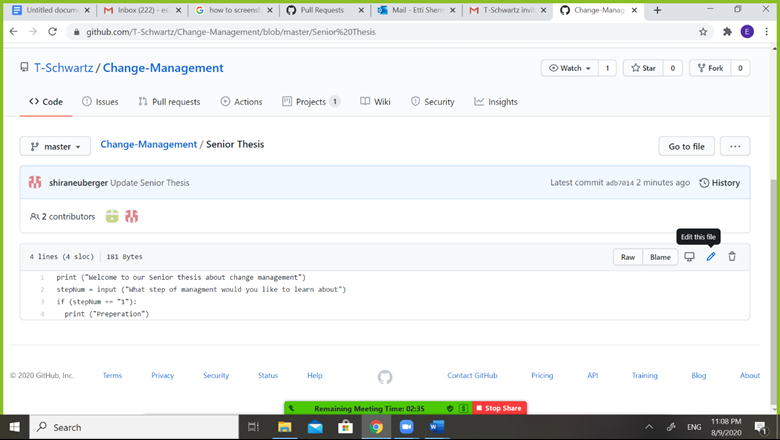
1. After all additions and updates or new information, one must commit their work.



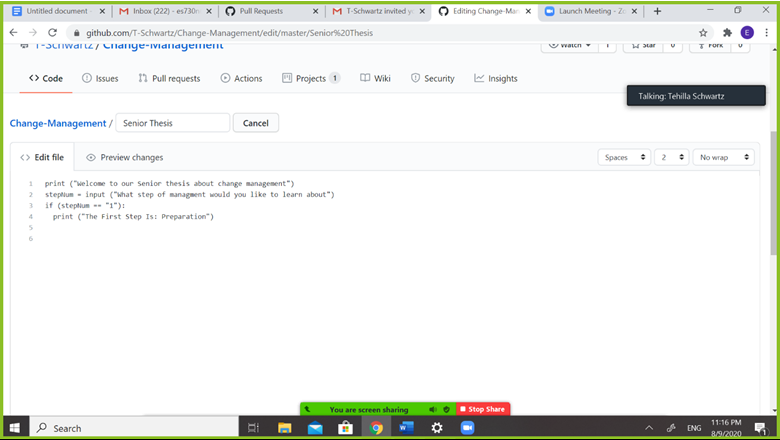
1. Next, collaborators can be invited to edit and add to the repository. Shira Neuberger and Etti Sherman were invited. They accepted the invitation.



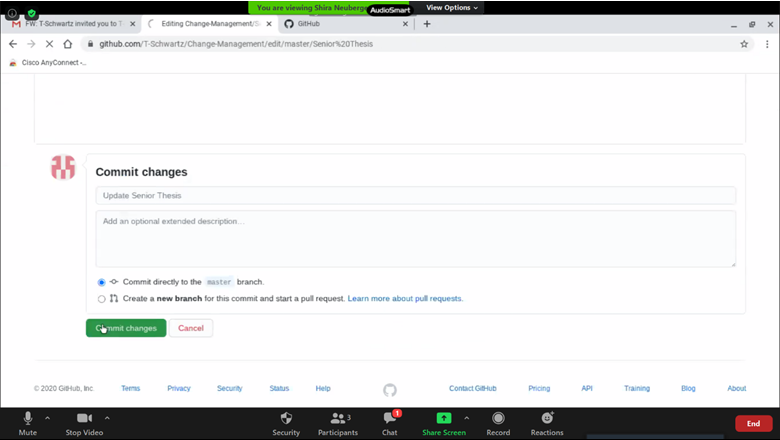
1. Shira Made an addition to the code.



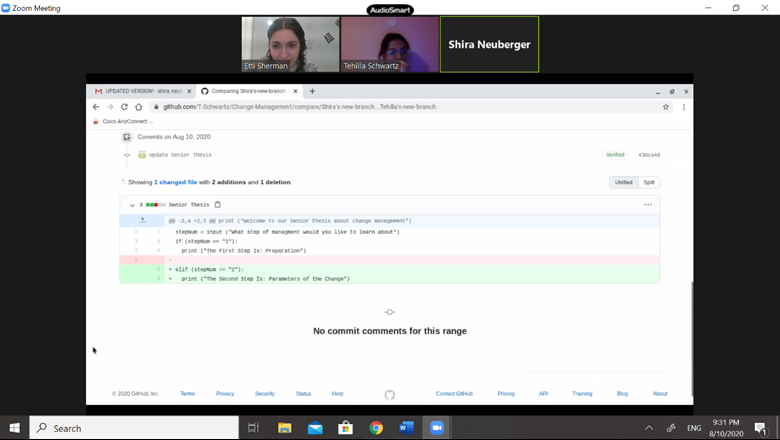
1. Etti edited Shira’s addition.



1. Both, Shira and Etti, made sure to commit their changes.



1. GitHub generated a Summary of the changes.



##### Branching and Merging

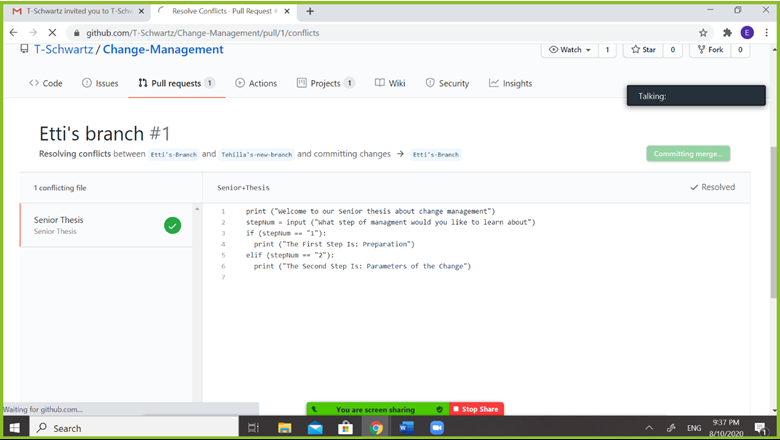
Branching and Merging is a function that helps greatly with collaboration, as it allows collaborators to make a copy of the central branch, change, update, or delete from it, and then add it back to the central branch or merge it with other branches. This reduces the of mutilating the main branch. Below is a sample of branching and merging.

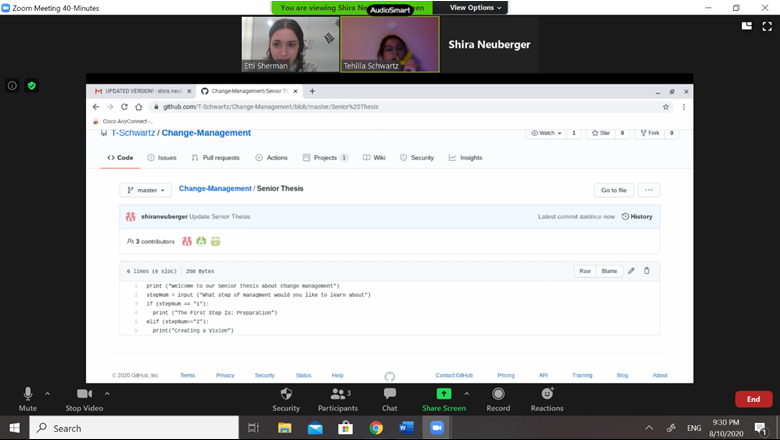
1. Tehilla created a branch from the master branch.

Machine generated alternative text:
M Inbox - ttschwartz613@gmail.cc X I @ Adobe Creative Cloud 
C github.com/T-Schwartz/Change-Management 
Google Calendar - August 2020 
T Schwartz/Change-Managemer 
o 
Creating and deleting branches 
Pull requests 
Search or jump to... 
T-Schwartz/ Change-Management 
Pull requests 
Actions 
1 branch O tags 
Issues 
Marketplace 
Projects 
Explore 
CD Wiki 
Star 
01 
Code 
o 
Issues 
Security 
Insights 
Go to file 
bbb78e3 
pr 
Settings 
Add file 
18 hours ago 
@ Unwatch • 
About 
master • 
Switch branches/tags 
Tehilla's new branc 
Branches 
Tags 
Create branch: Tehilla's new branch from 
'master' 
View all branches 
p Type here to search 
Code • 
3 commits 
18 hours ago 
Update Senior Thesis 
rstand your project by adding a README. 
Add a README 
No description, website, or topics 
provided. 
Releases 
No releases published 
Create a new release 
Packages 
No packages published 
Publish your first package 
Contributors 3 
T-Schwartz T-Schwartz 
ES21215 ES21215 
shiraneuberger shiraneuberger 
Fork 
4.•54 PM 
8/10/2020 

Machine generated alternative text:
M Inbox - ttschwartz613@gmail.cc X I @ Adobe Creative Cloud 
X Google Calendar - August 2020 
Editing Change-Management/Se 
o 
Creating and deleting branches 
-5 C github.com/T-Schwartz/Change-Management/edit/TehillaIs-new-branch/Senior9620Thesis 
Search or jump to.. 
Pull requests Issues Marketplace Explore 
T-Schwartz/ Change-Management 
Star 
No wrap 
01 
Code 
o 
Issues 
Pull requests 
Actions 
Projects 1 
Cancel 
CD Wiki 
Security 
Insights 
pr 
Settings 
Change-Management/ Senior Thesis 
Edit file 
@ Preview changes 
@ Unwatch • 
Spaces 
2 
1 
2 
4 
6 
7 
print ("Ldelcome to our Senior thesis about change management") 
input ("What step of managment would you like to learn about") 
stepNum 
if (stepNum 
print ("The First Step Is: Preparation 
elif (stepNum 
print ("The Second Step Is: Parameters of the Change") 
p Type here to search 
Fork 
4.•58 PM 
8/10/2020 

1. Etti and Shira also created a branch from the central repository. Tehilla’s, Etti’s and Shira’s branches do not match.





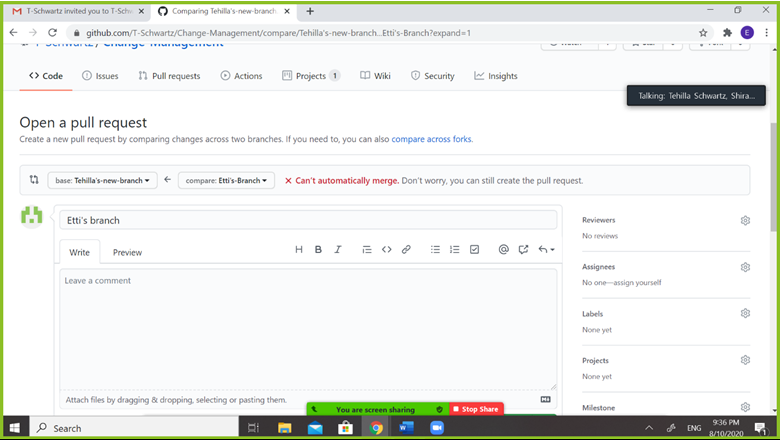
1. Next, a pull request was made to merge different branches.



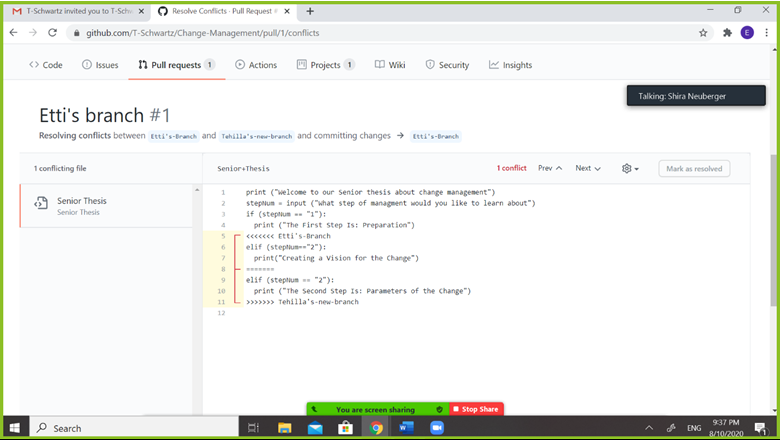
1. Tehilla’s and Shira’s branches merged without conflict.



1. There was a conflict when trying to merge Etti’s and Tehilla’s branches.



1. The conflict was resolved.



## 

## Caution Regarding Change

Changes in a database and programs require much caution because of the domino effect that they can have. Within a database it is important to consider other systems that use information from the database and the requirements those systems may have. For example, if a field is changed to hold long text where it previously held short text this can affect the intake form for the information, the report, and other programs and queries that are using the data. Additionally, adding and deleting columns, tables, and data should be approached with similar cautions. After a change to a database is made it is important to test the connected components to ensure they are not affected. When changing a program, one must consider its workability on different machines. Additionally, as with a database, one must ensure that the changes made on the program does not affect the surrounding and related programs, data, or databases.

## Example of Change in a Database

To demonstrate the changes that can be made in a database, we created a simple database and altered it. We created a database that holds the information of which companies offer which benefits to their employees. This database contains two tables. The first table, the customer table, holds the information regarding the customer – the companies name and number of employees that company employees. The second table, the employee’s benefits table, includes a list of which companies provide which benefits to their employees. Below are depicted the two tables that are included in this database.

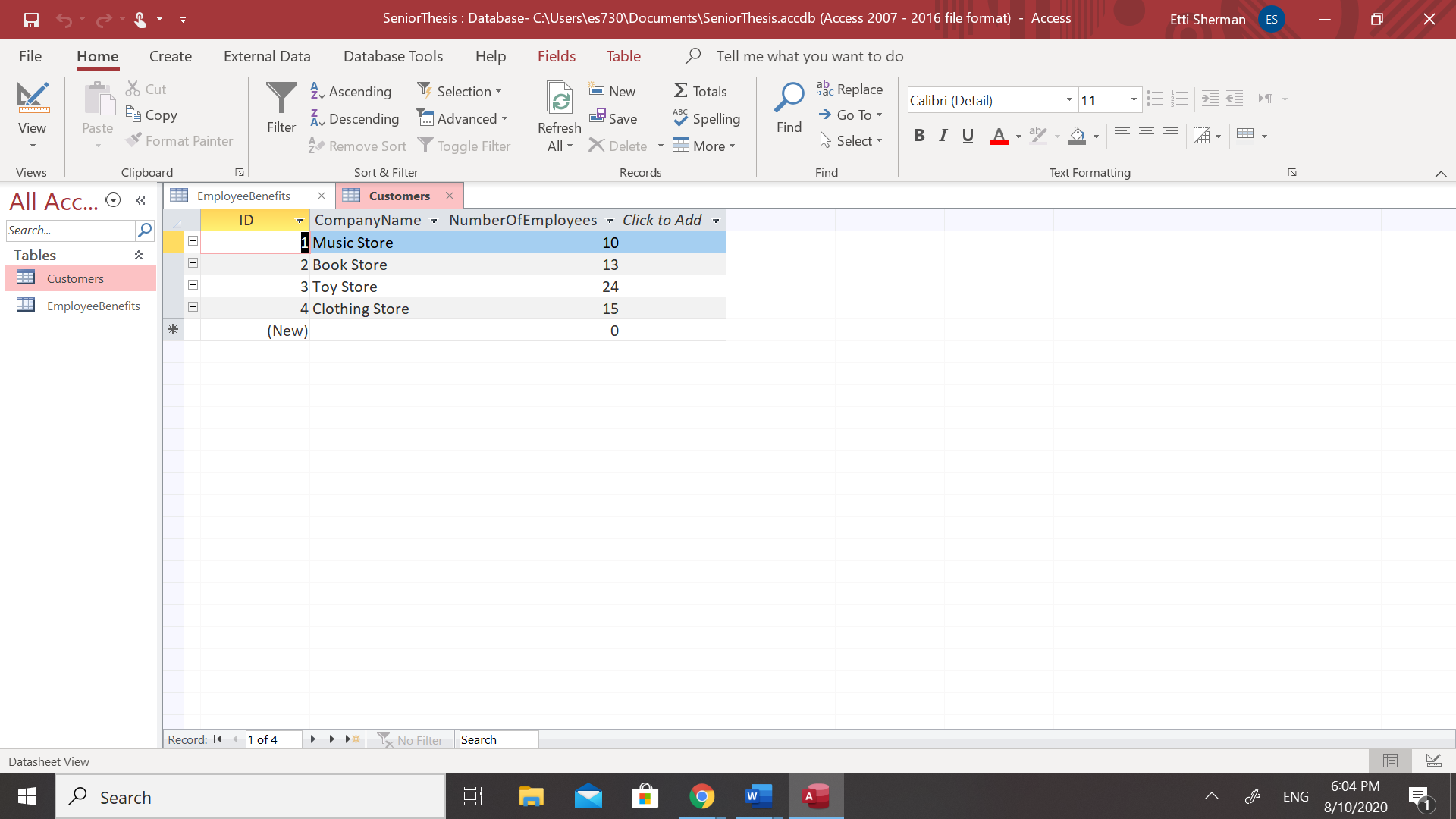


Table 1: Customer Table

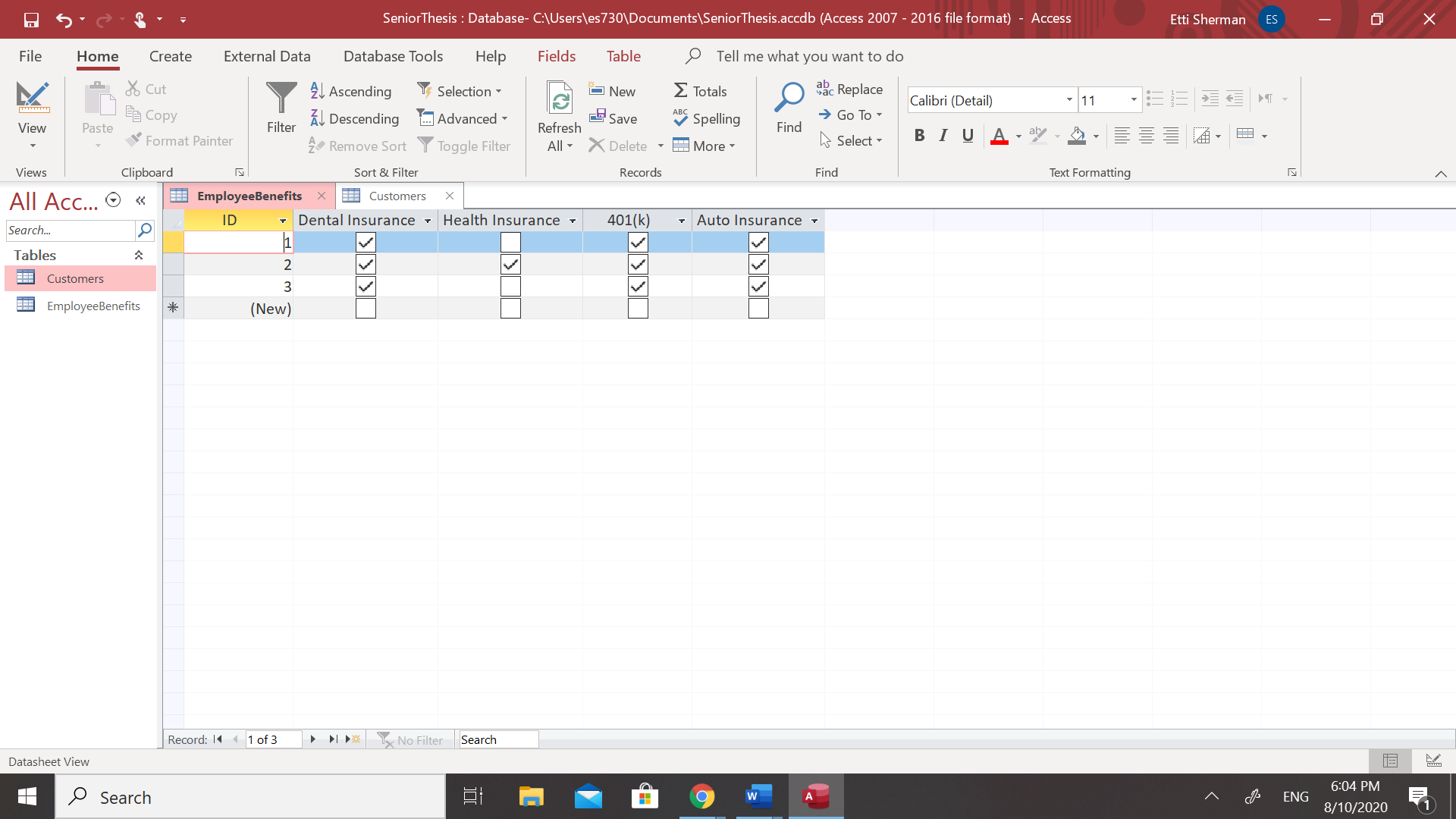


Table 2: Employee's Benefits Table

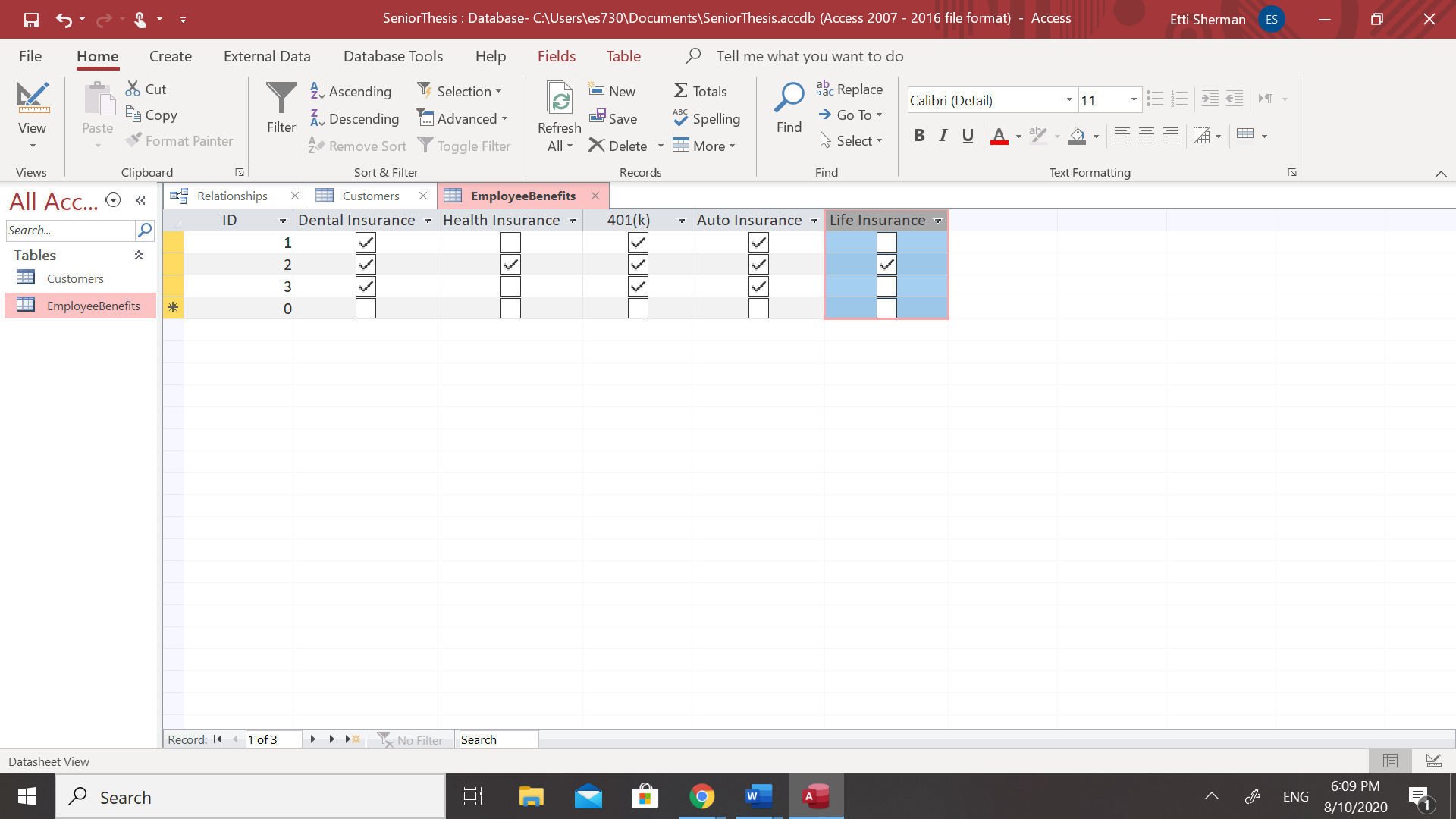
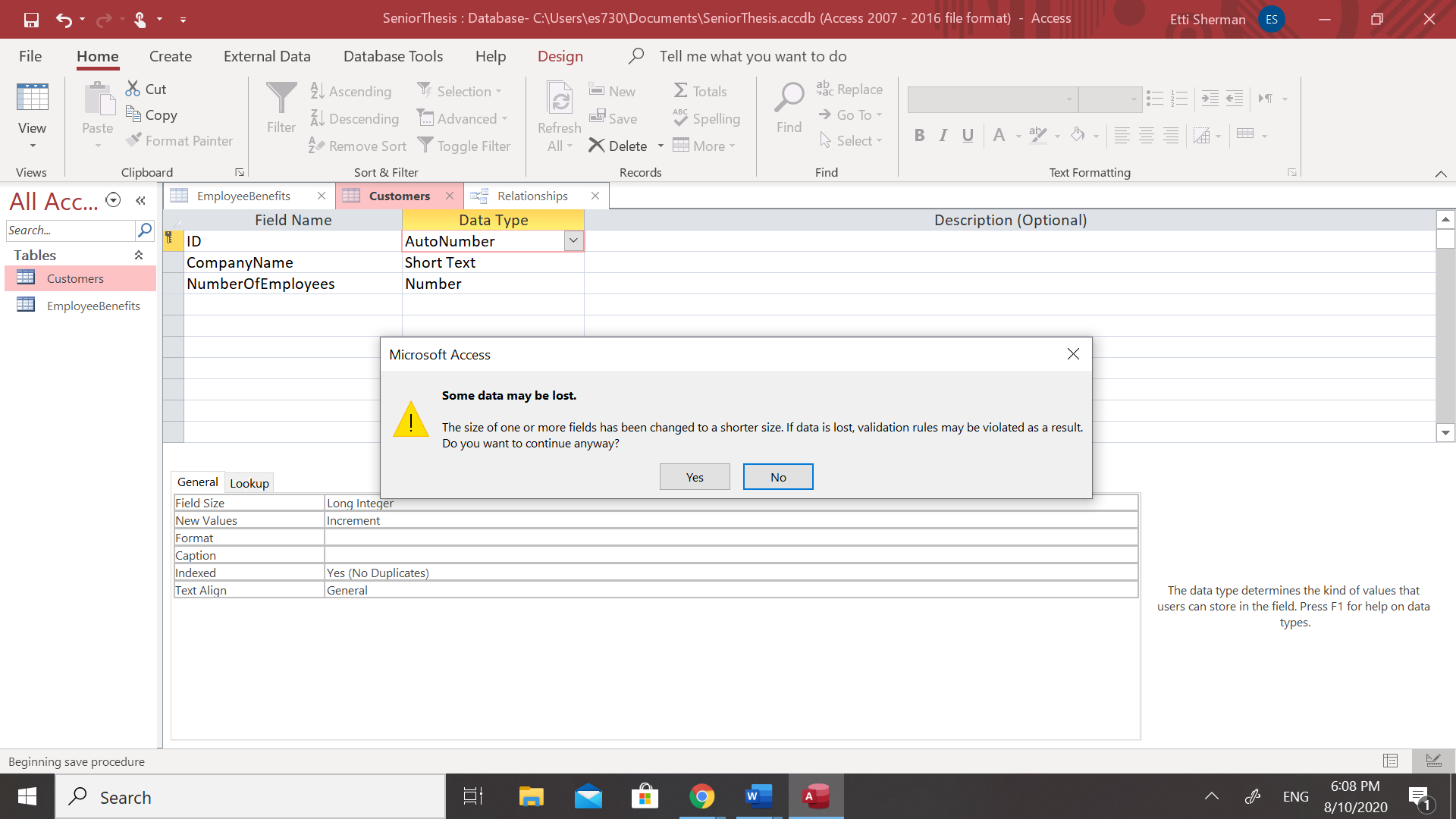
Regarding changes that can be made to a database, there are small changes that don’t take much time, money, and effort, and there are more complex changes that do. An example of a simple change would be adding or deleting a column to an existing table. Although this is a relatively simple change as demonstrated below, when dealing with a larger database that the example provided, even a small change, such as adding or deleting a column, can be much more complex. In the example below, we have some customers who started offering life insurance to their employees as an additional benefit and therefore we must add a “Life Insurance” column to the employee’s benefits table.

Table 2: Employee’s Benefits Table with Life Insurance Column Added

A database administrator must remember to update the data of all customers that now offer Life Insurance to their employees.

When discussing more complex changes done to a database, more time, money, and effort are usually involved. An example of a more complex change would be one that data can potentially be lost. As demonstrated below, When changing the field size of one of the fields on the customer’s table, an error message displayed warning that data can potentially be lost as the updated field size may not accommodate all the data in the table.

Using Microsoft Access, warning errors display to remind the database administrator that the change they are making may negatively affect their data.



However, must companies do not use Microsoft Access as a platform for their databases, and warning messages may not show when a change is negatively affecting the data. Therefore, a database administrator must check on their own that no data will be lost when the change is made as a warning message may not show.

The provided example is on an exceedingly small scale, and all changes made on a larger database are generally more complex than depicted above.

## Testing

### Regular Testing vs Regression Testing

A vital aspect to ensure successful changes made in a software system is testing. Regular testing and regression testing are two kinds of tests that must constantly be done to maintain efficient and reliable programs. Regular tests are tests done to find errors in code. New and revised code is tested meticulously to ensure that it did not negatively interfere with different parts of the code. Regression testing is testing pre-existing programs to ensure that the new code change does not interfere with other parts of the system. After the change is successfully implemented, one must retest the previous test to ensure the program is still intact. If there are bugs found, developers try to discover whether the issue is the old code that the new segment reintroduced or with the new code itself.

### Formal Regression Testing vs Informal Regression Testing

There are two kinds or regression tests known as formal test cases and informal test cases. Formal test cases can only be constructed once the tester has clarified two vital pieces of information – the known input and the expected output. The known input is responsible for testing the precondition and the expected output tests the post condition. If the output meets the post condition, then the code is working correctly and does not need further modifications. If the output is not as expected, modifications are necessary as code is erroneous. For each requirement, at least two test cases must be performed, a positive test and a negative test. Testers frequently enlist the help of a traceability matrix to organize the link between requirements and their corresponding tests. A test would be marked positive for a requirement if the no errors were found in that requirement’s specific code and negative if errors were found and modification is necessary. Contrary to formal test cases, informal test cases are not organized in a clear manner, rather the test outcome is reported after each run. To help a tester understand and evaluate complicated problems and systems, hypothetical stories are used These hypothetical scenarios may simply be a diagram or a depiction of a testing environment. Informal test case is generally used to test a system where there are no precise requirements and can therefore be based off of the operations of a similar system that has already been tested in a similar manner[[59]](#endnote-59).

### Reasons for Regression Testing

There are four main reasons for the need for regression testing. Firstly, requirement changes in a project often results in revising code to handle the new requirement. Any time a part of code is modified, regression testing must be done to insure the maintenance of the program. Secondly, if a new feature is being incorporated into an existing software, regression testing must be done. A third reason for regression testing is defect fixing. This is the way a program is marked by a developer who then passes this change to the testing team to verify it produces expected results. Lastly, performance issue fixing is ensuring that all the requirements are met properly. There are six common types of regression, local, unmasked, remote, corrective, and progressive. If the revised or changed code makes errors, it is considered a local change. Unmasked is a term used to describe changes that were implemented that revealed bugs in pre-existing code. A common effect of changes to programs is that it breaks a different area of the program, this is known as remote regression. “Corrective is applied when specifications are unmodified and test cases can be reused. It is triggered by corrections made to the previous version. Progressive It is applied when specifications are modified, and new test cases must be designed. It is triggered by new features added to the previous version.[[60]](#endnote-60)”

### Implementing Regression Testing

There are three primary techniques for implementing regression tests. Retest all, the first method, tests all the code again. This method is extremely expensive due to the time and resources it requires. This method is generally used on significant changes in the system because retesting all the code for minor changes is wasteful. Regression test selection is a technique used to select part of the test suite to run and be tested. There are reusable test cases which can be used in succeeding regression cycles and obsolete test cases cannot be used in succeeding regression cycles. The third method used for regression tests is known as prioritization of test cases. This is when the test cases are prioritized by the impact, how critical it is, and how often it is used[[61]](#endnote-61).

## Change Management for Large Business Critical Purchased Software systems

### Overview

A computer software system is considered business critical if the business will be severely damaged if the software fails or contains bugs. Business critical systems are often purchased from a software vendor and the vendor periodically releases a new version of the software system. Examples of critical business software are software for accounting and payroll, order entry and fulfillment, CAD/CAM design of vehicles, bank demands and loans, and stock exchange floor trading.

Generally, change management of these systems begins with the installation of the new version. After that is done, the data must be migrated from the old system to the new system. This is done using the vendor supplied migration software. Before using the new system, the users must be trained on the added and changed features[[62]](#endnote-62).

The vendors of these types of systems perform extensive testing to minimize problems before the new version becomes generally available to the customers. However, there may still be unforeseen problems or bugs in the new version that went undetected during the vendor's extensive tests. if the new version fails or has bugs, the company can be in danger of severe financial loss, severe regulatory penalties, or even go out of business. As such, migrating from an old version of the critical system to a new version must be done very carefully to ensure the smooth functioning of the business.

Steps must be taken to ensure that the new software works properly in the company’s environment, and that the transition to the new version of the software is not disruptive. Furthermore, if the new version of the software proves to be problematic after migration, there needs to be a way to back out the new version and continue business operations with the old version of the software. However, some work may have already been done using the new system and efforts must be made to minimize loss of work done on the new system, and minimal time to move back. Simply put, failure is unacceptable, and work should not be lost.

However, to make this happen correctly, the company must pay attention to many details. These extra efforts may be expensive, but the cost of a failed system is generally considerably more expensive.

### Up Front Preparation

#### Purchased Software Components and Customizations

To properly carry out a change, one needs to know all the parts that can potentially be affected by a change. The list will vary depending on whether the software system that is being used has been purchased as a “package” or it was built in house. When purchasing a software system, there are many components that can be customized and many options available to meet that organization's specific requirements. There are three main components that are included in all purchased system packages - necessary software, databases, and user interfaces. Upon purchase of the system, the customer will receive the main software system and any additional add-on software packages that may include specific features to meet the organization's unique needs. Many software packages allow the customer to use features that are part of the purchased software to customize how that software will operate. These features must be redone when the system is updated to a new version. Additionally, there are options that must be chosen during installation of the new system, making it necessary to evaluate previous versions of the system. Included in many software packages along with the system’s program, is the system’s databases. Ideally, an organization should know the complete structure of the databases included in the purchased system. At a minimum, the company must understand the main components of data that are included in the database of the purchased system. Lastly, the system package will include user interfaces and web pages that are necessary for the system to operate properly.

#### Communication with the User Community

When a vendor is updating the software system they sell, the vendor will provide the organization with all the necessary materials and knowledge to properly operate the updated system. This briefing will include a list of new features that have been added and a list of existing features that have changed. Additionally, the vendor will provide training materials for current users and new users to review or learn existing features and updated features of the system. Lastly, the vendor must communicate to all customers the date when support for the old version will be discontinued, as all customers should have updated to the new version by that time.

#### Evaluation of Additional Changes

When changing and updating a large system, one must consider all angles of the system at hand and decide which parts of the system necessitate change. Firstly, one must determine if the operating system needs to be upgraded. Additionally, one must evaluate the current systems software and decide if it needs modification. For example, one may need to upgrade the Database Management System to a new version or change the drivers of the system Lastly, one must consider the existing hardware, including capacity and speed, and determine if change is necessary.

#### Updating the User Community

Regarding the user community, the vendor must obtain a list of all current users of the current version of the system. Once the vendor has obtained the old list, a new list must be developed to include all users for the new version. It is important to remember that the new list of users may differ from the old list of users as there may be new users that are added or old users that are removed.

#### Interactions with Other Systems and Companies

Another consideration to evaluate when updating a large system is the interactions the system has and will have with other systems and companies, both internal and external. Internally, one must develop a list of all in house systems that will send data to the new version and/or receive data from the new version. Externally, one must develop a list of all data that is extracted from the system to be sent to outside companies and/or data that is received from outside companies to integrate into the system.

Lastly, a list must be developed that includes all reports and queries that use the data in the current version of the system, to ascertain that all those reports and queries will be updated to handle data from the new version of the system upon release.

### Making a Go or No-Go Decision for Upgrading to the New Version

When the company continues to use the software system, they must gradually migrate to the newer version of the system. As time progresses, the older versions of the software may become incompatible with parts of the computing environment. This should be done sooner rather than later because the longer one waits to migrate to a newer version, the more difficult and more costly it will be. For example, if a company is currently running version N, and skips a version but wishes to migrate to version N=+2, it may be necessary to temporarily migrate to version N+1 and then N+2. There are several issues that may arise unexpectedly by using double migration; therefore, one must always develop a change plan which includes a timeline, the people needed, and the total cost. After all factors are evaluated, one must decide if and when the company will go ahead with the move to the new version of the system. Contrarily, an organization may decide not to implement the new version and wait for the next version.

#### Evaluating Costs of a No-Go Decision

But, if one makes the decision not to migrate, or to delay migration there are also costs. The older version may not be as efficient, or one must do extra work because a work saving feature in the new version is not available in the old version. Regarding purchased software, vendor support for the old version is not as good as vendor support for the new version. Generally, one must look at both the cost of doing and the cost of not doing and make an educated decision once all factors are evaluated

### Develop Back Out Method in Case of Failure

Every company must have a back out plan which is used when the company decides to go ahead with full production of the new version and soon discovers significant issues related to the new version and must quickly return to the old version. Additionally, any work done using the new version should be migrated back to the old version as soon as possible.

The first step for a successful change management is to install a new version in a test region or machine followed by training team members on how to use the new version in this test region They migrate a limited amount of data to the new version which will be used for training and testing. Preliminary tests are done on the new version to modify the old work and create the new work. This will also serve to provide data that can be used to test the reverse migration.

### Prepare for Failure of the New Version

The company must prepare for the possibility of the new system failing, forcing them to change back to the old version. There must be programs written in advance which transfers all the work done on the new system to be inserted back into the old system. This way the company minimizes the amount of rework for any work done with the new system. These programs are crucial because vendors are only responsible for the transfer of data from old versions to new versions and not vice versa. This is the responsibility of the company and must be done because it is vital in case of a failed new system. There are two main types of programs which transfer data from the new version back to the old version. The first method is a direct database conversion. The second option is to export data using a facility in the new version and import the data using a facility in the old version. Programs may have to be written to modify the exported data so that it will go properly into the old version.

After all the data was transferred from the new version back to the old version and production was resumed with the old version, the vendor should try to resolve the problems with the new version. Once all the issues have been fixed, the company may try again to switch to the new version.

### Updated Migration Plan

The company should develop an updated plan for the migration based on knowledge gained while doing the early tests and reverse migration mentioned above. They should determine whether the new or improved features are helpful to the company, possibly helpful to the company, or no interest to the company at all. Important personal aspects should be included in the plan. For example, the vision of how the change can help the company. What new or updated skills will the users need? What incentives can be provided to ease the change? Are there additional resources that will be needed in the short or long run?How do we best communicate the plan to the users? Additionally, the cost should be estimated for migrating to the new version and for not migrating to the new version. This is an important factor whether the company should execute the plan or not.

### Test New Version

This testing of the new version is far more extensive than the earlier mentioned testing. The previous tests were interested in limited testing so that the company can begin to develop the reverse migration. At this point a full test must be done. The testing of the new version is done by the test team. In general, the test team should consist of the more experienced users, including some power users. Such a team will more readily recognize minor changes to the system that need to be pointed out when the general users are trained on the new system.

A copy of the new system must be installed in a test region or machine. A team should then be designated for testing the new system. The first step is for the data to be migrated from the old version to the new version to be used for the tests. Then the test team must be trained well on the new software. Training must include old features that have changes as well as new features. Tests should first be done on all the old features to make sure they still work on the new system. Additionally, one should keep track of any changes in how a user uses the old features on the new system and keep track of old features that are not available on the new system. What alternatives are available on the changed system? All problems encountered should be properly reported to the vendor. Tests should be done on all the new features. All problems encountered should be properly reported to the vendor. Do a major test of the reverse migration procedure. After the vendor resolves all the problems in the new version, redo the features tests and check out all the changes made by the vendor. A full regression test should be done on the whole software.

### Test Software Interactions with the System Being Changed

Before updating the system, one must verify that all software that interacts with the older version of the system is interacting properly with the new version as well. If the software is not compatible with the new version, it must be updated to insure compatibility. Firstly, one must verify that each system that extracts data from the system being changed works properly with the new version as well. One must ascertain that the extracted data is correct and usable following the release of the new version. Secondly, one must verify that the data that is sent and received from outside the company still works properly with the new version. Lastly, one must test that all reports and queries are updated and running properly.

### Phase New Version into Production

Before phasing the new version into production, one must fully back up the old version and have it backed up in case the new version fails. The vendor migration program is then used to move the data from the old version of the system to the new version. After the migration is complete, the vendor must test that the transfer was successful. Lastly, one must determine all the users of the new version. It is important to remember that the list of new version users may differ from the list of old version users.

After the migration is complete, one must appoint three special teams to provide training and support to users as they transfer to the new version of the system. The first team, the trainers, are responsible to train users on how to properly work with the updated system. The second team, the support, must be on the lookout for problems with the new version and try to resolve any problems the users may have. In the situation that the support team cannot resolve the user’s problem, the support team should request vendor support or a back out. The last team must be prepared and ready to do a reverse migration if necessary. Although each of the three teams is unique in their roles, it is possible for one person to be a member on more than one team and the team members of each group may overlap.

Since the new version of a system may fail, it is important to phase the new version into production in stages. Firstly, the users should be trained on all the features that can be migrated back from the new version to the old version. Once trained, users should only be allowed to use the features that can be migrated back to the old version. The vendor must allow several weeks for the trial period to run, giving the users an adequate amount of time to test if the new version is working properly. After the completion of the trial period, the users should then be trained on all features of the new system, especially those features that were added to the new version. Once trained on the new features, users may use all features - both those that existed on the old version and those added to the new version. By breaking the phasing into stages, the vendor is verifying that the new version works while preparing for failure if necessary.

### Support for the new version

There must be put into place a system of support and help for the users of the new version. Initially this may be handled by the members of the test team along with support from the vendor. The support needs to be gradually transferred back to the help team that will support the new version (along with help from the vendor as needed).

### People Consideration

For any change to be successful, one must consider the organizational readiness aspects of the change. There should be people in contact with the users on a reasonably regular basis to pick up signs of grumbles or problems. The faster these issues can be resolved, the more successful the migration.

## Conclusion

After analyzing the significance of change in software systems, it is evident how change is essentially fueling our technological advancements and improving society. When studying and examining the change management of software systems, one should focus on the progress and improvement being accomplished rather than the difficulty involved. This includes ensuring that each step of the change management process is completed properly. Change is what keeps a system relevant and the organization’s brilliance alive. As Albert Einstein wisely stated, “The measure of intelligence is the ability to change”. After closely examining the impact of change, one can understand why change plays a key role in computer systems management.

1. **Machiavelli**, Niccolò, and Peter Bondanella. The **Prince**. Oxford: Oxford University Press, 2005. Print. [↑](#endnote-ref-1)
2. Zelkowitz, M. V., Gannon, J. D., & Shaw, A. C. (1981). *Principles of software engineering and design*. Taipei: The Southeast Book. [↑](#endnote-ref-2)
3. Ibid [↑](#endnote-ref-3)
4. Ibid [↑](#endnote-ref-4)
5. Ibid [↑](#endnote-ref-5)
6. Dr Rosenbaum [↑](#endnote-ref-6)
7. Clegg, Chris, and Susan Walsh. “Change Management: Time for a Change!” *European Journal of Work and Organizational Psychology*, vol. 13, no. 2, 2004, pp. 217–239., doi:10.1080/13594320444000074. [↑](#endnote-ref-7)
8. Ibid. [↑](#endnote-ref-8)
9. Yu L, Mishra A. An empirical study of Lehman’s law on software quality evolution. Int J Software Informatics, Vol.7, No.3 (2013): 469–481. http://www.ijsi.org/1673-7288/ 7/i152.html [↑](#endnote-ref-9)
10. Lehman, M M, et al. “Metrics and Laws of Software Evolution - The Nineties View.” *History Studies International Journal of History*, vol. 10, no. 7, 2018, pp. 241–264., doi:10.9737/hist.2018.658. [↑](#endnote-ref-10)
11. Ibid. [↑](#endnote-ref-11)
12. Yu L, Mishra A. [↑](#endnote-ref-12)
13. Lehman, M M, et al. [↑](#endnote-ref-13)
14. Ibid. [↑](#endnote-ref-14)
15. 5 Critical Steps in the Change Management Process: HBS Online. (2020, March 19). Retrieved August 12, 2020, from https://online.hbs.edu/blog/post/change-management-process [↑](#endnote-ref-15)
16. Ibid [↑](#endnote-ref-16)
17. Dr Rosenbaum [↑](#endnote-ref-17)
18. Dr Rosenbaum [↑](#endnote-ref-18)
19. 5 Critical Steps in the Change Management Process: HBS Online. (2020, March 19). Retrieved August 12, 2020, from https://online.hbs.edu/blog/post/change-management-process [↑](#endnote-ref-19)
20. “Change Management Framework.” *Change Management Framework Powerpoint Presentation Slides*, [www.slideteam.net/change-management-framework-powerpoint-presentation-slides.html](http://www.slideteam.net/change-management-framework-powerpoint-presentation-slides.html). [↑](#endnote-ref-20)
21. Makarainen, Minna. “Olsen's Change Management Model.” *“Change Management Framework.” Change Management Framework Powerpoint Presentation Slides*, 2000. [↑](#endnote-ref-21)
22. [↑](#endnote-ref-22)
23. Makarainen, Minna. “V-Like change process.” *“Change Management Framework.” Change Management Framework Powerpoint Presentation Slides*, 2000. [↑](#endnote-ref-23)
24. Makarainen, Minna. “Ince's Change Management Process.” *“Change Management Framework.” Change Management Framework Powerpoint Presentation Slides*, 2000. [↑](#endnote-ref-24)
25. Makarainen, Minna. “Spiral-Like change management process.” *“Change Management Framework.” Change Management Framework Powerpoint Presentation Slides*, 2000. [↑](#endnote-ref-25)
26. Makarainen, Minna. “AMES Process Model.” *“Change Management Framework.” Change Management Framework Powerpoint Presentation Slides*, 2000. [↑](#endnote-ref-26)
27. [↑](#endnote-ref-27)
28. Makarainen, Minna. “Overview of Change Management Models.” *“Change Management Framework.” Change Management Framework Powerpoint Presentation Slides*, 2000. [↑](#endnote-ref-28)
29. “Change Urgency and Timing.” *Change Urgency and Timing | University IT*, 2019, uit.stanford.edu/service/changemgt/urgency\_timing. [↑](#endnote-ref-29)
30. Dr Rosenbaum [↑](#endnote-ref-30)
31. “Change Urgency and Timing” [↑](#endnote-ref-31)
32. Clegg, Chris and Susan Walsh [↑](#endnote-ref-32)
33. “Change Urgency and Timing” [↑](#endnote-ref-33)
34. Ibid [↑](#endnote-ref-34)
35. [↑](#endnote-ref-35)
36. “Risk Level Based Lead Times.” *Change Management Policies and Procedures Guide*, Tuskegee University, [www.tuskegee.edu/Content/Uploads/Tuskegee/files/OIT/Change%20Management%20Policies%20Final.pdf](http://www.tuskegee.edu/Content/Uploads/Tuskegee/files/OIT/Change%20Management%20Policies%20Final.pdf). [↑](#endnote-ref-36)
37. Dr Rosenbaum [↑](#endnote-ref-37)
38. Clegg, Chris and Susan Walsh [↑](#endnote-ref-38)
39. Ibid [↑](#endnote-ref-39)
40. [↑](#endnote-ref-40)
41. Thanh, Pham. “Agile Change.” *Agile Vs. Waterfall in Software Development*, Saigon Technology, [saigontechnology.com/blog/agile-vs-waterfall-in-software-development](http://saigontechnology.com/blog/agile-vs-waterfall-in-software-development). [↑](#endnote-ref-41)
42. Dr Rosenbaum [↑](#endnote-ref-42)
43. Clegg, Chris and Susan Walsh [↑](#endnote-ref-43)
44. Boivie, Catherine Aczel. “Change Management: Deciding Whether to 'Push' or 'Pull'.” *IT Business*, 2013, www.itbusiness.ca/blog/change-management-deciding-whether-to-push-or-pull/31268. [↑](#endnote-ref-44)
45. Clegg, Chris and Susan Walsh [↑](#endnote-ref-45)
46. Boivie, Catherine Aczel [↑](#endnote-ref-46)
47. “Version Management.” *Version Management | IT Best Practices | Nebraska*, its.unl.edu/bestpractices/version-management. [↑](#endnote-ref-47)
48. Ibid. [↑](#endnote-ref-48)
49. “Centralized vs Distributed Version Control: Which One Should We Choose?” *GeeksforGeeks*, 23 Aug. 2019, www.geeksforgeeks.org/centralized-vs-distributed-version-control-which-one-should-we-choose/. [↑](#endnote-ref-49)
50. Ibid. [↑](#endnote-ref-50)
51. Tech Tube, Help, director. *GitHub Tutorial 03 CVC versus DVC*. 2015, www.youtube.com/watch?v=y4TPcG\_hGW4&t=151s. [↑](#endnote-ref-51)
52. “Centralized vs Distributed Version Control: Which One Should We Choose?” [↑](#endnote-ref-52)
53. Brindescu, Caius, et al. “How Do Centralized and Distributed Version Control Systems Impact Software Changes?” *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, doi:10.1145/2568225.2568322. [↑](#endnote-ref-53)
54. “ Concurrent Versions System CVS.” *Concurrent Versions System CVS*, dickgrune.com/Programs/CVS.orig/. [↑](#endnote-ref-54)
55. “CVS—Concurrent Versions System v1.11.23.” *CVS-Concurrent Versions System v1.11.23*, www.gnu.org/software/trans-coord/manual/cvs/cvs.html. [↑](#endnote-ref-55)
56. Loeliger, Jon, and Matthew McCullough. *Version Control with Git*. O'Reilly, 2012. Pg 2-3 [↑](#endnote-ref-56)
57. Ibid. Pg 18 [↑](#endnote-ref-57)
58. Simplilearn, director. *Git vs GitHub*. 2020, www.youtube.com/watch?v=xJXgvr8bPes. [↑](#endnote-ref-58)
59. Singhrajal, Jaspreet, and Shivani Sharma. “A Review on Various Techniques for Regression Testing and Test Case Prioritization.” *International Journal of Computer Applications*, vol. 116, no. 16, 2015, pp. 8–13., doi:10.5120/20418-2718. [↑](#endnote-ref-59)
60. Ibid. [↑](#endnote-ref-60)
61. Ibid. [↑](#endnote-ref-61)
62. Dr Rosenbaum [↑](#endnote-ref-62)