Project Management Plan

Hayden Eubanks

September 1, 2023

Table of Contents

Section/Sub-section Title
1. Deliverables, Non-deliverable Products, and Milestones
2. Project Schedule
3. Risk Assessment

1. Deliverables, Non-deliverables, and Milestones

Deliverable Name	Description
Project Management Plan [PMP]	• **The Project Management Plan (PMP) provides an organized structure to the software development lifecycle (SDLC) and outlines the requirements that must be met in the deliverable product (Tsui, Karam, & Bernal, 2016).** • **Outlining requirements in this way allows for the communication of shared goals among the development team and product stakeholders and further, defines a metric by which the measurement of project completion and success can be evaluated (Jadhav et al., 2022).** • **The PMP then allows for the evaluation of the resources required to achieve project completion such as technical considerations, manpower, and the time needed for completion (Tsui, Karam, & Bernal, 2016).** • **From this, a preliminary risk assessment can be performed which highlights potential risks the project may encounter throughout development and allows for those risks to be addressed and mitigated before they are encountered (Tsui, Karam, & Bernal, 2016). Further, a thorough risk assessment can prepare a development team for responding to disruptive events therefore decreasing the negative effects incurred from disruptions.** • **Once these factors are identified, the PMP can then be used to evaluate development strategies such as traditional or Agile methodologies and a method can be selected that best meets the goals of the development team and stakeholders (Leong et al., 2023).**
Systems Requirements Specification [SRS]	• **With the project plan outlined in the PMP, the Systems Requirements Specification (SRS) can then be completed to outline the specific requirements the project must fulfill (Tsui, Karam, & Bernal, 2016).** • **This includes the defining of non-functional requirements (NFRs) that outline project objectives as well as functional requirements that detail specific implementations to meet the goals of the NFRs (Merkow & Raghavan, 2012).** • **NFRs implemented in this manner can address development criteria such as security, aesthetics, usability, effectiveness, and any other objective that must be present in the deliverable product (Merkow & Raghavan, 2012).** • **The functional requirements then outline the specific implementation that will be performed to accomplish the objective of an NFR such as specific controls that must be implemented for security, a help system to assist the user in use, and the minimum performance threshold that must be met in the final product (Merkow & Raghavan, 2012).**

	• **System requirements can be generated from needs expressed by the project stakeholders as well as from user stories that model specific consumer use cases the project would like to accommodate (Hartson & Pyla, 2012).** • **In this way, the SRS can assist in directing the development process by steering development toward the fulfillment of requirements that must be present in the deliverable product.**
Systems Design Specification [SDS]	• **The System Design Specification (SDS) then extends from the requirements outlined in the SRS to a design specification by outlining the way that the functional requirements are to be implemented in the product (Tsui, Karam, & Bernal, 2016).** • **The SDS can generate models for implementation such as UML diagrams that model relationships between functional components of the product allowing developers to structure their design and modularly develop system functionalities (Jadhav et al., 2022).** • **As the SDS addresses the implementation of system requirements, the SDS can also address user interface implementations and provide a guide to the flow of interface interaction as well as specify interface component implementations (Hartson & Pyla, 2012).** • **Having specifications for design can then allow for a collaborative understanding of the system's design and allow the development team and stakeholders to understand the structure of the product to be delivered.**
Systems Test Specification [STS]	• **Once a workable deliverable of the product is obtained, the product must be tested to ensure that the product meets the outlined system requirements as well as user and stakeholder satisfaction (Tsui, Karam, & Bernal, 2016). This can be accomplished through a System Test Specification (STS) that outlines the tests to be performed as well as the measurable objectives those tests must achieve.** • **The STS refers to the objectives set in the system requirements and then outlines tests that can be performed to measure the level of achievement of those requirements in the iteration of the product being tested (Tsui, Karam, & Bernal, 2016).** • **Factors outlined in the STS can address factors of testing such as specific tests to be performed, when those tests should be carried out, who the test should be administered to, and what metrics should be measured during the tests (Tsui, Karam, & Bernal, 2016).** • **Tests can be issued during the development phase to test the suitability of each software component by measuring target metrics against the STS. This then allows developers to understand if the current implementation satisfactorily meets the outlined requirements for the deliverable product.** • **Further, the STS can outline target groups that will be issued tests of deliverables in testing for consumer usability and satisfaction (Hartson & Pyla, 2012).**

	• **Performing tests according to the STS then allows the development team to communicate the status of deliverables to stakeholders in terms of meeting the outlined requirements and in doing so an assessment of project completion can be performed.**

Non-Deliverable Name	Description
Design Prototype Iterations	• **Before resources are allocated to the actual creation of a design concept, prototypes of the design to be implemented can be crafted and evaluated honing the parameters of the chosen design (Tsui, Karam, & Bernal, 2016).** • **This process could include the development of UML diagrams, storyboards for interfaces, and aesthetic design prototypes that can be discussed among developers for improvement (Hartson & Pyla, 2012).** • **While this is a non-deliverable used by the development team, members of the consumer population or stakeholders could be included in this process to ensure potential designs are on track to fulfill the goals of the project before resources are allocated to a design.**
Interface Prototype Iterations	• **Interface prototypes are then created as drafts of the software interface developed to assess usability, aesthetics, and functionality (Hartson & Pyla, 2012).** • **Through iteratively examining and documenting interface prototypes, developers and project managers can better assess the suitability of an interface implementation and change the direction of these implementations if needed.**
Consumer Interviews (Pre-Requirements)	• **Many software applications seek to perform a specific task or simplify and automate a problem facing a consumer group (Hartson & Pyla, 2012). Performing customer interviews regarding the task to be automated can greatly assist the development team in identifying system requirements, and in doing so, set the direction of the project development.** • **The interview transcripts collected as artifacts from this process will allow the development team to identify consistent themes in consumer sentiments and in addressing user feedback promote a user-centric design philosophy (Hartson & Pyla, 2012).**

Consumer Interviews (Prototype Testing)	• **Interviews can then be performed again amongst a consumer group once prototypes have been developed to assess the effectiveness and usability of a prototype iteration. These interviews could assist the development team in identifying design flaws earlier in the development lifecycle and in doing so allow for the correction of these flaws when it is less costly to correct (Tsui, Karam, & Bernal, 2016).** • **These interviews can be administered to varying target groups or with different variations of prototype design to further assess the suitability of specific implementations.**
Task Observation Report	• **In line with the pre-requirement user interviews, observations of the user performing the task to be automated can identify hard-to-articulate elements of design that could increase the experience of product consumers (Hartson & Pyla, 2012).** • **As an external observer, there may be blind spots in usage that a consumer struggles to identify and further allows for the identification of niche scenarios or exception cases that must be addressed in the product implementation.** • **Approaching design from a user-centric perspective can allow for a product to be designed in a manner that more closely mirrors the way that users interact with a given task, therefore increasing the likelihood that the deliverable product is relevant to the end consumer (Hartson & Pyla, 2012).**
Affinity Board	• **Affinity boards are a creative tool for developers to visualize the relationships between requirements and in doing so determine project scope and commit to requirements for project development (Hartson & Pyla, 2012).** • **This tool allows developers to organize all potential requirements and ideas without the bias of perceived scope to allow the creative process to thrive and then in turn gives a tool to accomplish scope narrowing effectively.** • **Further, Affinity boards allow for quick sketch prototyping where ideas can be presented in rudimentary sketches to visually demonstrate ideas or concepts (Hartson & Pyla, 2012).** • **The affinity board can then be maintained as an artifact for the design process where developers can look to for visualizing requirements concerning each other.**
User Stories	• **User stories are another method of visualizing requirements which function by placing a requirement within the context of a consumer of the deliverable product (Tsui, Karam, & Bernal, 2016).** • **These user stories paint an image of a representative user of the product giving that user a name and identity (Hartson & Pyla, 2012). A use case or problem for that user is then described as it would relate to a system requirement.**

	• **User stories assist in allowing a developer to consider system requirements within the context of the usage of the end product and in doing so further the principles of user-centric design.**
Periodic Progress Evaluation Reports	• **Throughout the SDLC it is essential for the development team to evaluate development progress in the context of the timeline initialized in project planning (Tsui, Karam, & Bernal, 2016).** • **Evaluating progress in this way can allow for projections as to updated project costs and timeframes and can allow for project management decisions such as decreasing scope or growing the development team before doing so would negatively affect the project (Tsui, Karam, & Bernal, 2016).**
Notes on Takeaways from Daily Meetings	• **Regular development team meetings are an essential part of agile development methodologies and allow the development team to understand the progress they are making and where more help is needed to fulfill objectives (Leong et al., 2023).** • **Documenting these meetings through light note-taking summaries can serve as an artifact for project managers to evaluate progress and identify areas where extra help is consistently needed in development.**
Detailed Minutes from Milestone Progress Evaluations	• **In addition to the light notes from daily meetings, detailed minutes and meeting breakdowns from the milestone meetings can serve as a tool for documenting development objectives and progress.** • **When a project milestone has been accomplished, the next milestone can be looked forward to and the documentation from these meetings can serve as a reference for outlining the path to the next milestone for the development team.**
Bug Documentation	• **An essential aspect of product testing is the accurate documentation of software bugs that can then be passed back to the development team for addressing (Hartson & Pyla, 2012).** • **Thorough documentation of bugs can make it easier for developers to replicate the bug and identify solutions (Tsui, Karam, & Bernal, 2016) highlighting the importance of generating quality bug documentation during the testing phase of development.**
Quality Testing Evaluations	• **Similar to bug testing, quality testing can be performed and documented to highlight flaws in qualitative elements of a product's design (Hartson & Pyla, 2012).** • **Accurately documenting these flaws can allow the development team to adjust design implementations and ensure the deliverable product meets the desired quality standards.**

Periodic Risk Assessment	• **As the SDLC progresses, it can be beneficial to reevaluate risks facing development and ensure that risks continue to be properly mitigated (Tsui, Karam, & Bernal, 2016).** • **The documentation of these risk assessments is essential as it provides a means of comparing the development of risks and risk prevention.** • **Further, documentation can outline response procedures to risk events or usage policies that mitigate damages to development from a security event (Jadhav et al., 2022).**

Milestone Name	Description
Receive Task from Stake Holders and Commit to Task Fulfillment	• **The first step in the SDLC involves receiving the task from the stakeholders and reaching an agreement with the stakeholders as to the task to be fulfilled (Tsui, Karam, & Bernal, 2016).** • **The first milestone is then achieved when an agreement between the project stakeholders and development team managers is met.** • **At the end of this milestone, the development team will then have committed to producing a deliverable product that fulfills the agreed-upon task.**
Outline Project Development and Create a Development Strategy	• **With the responsibility of the agreed-upon task, the development team is then able to develop a strategy for approaching the SDLC of the project.** • **This involves drafting a PMP and committing to a design methodology as this will vastly impact the remainder of the development process.** • **The proper structuring of the development process during this stage will then greatly improve the quality of the SDLC and allow developers to understand where they currently are in the process.** • **The end of this milestone will then be marked by the completion of the strategy and is the final step that must be performed before the project task can be approached.**

Perform and Document Task Analysis	• **Before generating requirements or beginning to develop the project, task analysis must first be performed (Hartson & Pyla, 2012).** • **Task analysis involves researching the task to be performed by the product, observing consumers interacting with this task, performing consumer interviews, and documenting these processes.** • **The end of this Milestone will then generate several artifacts that will be passed to the next phase for determining system requirements.**
Generate and Commit to System Requirements	• **Using the artifacts created in task analysis, the development team is then able to create and agree upon system requirements to be implemented in the deliverable product.** • **During this phase, several non-functional requirements will be created with functional requirements under them to implement the non-functional requirements.** • **These requirements can be organized into an easy-to-read structure and classified according to importance making it easy for the development team to focus on the most important elements first (Hartson & Pyla, 2012).** • **Further, an SRS document can be created in this stage to guide the remainder of the development process.** • **This milestone will then be reached when all of the requirements are set and ready to pass to the stakeholders for their agreement.**
Gain Agreement from Stake Holders to System Requirements	• **The requirements created by the development team are passed back to the stakeholders during this step and the stakeholders are then able to approve the requirement list or pass the requirements back to the development team for revision.** • **As the system requirements guide the development process (Tsui, Karam, & Bernal, 2016), setting sufficient requirements is essential and this process may be iterated until both the developers and stakeholders agree on a set of requirements to be implemented in the deliverable product.**
Set Implementation Specifications	• **With the requirements for the system set, an implementation model can be drafted that models the implementation of the requirements in the deliverable product. This is one aspect of what will make up the SDS and is essential in setting before development begins to give direction to development.** • **This milestone can be performed in conjunction with the interface specification which will form the other part of the SDS.**

Set Interface Specifications	• **In addition to setting specifications for implementing the requirements, an understanding of the interface elements used to accomplish that implementation must also be developed.** • **This milestone will mark the last planning phase that must be accomplished before the actual system development begins.**
Develop Prototype	• **During this phase, a draft of the product is developed according to the planning phases previously accomplished.** • **This phase may be iterated over several times depending on the reception of the prototype but can also be performed to test implementations of modular components of the bigger system (Tsui, Karam, & Bernal, 2016).** • **When a workable prototype is generated, it can then be passed to the testing teams for feedback marking the next milestone phase.** • **During this phase, a STS can also be developed to outline the way that the prototype will be tested.**
Prototype Testing	• **During the prototype testing phase, the prototype is tested by the development team and if appropriate shown to consumer groups for feedback and observation.** • **Depending on the stakeholder involvement, it may be that the prototype is also shown to stakeholders during this phase to gain approval of the direction that development is heading in.** • **This milestone will be repeated for as many prototypes that are generated until the final draft is set.**
Refactor Design	• **This milestone represents the iteration of the prototype development and testing phases.** • **The end of this milestone will then be marked by a design that does not need to be refactored and is ready to pass forward for further debugging.**
Debugging	• **With an implementation set in a final draft, the final draft can then be debugged to ensure that the quality of that draft meets the metrics outlined in the system requirements (Hartson & Pyla, 2012).** • **This milestone will then be completed when the project managers and stakeholders agree that the quality targets have been met and the full requirement set is then ready to be evaluated.**
Assess Sufficient Requirements Have Been Met	• **During this phase, the development team validates that the product has met all of the requirements and is ready to hand back to the stakeholders.** • **At this time if any requirements are still unmet, justifications as to their unfulfillment can be drafted and passed to the stakeholders who will assess satisfaction with the state of the final product.**

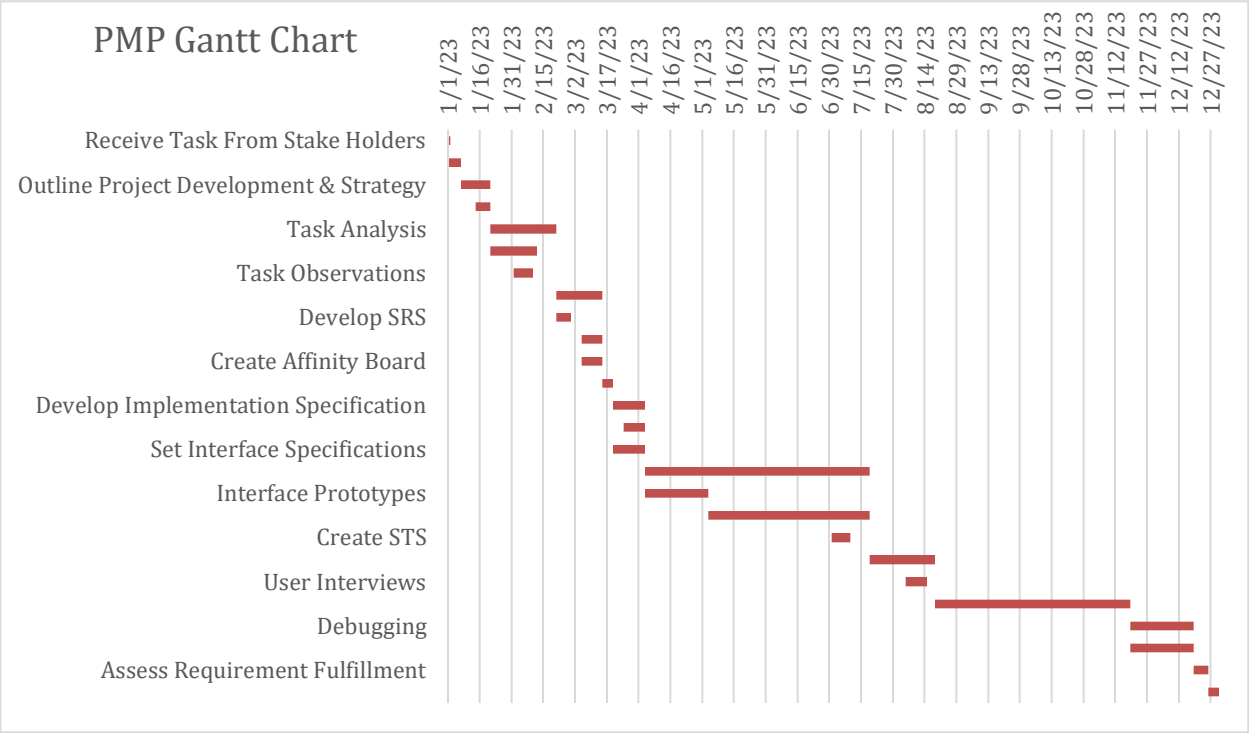
Gain Approval from Stake Holders	• **This milestone is then marked by the stakeholder's approval that all of the agreed-upon requirements have been met and that they are happy with the final deliverable product.**
Deliver Final Draft to Stake Holders	• **The product is then shipped to the stakeholders who can distribute the product to the consumers who will engage with it. Marking the end of product development.**

2. **Project Schedule**

<u>List of Schedule Events and timeframes</u>

1. Receive Task from Stakeholders [January 1]
2. Commit to Task Fulfillment [January 1st-7th: One Week]
3. Outline Project Development and Strategy [January 7th- January 21st: Two Weeks]
    a. Create PMP
4. Task Analysis [January 21st- February 21st: One Month]
    a. Consumer Interviews
    b. Task Observation
5. Develop System Requirements [February 21st- March 15th: 2.5 Weeks]
    a. Develop SRS
    b. Create User Stories
    c. Create Affinity Board
6. Gain Stake Holder Agreement [March 15th- March 20th: Five Days]
7. Develop Implementation Specifications [March 20th – April 4th: Two Weeks]
    a. Create SDS
8. Set Interface Specifications [March 20th – April 4th: Two Weeks]
   (Simultaneous with task seven)
9. Develop Prototype [April 4th – July 19th: 3.5 Months]
    a. Interface Prototypes
    b. Implementation Prototypes
    c. Create STS
10. Test Prototype [July 19th – August 19th: One Month]
    a. User Interviews
11. Refactor Designs [August 19th – November 19th: Three Months]
12. Debugging [November 19th- December 19th: One Month]
    a. Quality Testing
13. Assess Requirement Fulfillment [December 19th – December 26th: One Week]
14. Gain Approval and Deliver Product [December 26th- December 31st: Five Days]

Gantt Chart of Project Schedule



3. **Risk Assessment**

Risk Identification

Risk #	Type	Description
1	Risk of Going Over Schedule	• This risk defines the possibility for either an element of the project or the project as a whole to go over schedule. • In Software development, it is essential to return deliverables within a timely manner and the risk of going over schedule could have major implications on project costs or could negatively impact the stakeholders.
2	Risk of Going Over Budget	• Going over budget is another risk related to the inadequate estimation of required resources. • This improper estimation could result in the cost for project completion being much higher than initially intended and could occur as a result of going over schedule or needing to hire more developers.
3	Risk of Data Destruction	• This risk pertains to the risk of data being corrupted or destroyed due to natural disasters or cyber-attacks. • This risk can be mitigated by implementing strong security controls that protect the confidentiality, integrity, and availability of data (Basta, 2018).
4	Risk of Team Conflict	• A team that is not working well together could result in a rise in time or cost elements if work is disrupted or new team members are needed.

		• Communication is an essential element of a development team and the disruption of communication through team conflict could negatively impact a project.
5	Risk of Insufficient Skills among Development Team	• In agreeing to a task there is a risk that the development project managers have overestimated the skills of their team. • If a team is ill-equipped to fulfill a development progress then the time and cost needed to complete the project could increase drastically.
6	Risk of Requirements Changing	• The later that additional requirements are introduced into the project, the more expensive the implementation of those requirements will be (Tsui, Karam, & Bernal, 2016). • This introduction of new requirements could be representative of a poor initial gathering of project requirements or poor communication between project managers and stakeholders.
7	Risk of High Volume of Bugs	• If the quality of the initial code is poor then the scope of the debugging process could increase, increasing both the time and cost of development. • In the worst case, this could also involve bringing in new developers to assist in the debugging process.
8	Risk of Legislation Change	• Depending on the nature of the project, legislation changes could vastly impact the project requirements or implementation strategies.

		• This risk can be mitigated through understanding the sphere that is being developed for and tracking changes within that sphere.
9	Risk of Vulnerability Being Discovered Within Design	• If a vulnerability is discovered within the project code or a function that the project relies heavily upon, then the time and cost to eliminate that vulnerability could be high. • Through practicing security-centric design this risk can be vastly mitigated, and projects made more secure (Basta, 2018).
10	Risk of Not Meeting Stake Holder Expectations	• If the stakeholders are not happy with the direction or amount of progress then they could choose to terminate the project or send the project back for further development. This could increase the cost and time of the project.
11	Risk of Rejection from Consumers	• This risk pertains to the post-release phase of development where consumers could reject the product. • This risk can be mitigated through user-centric design practices and including user feedback throughout development (Hartson & Pyla, 2012).
12	Risk of Inadequately Defined Project Scope	• If the project scope is inadequately defined, then the development team may waste time and resources developing elements of the project outside of the desired scope. • Further, the opposite could also be true where the development does not cover all of the desired scope of the

		project and more time and resources must be spent to rectify this mistake.
13	Risk of Inadequate Development Resources	• The inadequate provision of resources to fulfill a project could result in higher costs as resources need to be purchased or project termination if the cost to gain these resources is too high.
14	Risk of Implementing an Ineffective Strategy	• If a development strategy that is not effective for a given project is implemented, then the project may not meet the desired standards and satisfaction of the stakeholders. • This could also result in ineffective development therefore increasing costs and timeframe for completion.
15	Risk of High Developer Turnover	• The high turnover of developers could result in the project going over time and budget as new developers need to consistently be trained and brought up to speed with the project's status.
16	Risk of Poor Code Performance	• Code that generates poor performance in the end product may need to be developed upon further increasing the costs and timeframe for project completion. • Further, this could result in dissatisfaction from the stakeholders or consumers if they encounter the poor-performing code.

17	Risk of Vulnerabilities Missed in Testing	• Any vulnerabilities missed in testing could result in more costly solutions down the line. • This risk is especially vital if the exploit is discovered by a malicious actor who exploits the vulnerability as part of a cyber-attack.
18	Risk of Poor Documentation	• The poor documentation of code or bugs could make the code difficult to maintain and this could result in further costs or time needed to complete the project.
19	Risk of Requirements Extending Beyond Scope	• If the project requirements are not thoroughly examined before the project has begun, then the scope of the accepted requirements could extend well beyond the intended scope of development. • This could cause tension between the development team and stakeholders or added costs as the project must be expanded.
20	Risk of Poor Communication Between Stake Holders and Development Team	• The lack of communication between the developers and stakeholders could result in the development process moving in a direction that does not suit the stake holder's needs. • Further, it may be that the stakeholders do not communicate well what it is they would like to accomplish through the project, and in this, the delivered product may not suit their needs.

Risk Analysis [Describe why you believe the ***probability*** and ***effect*** are as you state]

Risk #	Probability & Reason	Effect & Reason
1	**Going Over Budget**: High- • Along with the risk of going over schedule, the risk of going over budget is extremely high as many of the other risks contribute to added costs if they occur. • Further, initial estimates are difficult to make and poor estimates could easily result in the budget overrunning.	Tolerable (Severe)- • The effect of this risk depends on the risk appetite of the stakeholders and the amount that the budget is over, but in general, some amount running over budget is tolerable. • However, if the budget were to run over by a large margin then this risk could easily result in severe negative effects.
2	**Going Over Schedule**: High- • Similar to going over budget, the risk of going over schedule is extremely likely as many of the other risks contribute to added development time. • Further, time estimates are dependent upon stakeholder acceptance and the development of prototypes could extend longer than initially predicted as it can be difficult to predict time to an acceptable rendition.	Tolerable (Severe)- • In general, added development time is a tolerable risk, but this could easily extend to severe as development time is closely linked to costs. • If the development time extends beyond the amount that the stakeholders can fund, this risk could be classified as severe and have serious project implications.

3	**Poor Documentation**: High- • At some point in the system's development, it is very likely that poor documentation, or the lack thereof is encountered. • This risk is team-dependent, and a strong development culture of documentation could reduce this risk's likelihood but in general, there is a high likelihood that poor documentation will be encountered at some point.	Tolerable- • As with the first two risks, this risk is tolerable in small quantities but as it increases in prevalence the risk effects also increase. • However, this risk is classified as purely tolerable as the development could still continue, it is just in maintenance that problems could be more severe.
4	**Ineffective Strategy**: Moderate- • Choosing a development strategy could be difficult, especially for inexperienced project managers resulting in a moderate risk of occurrence. • Further, within a company that has a strong culture of a particular development strategy the development strategy implemented may not be the best fit for the project at hand.	Severe- • This risk could have serious implications for the project as a poor implementation strategy can affect every element of the development process. • All inefficiencies contribute toward added costs and development time making this risk severe if it were to occur.

5	**Missed Vulnerabilities**: Moderate- • Determining the chances of a vulnerability being discovered after the product is delivered is difficult, but as vulnerabilities are constantly being discovered there is a moderate chance it could occur. • However, the chances of this risk occurring are proportional to the degree that the development process implemented security-centric design principles (Basta, 2018).	Catastrophic- • If a vulnerability is initially discovered by a malicious actor who exploits the vulnerability, then the results could be catastrophic. • This exploitation could cause actual harm to the consumers depending on the nature of the exploit highlighting the importance of practicing security-centric design to ensure this doesn't happen.
6	**Poorly Written Code**: Moderate- • This risk is directly proportional to the skills and care taken by the development team and as the skill range of developers within a team varies, a moderate risk level was chosen. • In addition to this, certain portions of code could be poorly written and if a critical section of code is poorly written the effect could be more prevalent.	Severe- • The effects of this risk depend on what sections of the code are poorly written as well as what percentage of the code. • Poorly written code can have severe impacts on software performance and may even need to be rewritten. For this reason, this risk has been ranked as severe if it were to occur.

7	**Team Conflict**: Moderate- • Within any social environment team conflicts and tensions are bound to arise. • While conflicts are common it is occasionally that these conflicts can disrupt team communications and even workflow and there is a moderate chance of this occurring.	Tolerable- • Team conflicts are issues that in general can be resolved through good management and communication. • For this reason, while it is likely that team conflicts will occur it is in general a tolerable risk to accept.
8	**Vulnerability Within Design**: Low- • When vulnerabilities are discovered within a particular function the function needs to either be patched or deprecated to ensure the integrity of the underlying code. • The chances of this happening during development are low, but it is good for the development team to be aware of the security implications of the code they are using as well as changes in the security landscape.	Severe- • If a function that is widely used throughout the code is found to be vulnerable, then the efforts to revert the code to a safe state could be extremely demanding. • This would have serious implications for the development process and could vastly increase the time and cost of development.

9	**Legislation Change**: Low- • The chances of legislation passing affecting the development process are dependent on the field being developed, but in general, the chances of this occurring are very low. • Industries prone to legislation change may classify this risk differently but	Severe- • If a legislation change were to occur, it could result in the project needing to rapidly adapt and potentially large portions of the code may need to change. • This would have major implications on product costs and the timeframe to release indicating severe implications if this risk were to occur.
10	**Data Destruction**: Low- • Data integrity or availability could be disrupted through events such as natural disasters, hardware failures, or cyber-attacks, and in extreme cases that data may be irrecoverable. • Often, redundancy tactics are employed to ensure the availability of data but in the case where data is destroyed, the results could be catastrophic.	Catastrophic- • In the case where system data is destroyed, it could be catastrophic to development, stopping development entirely or resulting in large portions of code needing to be rewritten. • This would result in an extreme increase in costs and development time proportional to the amount of data that was lost.

Probability [High, Moderate, Low]
Effect [Catastrophic, Severe, Tolerable, Insignificant]

Risk Planning [in terms of ***avoidance, minimization,*** and ***contingency***]

Risk #	Strategy
1	To avoid going over budget, great care can be taken in the estimation phase to think through potential costs and account for them in the plan. Further, routine evaluations can assist in tracking costs and understanding if changes need to be made to ensure the budget is kept. Good project management can greatly assist in practices that maintain a set budget and this will allow for the quick understanding of when and if the budget is off track. This alertness to budget variance can mitigate the damages of a growing budget and can allow for budget correction when needed.
2	Similarly, project schedules can be maintained through good project management and the risk of the schedule going over decreased. In addition to this, ensuring quality is prioritized in all levels of design can ensure that a product achieves acceptance quickly and less time needs to be spent correcting mistakes later on in development.
3	The risks associated with poor documentation can be mitigated by developing a culture of documentation and holding developers accountable for documenting changes they implement. This will vastly assist in the maintenance of the product and decrease the time in onboarding new developers.
4	Taking the time to observe the task to be developed and understanding the task requirements can greatly assist in choosing an effective implementation strategy. Certain strategies are better fit for a task than others and understanding the benefits of each implementation can allow for the evaluation of different strategies and choosing one that best fits the task. Implementing an effective strategy can optimize the SDLC highlighting the importance of choosing an effective strategy.

5	While it is impossible to eliminate unknown vulnerabilities, steps can be taken to mitigate the damages when they occur. This includes practicing security-centric design concepts and ensuring that defense-in-depth philosophies are applied. Practicing security in this way will help ensure that if a vulnerability is discovered the damages are minimal.
6	Poorly Written code can be mitigated by creating a culture of quality and setting standards of control within the development process. Further, principles of accountability can be practiced within the development team to ensure that developers have the support they need to develop quality code and that a high standard is achieved.
7	Team conflicts can be mitigated through proper communication and good team management. Many conflicts result from miscommunications and by ensuring a culture of communication and support, team conflicts and the risks associated with them can be mitigated.
8	By thoroughly understanding the functions implemented within code, developers can lower the risks of implementing vulnerable functions within their code. In addition to this, understanding the task being solved and the potential solutions can allow for the selection of a solution that best fits the task in terms of effectiveness and security.
9	Legislation changes are often unavoidable, but being aware of current events within a sphere can assist in preparing for potentially coming legislation changes. Staying ahead of changes such as this can mitigate the risk of unusable code due to legislation changes.

10	Implementing strong data redundancies can mitigate the potential damages from hardware damage or cyber-attacks. Ensuring that data is available is essential to development highlighting the importance of implementing strategies to ensure that data remains available despite an incident occurring. Planning strategies for when an incident occurs can allow for a quick bounce back and ensure that development disruptions are minimal.

Risk Monitoring [whether the risks are more or less probable and how they might have changed]

Risk #	Indicator(s)
1	Keeping accurate records of spending can allow for the early and easy-to-detect indicators that the budget is not being kept. When actual spending exceeds budgeted values, the budget is not being kept and these indicators can be acted upon to get the budget back on track.
2	Similar to budget tracking, the development schedule has assigned start and end dates for each task, and task progress can be measured against these dates to evaluate how well the schedule is being kept. These evaluations can provide an early indicator that a schedule may be falling behind and allow for a rapid response to attempt to get back on track.
3	Poor documentation may be difficult to detect within a culture that does not check for documentation. Developing a culture of keeping documentation up to date and checking documentation can facilitate an environment where code is easy to maintain and understand. A strong indicator of poor documentation can be observed through the occasional auditing of documentation to ensure that it is being maintained properly.
4	An effective implementation strategy can greatly complement the design process but an ineffective strategy for a given task could be difficult to detect or correct if it is not discovered in the early phases of planning. Despite this, constant reviews of the development philosophy and progress can allow for an evaluation of whether a better strategy should be implemented. Changing development strategies can be difficult to accomplish and the benefits of making a change should be weighed against the opportunity costs of implementing that change.

5	An unknown vulnerability may be difficult to detect before it is discovered, but once it is discovered and reported, the indicator becomes quite clear. These indicators should be acted upon immediately when discovered and the vulnerabilities mitigated through patch management.
6	Indicators of poorly written code can be observed in the code performance and slow or overly buggy code can be a clear indicator that code is not being written to a high standard. Creating a culture of quality can also help in discovering poorly written code and a peer review process can work towards identification and correction of poor-quality code before it causes problems.
7	Good communication between the project managers and developers can allow for the quick identification of team conflicts. Further, this communication can allow for the addressing of conflict issues before they reach a point where they affect the project's performance.
8	Keeping up to date with vulnerabilities in commonly used functions and dispersing this information among the development team can allow for the identification of vulnerabilities existent within code and can also establish a peer review process of editing these vulnerabilities when they are detected. Further, the development standards can identify preferred functions for certain tasks that promote security and mitigate the risk of vulnerable code making it into production.
9	Staying current on developments within the development sphere can allow for the quick identification of potential legislative changes that may be approaching. Once a potential change is detected changes can be made proactively to ensure that development is not disrupted when the changes come into effect later down the line.

10	Keeping track of potential threats to infrastructure such as coming storms, local fires, and the age of technical components can allow for the mitigation of data loss before it occurs. If data loss does occur it will be quickly evident as the data will no longer be accessible and the recovery plan can immediately go into effect.

References

Basta, A. (2018). *Oriyano, cryptography: Infosec pro guide.* McGraw-Hill Education. https://bookshelf.vitalsource.com/reader/books/9781307297003/pageid/14

Hartson, H. R., & Pyla, P. S. (2012). *The UX book: Process and guidelines for ensuring a quality user experience*. Morgan Kaufmann. https://doi.org/10.1016/C2010-0-66326-7

Jadhav, S., Shinde, S., Ghuge, V., Godse, D., Golsangi, M., & Harne, P. (2022). A software development lifecycle case study on: Diet recommendation system based on user activities. *ITM Web Conferences; Les Ulis 50*. https://doi.org/10.1051/itmconf/20225001009

Leong, J., May Yee, K., Baitsegi, O., Palanisamy, L., & Ramasamy, R. K. (2023). Hybrid project management between traditional software development lifecycle and agile based product development for future sustainability. *Sustainability (Basel, Switzerland), 15*(2), 1121. https://doi.org/10.3390/su15021121

Merkow, M. S., & Raghavan, L. (2012). *Secure and resilient software: Requirements, test cases, and testing methods* (1st ed.). CRC Press. https://doi.org/10.1201/b11317

Tsui, F, Karam, O., Bernal, B. (2016). *Essentials of Software Engineering* (4th ed.). Jones & Bartlett Learning. https://libertyonline.vitalsource.com/books/9781284129755