Risk Management

JD Kilgallin

CPSC:480

11/14/22

Pressman Ch 18, 26

Notes

- Quiz 7 was mostly good. Top issue: Cyclomatic complexity is the minimum number of test cases needed to achieve 100% code coverage.
- Project 3 checkpoint was mostly good. For the rest of the semester, you should continue using pull requests to merge code from your branch, but you may merge with just one teammate's approval after 48 hours if there are no outstanding comments to address.
- Project 3 last call 11:59 PM tonight.
- New team project survey posted. Less finger-pointing and more assignment feedback questions. Submit on Brightspace.
- Project 4 is out now; first checkpoint Tuesday, Nov 22; second checkpoint Sunday, Nov 27; presentation Monday, Nov 28; final report Friday, Dec 2.
- Wednesday's lecture will be from Mikyla Wilfred, Keyfactor Product Owner and UI/UX engineer. Practice final will be out; due Monday, Nov 21 in class.

Project 4 Scenario

- All teams have just been acquired by my company “EdgeCase Solutions, LLC” for further development next semester.
- As part of the acquisition, I will be restructuring the organization for efficiency. This means that in 30 days, all six teams will be merged to work on just one project for another five months, abandoning the other software projects.
- There may also be layoffs of Twitter-like proportions, so you will need to convince me that you're worth keeping on
- The only part of the scenario that isn't real is that development will continue past 30 days, but this framing is meant to provide a concrete target for your remaining project work.

Project 4 Qualities to Demonstrate

- A well-defined and appropriate software development process and evidence that they've followed it
- Thorough product testing and documentation
- Quality as a priority consideration throughout the whole project
- The agility to adapt to changing requirement priorities
- A vision for what the software could do in the next six months and beyond
- Compelling arguments for the likely success of their project
- Dedication from every member, positive team dynamics, and mutual trust
- Honest assessment of strengths & weaknesses and a plan to improve
- The ability to convince other engineers of the project's potential

Project 4 Part 1

- Assess your requirement priorities and goals, and determine what's most important for making the best prototype demo on Nov 28, keeping in mind that the project with the best chance of success will have another five months of development to complete a first released version.
- Add and modify detail to the related tasks so that the work can be assigned and completed. Refer to your project 3 velocity calculations to determine what can be completed in one week.
- Identify what went well and what went poorly on the previous sprint and what changes can be made to improve for this one.
- Identify the major risks to completing the work planned for the demo this semester, as well as the top risks to the project if selected for next semester. Use the fields in the "Risk Management" slide below.
- Keep notes from this discussion and submit them in part 3.

Project 4 Part 2

- Software should be "code complete" for demo purposes by Nov 22 (next Tuesday).
- You will need to carefully manage bug fixes and changes after this point to prevent introducing new bugs in the build you demo. Follow the code review and approval process.
- Create a release in your GitHub repo when your software is code complete. This is your checkpoint 1 submission.
- It isn't necessary for everyone to contribute code, but you should continue to track time spent, keep repository issues and tasks up to date, and follow all the steps in your engineering process.

Project 4 Part 3 Overview

- Write a report explaining and defending your software engineering process and project management activities, along with a few additional elements (e.g. function with highest cyclomatic complexity)
- Ensure your repository contents are appropriate and demonstrate a consistent process and work tracking.
- Submit the report to your repository and create a PDF release by Sunday, Nov 27, 11:59 PM for checkpoint 2.
- You will use this content for the primary contents of your project presentation in part 4.

Project 4 Part 3 Report Core Elements

1. A summary of the code quality and technical debt in your project.
2. A description of how you've followed a software engineering process effectively or cases where you haven't followed the process.
3. A project plan and timeline for how your product could be launched with another semester or less of work and what resources would be required.
4. An overview of the codebase and repository aimed at engineers who might be joining your project.
5. Major risks identified for the project over the course of the next semester. Use a risk assessment form or follow the format at the end of Risk lecture.

Project 4 Part 3 Report Additional Elements

6. Part 1 meeting notes
7. Change control process documentation from part 2
8. Product documentation describing operation of your product.
9. Technical documentation on building, testing, and installing the product.
10. Identification of the function in your code with the highest cyclomatic complexity, what the value is, and how it could be refactored or why it’s okay as is.
11. A description of how your project uses or could use automation in the software development process.

Project 4 Part 3 Repository Contents

1. Work items with task lists, original estimates, and comments with time taken to implement and test.
2. Bug reports for all known issues in your product
3. Clean commit history and repository organization
4. Revision history of all artifacts
5. Pull requests and code reviews for changes

Learning objectives

- Risks associated with software engineering
- Risk factors
- Risk management concepts

What is risk?

- Noun or verb
- "Exposure to harm or danger" -OED
- "[Cause the] possibility of loss or injury" -Mirriam-Webster
- "[Create] a condition with the potential to cause an adverse outcome" -me
- Note that "exposure", "danger", "possibility", and "potential" all involve *uncertainty.* There is no risk with 100% certainty; something that is *guaranteed* to cause an adverse outcome is just an unfortunate circumstance or constraint. Conversely, no risk has 0%; it must be *possible.*
- Once a risk materializes, it reaches 100% chance of occurring, and ceases to be a risk. Thus, risk only describes *future* states. "harm, "danger", "loss", "injury" and "adverse [outcome]" all indicate *undesirable* future states.
- These are the key aspects of a definition of risk: a future state that is neither impossible nor guaranteed, but preferable to avoid.

Types of Risk in Software Engineering

- Project Risks – Threats to the project plan. That is, factors that may cause the software to not be deliverable on time and on budget.
- Technical Risks – Threats to the software design. Implementations that turn out to be more challenging than anticipated can make it more difficult or even impossible to realize software requirements.
- Business Risks – Threats to the development team. This includes:
  - Market Risk – Building a product nobody wants. Leads to project termination.
  - Strategic Risk – Building a product that no longer fits into the vendor's business strategy (reorganizations/M&A, business pivots, new opportunities)
  - Sales Risk – Building a product that is hard to sell due to complexity.
  - Management Risk – Losing support from a senior executive or key stakeholder
  - Budget Risk – Vendor becomes unable to allocate resources to the project.

Risks by Forecasting Potential

- Known Risks – Evident from analysis of project plan & other artifacts.
  - Insufficient or ambiguous requirements
  - Technical uncertainty
  - Doubt in usability of the product as designed
  - Personnel and expertise uncertainty
  - Impractical budget or timeline
- Predictable Risks – Extrapolated from past project experience.
  - Staff turnover, absence, and reassignment
  - Changing requirements and scope creep
  - Wavering stakeholder commitment
  - Quality Assurance failures
  - Security vulnerability exploits
- Unpredictable Risks – Novel threats and "Acts of God" that cannot easily be modeled, mitigated, or planned for. Even if they are unlikely, they do occur.

Project Risks by Associated Source

- Product Size – Scale & scope of requirements and codebase.
- Business Impact – Priority to the vendor's and customer's businesses.
- Stakeholder Characteristics – Sophistication, domain knowledge, & ability to communicate between stakeholder and development team.
- Process Definition – Completeness and adequacy of the defined development process and the team's ability to follow it.
- Development Environment – Availability & quality of tools for project.
- Technology to be Built – Complexity & newness of encapsulated tech.
- Staff Size and Experience – Technical expertise to complete implementation, and to follow a process that avoids repeat mistakes.

Project Risk Drivers

- The U.S. Air Force maintains a software risk assessment system with these 4 considerations.
- Performance – The degree of uncertainty that a product will meet its requirements and prove suitable for its intended purpose.
- Cost – The degree of uncertainty that a project budget will be kept.
- Support – The degree of uncertainty that the resulting product will be maintainable and evolvable.
- Schedule – The degree of uncertainty that a project will be on time.
- "Cheap, fast, and good. Pick any two."

Technical Risks

- Poor understanding or specification of requirements can lead to commitments that can't be kept.
- Libraries, frameworks, and other technologies needed may turn out to have bugs, deficiencies, and incompatibilities that prevent implementation of requirements.
- Available tools may not be sufficient for design (e.g. Keyfactor can't apply automatic code generation for target version of .NET)
- It may turn out to not be feasible to meet non-functional requirements (implicit or explicit) for performance or scale, or to make a product simultaneously usable and secure.
- Complexity and technical debt can create QA and maintenance challenges that impact the product quality. Critical bugs that aren't found until after release are an especially costly risk.

Business Risks

- Scandal, controversy, bad press, and public failures can permanently damage a brand's reputation, leading to loss of business and revenue, and possibly dooming the organization. A good business mitigates this by attending carefully to personnel quality and public relations staff.
- Legal challenges to the technology (IP or application), intellectual property, business model, or business practices can similarly doom an organization. A good business mitigates this with quality legal counsel.
- Companies that fall victim to security breaches and experience technology theft, ransomware, customer data compromise, etc. can go out of business. A good company mitigates this with carefully managed information security practices and investments in staff and tools (e.g. Keyfactor products), plus employee education (e.g. anti-phishing training webinars), .
- Major market shifts (e.g. COVID) or new technologies and competition may eliminate the company's potential customers, users, or investors.
- Even events that don't destroy the company can result in changes to management, organization, priorities, and practices.

People-Related Risks

- Software is developed, maintained, operated, and consumed by humans.
- People are:
  - Fallible – Key assumptions about the business plan, product design and implementation, resource allocation, or market for users may be wrong. Decisions that are obviously, repeatedly, or egregiously wrong lead to firings.
  - Fragile – Developers and other stakeholders experience injuries, illnesses, personal emergencies, disabilities, bereavement, family care obligations, and death.
  - Diverse – People are motivated by different things, and pursue and transfer to other teams or companies. Most common is transfer for better pay but sometimes for location, ethical justification, or other personal considerations.
- Team departures leave knowledge and skill gaps, as well as project resource shortages. It's important to distribute knowledge so that the sudden departure of any *n* arbitrary members doesn't doom the project. This is called the team's "bus number". A bus number of 1 is very dangerous.

Risk Identification

- A systematic attempt to specify threats from various sources.
- Purview of project management.
- Goal is to avoid known/predictable risks entirely where it's possible, minimize their probability and impact where it's not, be fully prepared for them to manifest, and control them when they do.
- Starts with reviewing the project plan and considering what parts are like previous projects (and thus have the same risks) and what parts are new (and thus could present new risks).
- An experienced project manager may use a checklist or form.

Risk Assessment

- Not all risks identified are of equal significance. Need to prioritize based on a combination of *impact* – how bad is it (in $) if the outcome occurs – and *probability* – likelihood of condition occurring and leading to the outcome.
- Some risks are so improbable that extensive preparations and detailed assessments are not worth the effort.
- Some risks have such small impact that they can be handled easily without extensive preparations or detailed assessments.
- The impact of a risk is usually easier to project than the probability, especially the more uncertainty is involved.
- Risk *exposure* = impact x probability, giving an expected dollar figure.
- Pareto rule – 80% of overall project risk comes from 20% of risks IDed.

Risk Refinement and Monitoring

- Risks with higher exposure deserve more careful exploration of probability and impact, as well as degree of preparation. If you calculate exposure as $5, further refinement doesn't matter as much as a $500,000 risk.
- As uncertainty is removed and conditions are more carefully analyzed (e.g. contingency plan limits associated cost to less than the original assessment), the exposure calculation can be refined.
- As the project evolves, it's important to continually reassess risks to see if the probability is increasing or decreasing.
- There is a natural tendency to revise probability assessments after the fact, even if the figures were robust (e.g. if a probability is carefully and extensively calculated to 5%, but *does* occur, the tendency is to challenge that the probability should have been much higher, even though 5% events *do* happen). This is something to resist.

Risk Mitigation

- When risks are identified and assessed, the next question is what can be done to minimize the exposure by decreasing probability or, as a last resort, limiting the impact.
- Investments in planning, design, QA, project documentation, and a clear software engineering process are all targeted at mitigating project and technical risks involved.
- Unpredictable risks and many business risks can only really be handled by reactive management, and not much can be done by the development team to mitigate the risks.
- When risks do materialize, the best way to react is to follow a pre-defined contingency plan to limit the impact to the business.

Emergency Response

- Some risks inevitably materialize and resulting problems must be dealt with.
- Communication and engagement with everyone who's impacted and who can assist with the problem is important.
- Most problems can be dealt with while limiting losses to an acceptable level, frequently called "fire fighting".
- Some problems become true crises that leave a lasting impact on the project, team, and business.
- Contingency plans to follow can substantially improve outcome.
- Tracking how the problem arose, how it was handled, and what could have been done differently is important for assessing future risks.

Risk Management

- Risks can be tracked similar to product backlog work items and bugs with a Risk Identification Sheet or Risk Management Plan entry.
- Fields may include:
  - Name and description
  - Risk type
  - Probability assessment and reasoning
  - List of impacts
  - Triggering event
  - Owner or responsible party
  - Mitigations to probability and impact
  - Planned incident response (contingency plan)
  - Date and project phase when risk was identified and source or method for ID
  - Previous incidents or related risks
  - Outcome (if it occurs) and notes for future reference

SEI Risk Management Principles

- Maintain Global Perspective – View software risks from the context of the larger system of which it is part & the value it provides the user. Remember, your software is only one part of their business/job/life.
- Take a Forward-Looking View – Think about risks that may arise in the future and how action and contingency plans now can mitigate them.
- Encourage Open Communication – Don't discount anyone's concerns.
- Integrate – Incorporate risk management throughout the process.
- Emphasize a Continuous Process – As more information becomes known throughout the project, add and modify risks as outcomes of uncertainty (and new possibilities) become apparent based on new insights.
- Develop a Shared Product Vision – If two stakeholders have conflicting or misaligned goals for the product, at least one of them is at high risk.
- Encourage Teamwork – Pooled risk is easier to both catch and manage.

Summary

- There are many types of risk that occur in software engineering.
- Good preparation helps to avoid risks turning into incidents and to limit the impact of the incident.
- Not all risks can be anticipated and can only be reacted to.
- Most risks don't lead to absolute catastrophe but it is still worthwhile to try to prevent them and develop contingency plans for them.
- The risks that are mainly the responsibility of the development team are to the project budget and timeline, or implementation in the face of technical limitations.

References

- Fast, Good, or Cheap - Pick Three? Jamie Johnson. Aug. 2022. Business.com.
- Software Engineering Risk Management. Sonoo Jaiswal. 2021. Javatpoint.
- Risk Management in Project Management. Narudom Roongsiriwong. Aug. 2012. Slideshare.

- *For next lecture: Prepare questions for Mikyla on user interface, user experience engineering, product ownership, and related concepts.*