SOEN 6841: SOFTWARE PROJECT MANAGEMENT

Fall 2023

Prof: P. Kamthan

Learning Journal

Prepared by:
Name: Dhruvkumar Parmar
Student Id: 40235232

Table of Contents

1. Overview of Lecture

WEEK 1: Introduction to Software Project Management

Methodological Diversity: The documents encompass various project management methodologies, such as traditional, agile, and hybrid approaches. Understanding the strengths and weaknesses of each is crucial for effective project planning.

Critical Success Factors: Sudhakar's work highlights critical success factors for software projects, emphasizing the importance of a well-defined model to ensure project success. This underscores the significance of comprehensive planning and execution.

Soft Skills Integration: Several sources, including Sukhoo et al. and Wastian et al., stress the importance of soft skills in project management. Accommodating these skills is crucial for effective communication and team collaboration.

Continuous Learning: The references to various conferences, reports, and literature reviews, such as those by Marques and Ochoa, showcase the dynamic nature of software project management. Continuous learning and staying updated on industry trends are essential.

Agile Evolution: The evolution of agile methodologies, as evidenced by Layton and Ostermiller's editions over the years, highlights the dynamic nature of software development. Adaptability and a willingness to embrace change are crucial for success.

Risk Management: McGrath's exploration of failing by design and Boehm's perspective on managing software project risks shed light on the inevitability of challenges. Implementing effective risk management strategies is key to overcoming obstacles.

Leadership and Culture: Stanier's work emphasizes the role of leadership in software engineering management. Developing leadership skills and fostering a culture of collaboration are pivotal for project success.

WEEK 2: Introduction to Software Project Assessment

Understanding Software Project Failures

1. Comprehensive Literature Review:

    - Explored a wide range of literature on software project management.
    - Authors covered various aspects, including methodologies, critical success factors, and reasons for failures.

2. Key Themes:

    - Identified recurring themes across literature, emphasizing the complexity of software projects.
    - Recognized the importance of both technical and non-technical (managerial, human, social) factors in project outcomes.

3. Diverse Perspectives:

    - Literature spans diverse viewpoints, considering different contexts, experiences, and organizational cultures.
    - Varied insights from academia, industry practitioners, and researchers contribute to a holistic understanding.

4. Critical Success Factors:

    - Explored numerous critical success factors influencing software project outcomes.
    - Factors include realistic schedules, appropriate staffing, effective communication, and proactive risk management.

5. Patterns and Anti-Patterns:

    - Explored the relationship between patterns for software project management and critical success factors.
    - Considered anti-patterns as 'bad' solutions and their link to commonly-cited reasons for software project failure.

6. Project Management Practices:

- Emphasized the role of project management practices, such as effective communication tools and common repositories, in global software projects.

7. Reasons for Failure:

- Identified a range of reasons for software project failures, including unrealistic objectives, poor communication, and inadequate skills.
- Recognized that failures result from a combination of factors and are often non-technical in nature.

8. Anecdotal Nature:

- Acknowledged that reasons for failure are anecdotal and may not universally apply.
- Appreciated the multifaceted nature of project failures, challenging the notion of a single, definitive cause.

9. Software Project Maturity:

- Considered the relationship between software project maturity and project outcomes.
- Acknowledged that different approaches and maturity levels may influence the likelihood of project success.

10. Implications for Practice:

- Acknowledged the need for a nuanced understanding of project failures.
- Recognized the value of using reasons for failure as 'warnings' and input for checklists in project management.

11. Continuous Learning:

- Emphasized the iterative nature of learning from project outcomes.
- Encouraged ongoing reflection on literature to adapt practices and improve software project management.

12. Creative Commons License:

- Noted that the document is under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International license.

Conclusion:

- The exploration of diverse literature provides a rich foundation for understanding software project management, emphasizing the need for a multifaceted approach and continuous learning.

WEEK 3: Introduction to Stakeholder Management

Overview:

- Comprehensive exploration of software requirements and stakeholder management in the software development lifecycle.

- Diverse range of sources from academic papers to industry guides, providing a holistic view of the subject.

Key Concepts:

Stakeholder Management:

- Definition: Individuals or groups with a vested interest in the project's outcome.
- Importance: Critical for project success; stakeholders can positively/negatively influence the project.
- Methods: Various identification and analysis methods discussed.

Software Requirements:

- Definition: Descriptions of a system's functionality, performance, and constraints.
- Types: Functional, non-functional, and transitional requirements discussed.
- Significance: Foundation for project planning, design, and development.

Literature Review:

- Extensive list of references spanning various aspects of software requirements and stakeholder management.
- Authors like Kotonya, Sommerville, and Lauesen contribute to the foundational understanding of requirements engineering.

Highlighted Papers and Authors:

Kamthan, 2008a and 2008b:

- Discusses Pair Modeling and a Situational Methodology for Pragmatic Quality in Web Applications.
- Published in the Encyclopedia of Networked and Virtual Organizations.

Lim, Damian, Finkelstein, 2011:

- Explores StakeSource 2.0, a method using social networks for stakeholder analysis.
- Presented at the Thirty Third International Conference on Software Engineering.

Kuusinen, 2017:

- Addresses value creation and delivery in agile software development, overcoming stakeholder conflicts.
- Presented at the International Conference on Human-Computer Interaction.

Project Management and Metrics:

Noteworthy references to project management literature by authors like Kerzner (2023) and PMI's Guide to the PMBOK® (2021).

Practical Insights:

References to real-world challenges, such as social networks for large-scale requirements elicitation and stakeholder conflicts in agile projects.

Definitions and Observations:

- Definitions of stakeholders vary but commonly involve individuals or groups impacting or being impacted by the project.
- General definitions are project-oriented, while specific ones relate to software projects.

Overall Reflection:

- Rich resource for understanding the multifaceted nature of software requirements and stakeholder management.
- Insights from academia, industry experts, and project management perspectives provide a well-rounded view.

Next Steps:

- Explore specific methodologies mentioned, e.g., StakeSource 2.0 and Pair Modeling.
- Apply knowledge gained to practical scenarios, considering challenges highlighted in the literature.

Conclusion:

Valuable insights gained into the intricate relationship between software requirements, stakeholder management, and project s

WEEK 4: Introduction to Software Project Cost Estimation

Introduction to Software Project Estimation:

- Software project estimation is crucial for effective project management.
- Estimation involves predicting effort, duration, and cost associated with software development.

Software Estimation Techniques:

- COCOMO (COnstructive COst MOdel) is a widely used technique for software effort estimation.
- COCOMO I involves three modes: Organic, Semi-Detached, and Embedded.
- COCOMO II introduces additional features for more accurate estimations.

Basic COCOMO Equations:

- Effort estimation equation
- Duration estimation equation
- Cost estimation equation

COOMO Extensions:

- COCOMO I has been extended for cloud computing, introducing a "cloud computing platform" development mode.

Tools for COCOMO:

Calculations become intricate, and for larger projects, tools like COCOMO calculators and Cost Xpert Tool Suite are recommended.

Limits to Cost Estimation:

- Donald Rumsfeld's quote categorizes knowledge into known knowns, known unknowns, and unknown unknowns.
- Social, organizational, and technical limitations affect cost estimation.

Training and Subjectivity:

- Lack of formal training in estimation is a challenge for many software engineers.
- Cost estimation involves both science and art, with inherent subjectivity.

Basis and Environment:

- The relationship between size and effort is not always clear.
- Difficulty in obtaining accurate estimates of adjustment factors due to dependencies.

Conclusion and Approach:

- The estimator's role is to provide a reasonable projection of what developers will do.
- Cost estimation is not an exact science, and multiple approaches may be used.

WEEK 8: Introduction to Software Project Teams

Agile Practices and Teamwork Quality Evaluation:

- Two sources discuss the evaluation of Agile teamwork quality and propose approaches for measurement.
- Metrics and evaluation mechanisms are essential for continuous improvement in Agile environments.

Understanding Team Scaling:

- A source warns against the team scaling fallacy, underlining the declining efficiency of larger teams.
- Team scalability should be considered carefully, and strategies need to adapt with team size.

Remote Engineering Management:

- A source addresses the challenges of managing engineering teams in a remote-first world.
- Remote work requires tailored management strategies for sustained productivity.

Patterns for Effective Design Teams:

- A source introduces patterns for designing in teams, emphasizing collaborative design approaches.
- Recognizing and implementing effective design patterns contributes to team success.

Personality-Oriented Orientations in Gender Inequalities:

- A source presents an empirical theory of male software engineers' orientations in gender inequalities.
- Understanding diverse perspectives on gender issues contributes to creating inclusive teams.

Lessons from Software Engineering Pioneers:

- A source reflects on lessons from the pioneers of software engineering.
- Historical insights provide a foundation for understanding the evolution of the software engineering discipline.

Psychology in Project Management:

- A source discusses applied psychology for project managers, emphasizing the psychological aspects of project management.
- Project success is not only about technical aspects but also about managing the human side effectively.

Challenges in Software Development Waste:

- A source explores the concept of software development waste, addressing inefficiencies in the development process.
- Identifying and mitigating waste is crucial for optimizing software development processes.

WEEK 9: Introduction to Software Risk Management

Risk Management Standards:

- Various standards exist for risk management in software engineering, such as IEEE and ISO/IEC standards.
- Standards like ISO 31000:2009 provide principles and guidelines for risk management.

Literature Review:

- Extensive literature covers software risk management, including books by authors like C. L. Pritchard, H. Kerzner, and D. Kahneman.
- Topics include project recovery, decision-making, and the psychological aspects of risk.

Agile and Scrum:

- Agile methodologies, like Scrum, are discussed in works such as "Agile Software Development with Scrum."
- "The Practitioner's Handbook of Project Performance" explores Agile and Waterfall approaches.

Failure Mode and Effect Analysis (FMEA):

- FMEA is a method for identifying and preventing potential process and product failures.
- Factors like severity, occurrence, and detection are used to calculate the Risk Priority Number (RPN).

Risk Definitions:

- Definitions of risk vary, including a combination of the probability of occurrence and consequences.
- The concept of risk extends beyond uncontrollable factors, as some risks are within the control of project teams.

Case Studies and Real-world Applications:

- Real-world case studies are discussed in works like "Risk Management Framework for Information Systems and Organizations" by JTF.
- Authors like H. van Vliet discuss reflections on software engineering education.

Standards Evolution:

- Standards evolve over time, with recent standards like IEEE Standard 2675™-2021 covering DevOps.
- Ongoing updates, such as ISO/IEC/IEEE 16085:2021, reflect the dynamic nature of the field.

Multidisciplinary Approach:

- Risk management in software engineering often requires a multidisciplinary approach, considering factors like psychology (e.g., "Thinking, Fast and Slow" by D. Kahneman).

2. Important Topics Notes

WEEK 1: Introduction to Software Project Management

1. Introduction

Overview of Software Project Management.

Definition and Scope:

- Software Project Management involves planning, executing, and overseeing software development projects.

- Encompasses a range of activities from project initiation to closure.

Importance of Effective Software Project Management:

- Critical for project success and meeting stakeholder expectations.
- Addresses challenges unique to software development, such as changing requirements.

Challenges Faced by Software Project Managers:

- Uncertain requirements.
- Tight deadlines.
- Evolving technologies.
- Team collaboration and communication.

Significance of Successful Project Outcomes:

- Affects client satisfaction and project acceptance.

- Impacts organizational reputation.
- Influences future project opportunities.

Adopting Best Practices:

- Use of established methodologies (e.g., Agile, Waterfall) for project structure.
- Emphasis on communication, collaboration, and adaptability.

Overview of Subsequent Sections:

- The document explores various aspects of software project management.
- In-depth discussions on methodologies, challenges, and key success factors.

2. Foundations of Project Management

Fundamental principles and concepts in project management.

Project Management Defined:

- Project management involves the application of knowledge, skills, tools, and techniques to project activities.
- It aims to meet project requirements within defined constraints such as scope, time, cost, and quality.

Key Elements of Project Management:

Initiation:

- Defining project goals, scope, purpose, and feasibility.
- Identifying stakeholders and their expectations.

Planning:

- Developing a comprehensive project plan.
- Allocating resources, defining tasks, and establishing timelines.

Execution:

- Implementing the project plan.
- Coordinating people and resources to achieve project objectives.

Monitoring and Controlling:

- Tracking project performance against the plan.
- Taking corrective actions as needed.

Closing:

- Completing all project activities.
- Formalizing project closure and obtaining client acceptance.

Project Life Cycle:

- Projects typically go through phases: initiation, planning, execution, monitoring/control, and closure.
- Phases may overlap, and the life cycle is adapted based on the project's nature.

Role of the Project Manager:

- The project manager is a crucial figure responsible for overall project success.
- Involves leadership, communication, risk management, and conflict resolution.

Importance of Project Management:

- Enhances efficiency and effectiveness.
- Reduces risks and uncertainties.
- Facilitates communication and collaboration.
- Improves overall project outcomes.

Foundational Project Management Methodologies:

Waterfall:

- Sequential, linear approach.
- Progresses through defined phases.

Agile:

- Iterative and flexible.
- Emphasizes adaptability to changing requirements.

Challenges in Project Management:

- Balancing constraints (scope, time, cost).
- Managing uncertainties.
- Ensuring stakeholder satisfaction.

Conclusion:

- A strong foundation in project management principles is essential for successful software project delivery.
- Effective project management contributes to organizational success and client satisfaction.

3. PITFALLS OF (SOFTWARE) PROJECT MANAGEMENT

- Be Completely and Unrepentantly Obsessed with the Customer
- Provide Shared, Measurable, Challenging, and Achievable Goals as Clear as Sunlight
- Engage in Effective, Vociferous, Unrelenting Communication with All Stakeholders
- Ensure that Roles and Responsibilities are Unmistakably Understood and Agreed Upon by All
- Create Viable Plans and Schedules that Enjoy the Team's Hearty Commitment
- Mitigate Big, Hairy, Abominable Risks, and Implement Innovative Accelerators
- Prioritize Ruthlessly, Choosing Between Heart, Lungs, and Kidneys if Necessary
- Anticipate and Accommodate Necessary and Inevitable Change
- Challenge Assumptions and Beliefs, Especially Insidious Self-Imposed Limitations
- Manage the Expectations of All Stakeholders: Under-Promise and Over-Deliver
- Learn from Experience. Make New and More Exciting Mistakes Each Time!
- Attitude of Gratitude: Celebrate Project Success... And Some Failures6 , Too!

4. A CLASSIFICATION OF SOFTWARE PROJECTS

The nature of a software project determines how that project should be pursued

The two dimensions are:

1. Problem Domain Uncertainty (Uncertainty in “What”): This is about uncertainty in the knowledge of the problem domain, specifically uncertainty in requirements.

2. Solution Domain Uncertainty (Uncertainty in “How”): This is about uncertainty in the knowledge of the solution domain, specifically uncertainty in technique or technology.

5. TYPES OF COMPLEXITY IN SOFTWARE PROJECTS

1. Structural Complexity: This is about the size, scope, and interdependence of tasks and personnel.

2. Sociopolitical Complexity: This is about the importance of the project to the organization, people, power, and politics within and outside of the organization, and conflicting agendas within the stakeholder community.

3. Emergent Complexity: This is about the uncertainty of the outcome, lack of experience with the problem (application) domain or technology, lack of information, or some combination of these.

6. Agile Project Management

Introduction:

- Agile Project Management is an iterative and flexible approach to managing software development projects.
- It prioritizes collaboration, customer feedback, and small, rapid releases over strict planning and extensive documentation.

Core Principles:

Customer Collaboration:
- Continuous engagement with customers to understand and meet their evolving needs.

Adaptability:
- Embracing changes in requirements even late in the development process.

Individuals and Interactions:
- Valuing team members and their interactions over rigid processes and tools.

Working Solutions:

- Prioritizing delivering a working product over exhaustive documentation.

Key Features:

Iterative Development:

- Work in small, functional iterations called sprints.
- Regularly reassess and adjust project goals.

Scrum Framework:

- Employs Scrum roles (Product Owner, Scrum Master, Team) and events (Sprint Planning, Daily Standup, Sprint Review, Sprint Retrospective).

User Stories:

- Requirements expressed from an end-user perspective.
- Encourages collaboration between developers and users.

Advantages:

Flexibility:

- Easily adapts to changing requirements.
- Welcomes customer feedback for continuous improvement.

Faster Delivery:

- Quick and frequent releases.
- Faster response to market changes.

Enhanced Collaboration:

- Close-knit, self-organizing teams.
- Regular communication and feedback.

Challenges:

- Documentation:
- May lack comprehensive documentation.
- Balance needed between agility and necessary documentation.
- Customer Involvement:
- Requires consistent and active customer engagement.
- Not suitable for projects with distant or uninvolved clients.

Conclusion:

- Agile Project Management is a dynamic, customer-centric approach suitable for projects with evolving requirements.
- Its success relies on effective communication, collaboration, and the ability to adapt to change swiftly.

7. MODEL 1

- The psychological profiling of software project managers has revealed the following desirable skills.

- Planning
- Scheduling
- Staffing
- Motivating
- Controlling

8. MODEL 2

- Technical Skills
- Strategic and Business Management Skills
- Leadership Skills

9. Knowledge Management in Software Projects

- Techniques for transferring and managing knowledge within software development teams.

Introduction:

- Knowledge Management (KM) in software projects involves capturing, organizing, and leveraging the collective knowledge and experiences of a project team.

Key Components:

Knowledge Capture:

- Systematic collection of explicit and tacit knowledge.
- Documentation of best practices, lessons learned, and project artifacts.

Knowledge Organization:

- Structuring information for easy retrieval.
- Utilizing knowledge repositories, databases, or wikis.

Knowledge Sharing:

- Encouraging open communication and collaboration.
- Regular team meetings, workshops, and collaborative platforms.

Knowledge Transfer:

- Ensuring knowledge is passed on when team members change.
- Mentorship programs, training sessions, and comprehensive documentation.

Importance in Software Projects:

Reduced Redundancy:

- Avoiding the repetition of mistakes by learning from past experiences.
- Efficient problem-solving through shared knowledge.

Enhanced Productivity:

- Accelerating decision-making with readily available information.
- Minimizing delays caused by repeated problem-solving.

Continuous Improvement:

- Facilitating a culture of learning and adaptation.
- Iterative enhancement based on insights from previous projects.

Risk Mitigation:

- Identifying and mitigating risks through shared insights.
- Anticipating challenges based on historical data.

Challenges:

Knowledge Hoarding:

- Some team members may resist sharing knowledge for various reasons.
- Addressed through a culture of collaboration and recognition.

Technological Barriers:

- Ineffective or outdated knowledge management tools.
- Regularly updating and upgrading tools to match project needs.

Cultural Shift:

- Resistance to adopting a knowledge-sharing culture.
- Leadership plays a crucial role in fostering an environment of openness.

Best Practices:

Establish a Knowledge Sharing Culture:

- Encourage a mindset that values and promotes knowledge sharing.
- Recognize and reward contributions to the knowledge base.

Use of Technology:

- Implement efficient knowledge management tools.
- Ensure accessibility and user-friendly interfaces.

Continuous Learning:

- Regularly update documentation and knowledge repositories.
- Conduct periodic training sessions and workshops.

Conclusion:

- Knowledge Management is integral to software projects for efficiency, risk reduction, and continuous improvement.
- Overcoming challenges requires a combination of technological solutions, cultural shifts, and leadership support.

WEEK 2: Introduction to Software Project Assessment

1. SOFTWARE PROJECT ASSESSMENT: DELIVERY VIEWPOINT

Definition [Successful Software Project]. A software project is considered successful if a product is delivered and all its success criteria are within an acceptable range.

Definition [Challenged Software Project]. A software project is considered challenged if a product is delivered; however, some of its success criteria are not within an acceptable range.

Definition [Failed Software Project]. A software project is considered failed if a product is not delivered.

2. LAWS OF FEASIBILITY

First Law of Project Feasibility: Law of Positive and Negative Forces If the sum of the negative forces on the project (which play against the project, such as risks) is larger than the sum of the positive forces on the project (which play in favor of the project, such as potential for new markets), then the project is not feasible.

Second Law of Project Feasibility: Dependencies The stronger the dependencies between the project's tasks, the more vulnerable the project is. Furthermore, high task interdependence generates high potential vulnerability.

Third Law of Project Feasibility: The Law of Points of Vulnerability The higher the total vulnerability is and the weaker the remedial actions are, the less the project is feasible.

Fourth Law of Project Feasibility: Law on the Forces of Production Let F denote forces of production that are under control, and UF denote forces of production that are not under control. The more UF is greater than F, the higher the probability of failure.

Fifth Law of Project Feasibility: Law on Conflicts The more conflicts among stakeholders are intense, frequent, and cover critical issues, the less likely the project is feasible.

Sixth Law of Project Feasibility: Law of Complexity The more complex a project is, the riskier it is, the more POVs it contains, and the higher probability of failure. The data on software project failures can be used as an input to feasibility studies for a new software project.

3. COST OF SOFTWARE PROJECT FAILURES

Loss of Capital. There can be partial or total loss of the capital invested in the software project. In case of partial loss, it is possible to recover some of the capital, for example, by means of reallocation or reuse of infrastructure. In any case, the Return On Investment (ROI) [Denne, Cleland-Huang, 2003] is unacceptably low.

Loss of Market. There can be loss of market, say, in form of share value or potential clients (or customers) if marketing of the software product preceded its delivery (that did not occur).

Loss of Reputation. The organization, in general, and/or the team, in particular, can incur a loss of reputation if the outcome of being not successful of the software project is known externally, such as by other organizations and/or public-at-large.

Loss of Goodwill. In case a software project is not successful, the relationship between an organization and its clients (or customers or consumers), or between an organization and its contractors or partners, can be damaged, perhaps irreparably.
Loss of Professional Career. In case a software project is not successful, there can be consequences for employees of the organization, involved in that project directly, or related to the project indirectly. The repercussions take one or more of the following forms: suspension or dismissal from job, premature termination or non-renewal of contract, and freeze on hiring.

4. CHALLENGES IN FORMULATING SUCCESS CRITERIA

- Variability of Perception of Success.
- Multi-Dimensionality of Success
- Quality of the Success Criteria
- Lack of Sufficiency of Criteria.

5. THE RELATIONSHIP BETWEEN SOFTWARE PROJECT ASSESSMENT AND SOFTWARE DEVELOPMENT PROCESS

Definition [Ad-Hoc Software Project]. In an ad-hoc software project, the team does not follow a defined process.

Definition [Iterative Software Project]. In an iterative software project, the team follows a process that is organized into periods that are often referred to as iterations or time boxes. On any given day, the project team members may be eliciting requirements, designing, coding, testing, and so on

Definition [Agile Software Project]. In an agile software project, the team follows an agile software development process.

6. REASONS FOR SOFTWARE PROJECT FAILURE

The following are considered common reasons for software project failure:

- Communication
- Estimation
- Knowledge
- Involvement
- Management
- Education
- Risks
- Tools

7. A MODEL OF LEVELS OF SUCCESS

- Level 1: Project Management Success
- Level 2: Project Success
- Level 3: Business Success
- Level 4: Future Potential

Level 1: Project Management Success. This level is concerned with the effectiveness of the project management. In doing so, it seeks the satisfaction of project constraints, such as those set by the equilibrium triangle. The measures of success for this level include the use of resources, earned value management (EVM), change requests, and so on.

Level 2: Project Success. This level is concerned with the effectiveness of the project. It looks at what is actually delivered to the stakeholders upon conclusion of the project. The measures of success for this level include the satisfaction of requirements, utility and usability of the product, complaints, and so on.

Level 3: Business Success. This level is concerned with the effectiveness of the organization. It looks at the creation and delivery of internal value. The measures of success for this level include the return on investment (ROI), net present value (NPV), internal rate of return (IRR), revenue measures, sales, reputation, and so on.

Level 4: Future Potential. This level is concerned with the business horizon of the organization. It looks at the future opportunities and areas of gains by opening new avenues, acquiring new capabilities and skills, and venturing new markets. The measures of success for this level include the competitive advantage, recognition in new market segment, derived products, and so on.

WEEK 3: Introduction to Stakeholder Management

1. DEFINITIONS

Definition [Stake] [Bourne, 2009, Page 30]. [It] may be one or more of the following: an interest, rights (legal or moral), ownership, or contribution.

The following definition is specific to a software project:

Definition [Stake] [McManus, 2004a, Page 23]. [It] can be described as an interest or share in a project undertaking to achieve business, technical, or social goals.

Definition [Stakeholder] [Wiegers, 2006, Chapter 2]. [An] individual or group who is actively involved in the project, who is affected by the project, or who can influence its outcome

Definition [Stakeholder] [ISO/IEC/IEEE, 2011]. [An] individual, team, organization, or classes thereof, having an interest in a system.

Definition [Stakeholder] [ISO/IEC/IEEE, 2018]. [An] individual or organization having a right, share, claim or interest in a system or in its possession of characteristics that meet their needs and expectations.

Definition [Organization] [ISO/IEC/IEEE, 2010]. [A] group of people and facilities with an arrangement of responsibilities, authorities, and relationships

Definition [Team] [Tabaka, 2006]. A small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.

2. STAKEHOLDER ANALYSIS

There are several reasons for stakeholder analysis (that is, paying attention to the study of stakeholders):

- Understanding Ecosystem
- 'Peopleware' and the 'People Factor'
- The Human and Social Dimensions of Software Engineering
- The Human and Social Dimensions of Requirements Engineering
- Understanding the Belief System of People involved in Software Project
- Understanding People involved in Collaboration
- Understanding Purpose of Software Project
- Understanding Scope of Software Project
- Understanding Value of Software Project
- Economics of Software Project
- Outcome of Software Project
- Organizational Change

3. EXAMPLES OF STAKEHOLDERS:

The following are some examples of stakeholders, in no particular order of significance and with no claim of exhaustiveness:

- Analyst
- Buyer
- Coach
- Consultant (Information Technology Consultant)
- Crowdsourced Persona
- Crowdsourced Tester
- Decision Maker (Owner, Executive (President, Director), Manager, Other)
- Domain Expert (Subject Matter Expert)
- Ethical Hacker
- Financial Beneficiary (Seller, Stockholder, Other)
- 17
- Human Factors Expert (Usability Expert, Other)
- Legislator (Lawyer, Parliamentarian, Other)
- Negative Stakeholder (Bad Actors, Competitor, Fraudster, Hacker, Prankster,
- Vandal, Other)

- Non-Governmental Organization
- Organization (Affiliate, Partner, Other)
- Product Owner
- Publisher (Book Author, Software Magazine Editor, Other)
- Regulator (Auditor, Representative of Standards Organization, Government Official,
- Other)
- Software Engineer (Business Analyst, Designer, Programmer, Internal Evaluator
- (Inspector, Tester, Other), Maintainer, Technical Writer, Other)
- Special Interest Group (SIG)
- Sponsor
- Team
- Trainer
- User
- Vendor (Seller)

4. CHALLENGES IN THE MANAGEMENT OF STAKEHOLDERS

There are a number of technical and social challenges in the management of stakeholders, including the following:

- Tractability
- Variability

TRACTABILITY

The main technical challenge in the management of stakeholders is tractability. There are a number of factors that can adversely impact tractability of stakeholders.
For example, the stakeholder information recorded at one time could get obsolete at another time, say, before the end of the project. In a distributed software development environment, such as global software development [Hanisch, Corbitt, 2004; Berenbach, 2006; Damian, 2007], it can be difficult to track stakeholders. This is especially the case if the work related to the project is outsourced at multiple levels, and there is lack of transparency due to commercial, legal, or political reasons.

VARIABILITY

The main social challenge in the management of stakeholders is variability [Missonier,

Loufrani-Fedida, 2014]. It is known that people's relationships with the project (such as interests, loyalties, and commitments) and relationships with each other can evolve and change over time. This means that somebody who is considered a stakeholder today may not remain so tomorrow. There are consequences of variability. For example, the degree of commitment of a stakeholder can have an impact on the quality of requirements elicitation [Burnay, Jureta, Faulkner, 2015].

5. IDENTIFICATION OF STAKEHOLDERS

The commonly-used approaches for the identification of stakeholders, discussed in detail
in the following, are:

- Organizational Chart
- Similar Projects
- Communities of Practice
- Context Analysis
- 35
- Social Web Analytics
- Question-Directed Brainstorming and Discussion

6. CLASSIFICATION OF STAKEHOLDERS BY VIEWS

These views are described below.

Contribution View. For example, stakeholders could be classified as technical and non-technical (in the sense of their experience and education that they bring to contribute to the software project).

Product View. For example, stakeholders could be classified as producers and consumers (in the sense of input to and output from the software project).

Contactual View. For example, stakeholders could be classified as direct and indirect stakeholders (in the sense of interaction with the software system) [Kotonya, Sommerville, 1998; Aschauer, Hruschka, Lauenroth, Meuten, Rogers, 2018, Section 2.2; Müller, Hussain, Grundy, 2022].

In [Alkhusaili, Qureshi, 2021, a case study of the Obesity Control Health Information System (OCHIS) is presented, and Managers, Therapists, and Staff have been identified as direct stakeholders, and Patients, Guardians (Parents), Prospective Patients have been identified as indirect stakeholders.

Participation View. For example, stakeholders could be classified as internal and external stakeholders (in the sense of involvement in the software project) [Smith, 2000].

Impact View. For example, stakeholders could be classified as positive and negative stakeholders (in the sense of the nature of impact on the software project) [Glinz, Wieringa, 2007].

Outcome View. For example, stakeholders could be classified as primary and secondary stakeholders (in the sense of being affected by the success, or the lack thereof, of the software project) [McManus, 2004a, Section 2.2; Pan, 2005; Freeman, Harrison, Wicks, Parmar, Colle, 2010, Page 24; Ambler, Lines, 2012, Chapter 4]. Figure 18 shows primary and secondary roles in the Disciplined Agile Delivery (DAD) [Ambler, Lines, 2012]. (The definition of stakeholder adopted by DAD is different from that used in this document.)

Vocational View. For example, stakeholders could be classified as P – Political, E – Economic, S – Sociocultural, T – Technological, L – Legal, and E – Environmental (abbreviated as PESTLE, in the sense of the type of vocation with respect to the software project)

7. CLASSIFICATION OF STAKEHOLDERS BY ROLES

1. Responsible: These stakeholders conduct the actual project work. These stakeholders are those who do the work to complete a task or produce a deliverable. There is usually at least one R role. They may carry different kinds of titles, including requirements engineer, software architect, programmer, or software tester. The communication with them is unidirectional.

2. Accountable: These stakeholders report to or are reported by those at different levels of management. These stakeholders are those who approve completion of the work towards a task or deliverable. There is usually exactly one role for each task or deliverable. They may carry different kinds of titles, including business liaison, program manager, or project manager. The communication with them is unidirectional.

3. Supportive: These stakeholders are those who provide resources (such as information) for work towards a task or deliverable. There is usually at least one S role. The communication with them is bidirectional.

4. Consulted: These stakeholders are those who are consulted (such as problem domain experts) to complete a task or produce a deliverable. These stakeholders are consulted on a regular basis, but do not carry out the actual work. The communication with them is bidirectional. There is usually at least one C role. They may carry different kinds of titles, including subject matter experts. The communication with them is bidirectional.

5. Informed: These stakeholders are those who are kept up-to-date on the progress of the work towards a task or deliverable. These stakeholders do not have a say in the project. There is usually at least one I role. These stakeholders may carry different kinds of titles, including users and vendors. The communication with them is unidirectional.

8. NEGATIVE STAKEHOLDERS

- Negative User. A user engaging in a negative use is called a negative user [Courage, Baxter, 2005]. A negative user is a kind of negative stakeholder.

- Human Misuser. The negative uses of a software system can be modeled by a negative use case or misuse case [Sindre, Opdahl, 2005].

- A human misuser (that is, a human actor of a misuse case) is a kind of negative stakeholder. A negative persona is a kind of negative user, and, therefore, a kind of negative stakeholder.

- Black Hat Hacker. There are different kinds of hackers: black hat hacker, grey hat hacker, and white hat hacker. (In other words, hackers vary on a spectrum [Peterson, 2016].)

- A black hat hacker is a hacker who has the skills and intent to “break into a system or network” and in doing so “violates computer security for little reason beyond maliciousness or for personal gain” [Wikipedia]. In other words, a black hat hacker is a kind of hacker as well as a negative user. A black hat hacker is therefore a kind of negative stakeholder. Figure 20 presents an example.

- Negative Extreme Character. An extreme character [Djajadiningrat, Gaver, Frens,

2000] is a non-stereotypical user. An extreme character could be positive or Negative.

9. THE 'ONION' STAKEHOLDER MODEL

The following is an adaptation of [Alexander, 2005]:

1. Construct Circles. Let there be a collection C = {C1, C2, C3, …} of concentric circles and the software project P in progress be placed at the center of C1.

The distance of a circle from P is inversely proportional to and is a reflection of its degree of influence on P.

For example, the further a circle is from P, the lesser the influence it has on P.

2. Identify Stakeholders. The collection S = {S1, S2, S3, …} of stakeholder roles with respect to P are identified. Then, a role is placed in the intervening space between circles, depending on its degree of influence on P.

In an 'onion' stakeholder model, it is possible to have multiple stakeholder roles with the same influence on P.

10. STATES OF A STAKEHOLDER

1. Identified: A stakeholder is identified (as being a legitimate stakeholder) and given an identification number.

2. Classified: A stakeholder is classified in some manner, which can aid understanding.

3. Represented: A stakeholder is represented in some manner, which can aid effective and efficient searching and retrieving information about him or her.

4. Involved: A stakeholder is involved actively in the project and fulfilling his or her responsibilities.

5. In Agreement: A stakeholder is in agreement with the other stakeholders.

6. Satisfied With Deployment: A stakeholder is satisfied with the system deployment since it meets the minimum expectations set.

7. Satisfied In Use: A stakeholder is satisfied with the system use since it meets the expectations set.

WEEK 4: Introduction to Software Project Cost Estimation

1. DEFINITION & MOTIVATION FOR COST ESTIMATION

Definition [Estimate] [DeMarco, 1982; Fenton, Pfleeger, 1997, Page 428; Laird, 2006]. A prediction that is equally likely to be above or below the actual result.

There are number of reasons for conducting cost estimation, including the following:

- Feasibility Study
- Budgeting
- Trade-Offs
- Risk Analysis
- Creating Pedigree
- Collaborative Learning and Improving

2. CHALLENGES IN SOFTWARE PROJECT COST ESTIMATION

Complexity. The software projects over the years have become complex undertakings, characterized by (1) uncertainty in the software ecosystem, (2) interaction between an intimidating number of elements in the ecosystem, and (3) changes in the software ecosystem

Human-Dependency. The software projects include a major human component (more so than other ‘conventional’ engineering disciplines). This essentially means that many software engineering processes cannot be formalized or automated completely. (There are advantages of artificial intelligence (AI), but also essential limitations of AI.)

Circularity. This is a manifestation of Catch-22. The estimates are most useful at a very early stage in software development. However, usually, little is known at a very early stage (for example, software requirements are yet to be elicited and decided)

Psychology (Human-Dependency Redux). The software projects are subject to various forms of cognitive biases [Holm, 2015; Dinwiddie, 2019] due to social and political pressures

- Wishful Thinking
- Planning Fallacy
- Anchoring
- Cognitive Dissonance
- Numeracy Bias

3. STAKEHOLDERS FOR SOFTWARE PROJECT COST ESTIMATION

The stakeholders specific to management aspect of cost estimation are the following

Domain Expert. This role provides input information for building an estimation model when measurement data are missing or insufficient. The domain experts (or, equivalently, subject matter experts) do not have to be knowledgeable in estimation.

Estimation Process Owner. This role is responsible for introducing and maintaining estimation processes, methods, models, and data within an organization.

Estimator. This role uses estimation methods and estimation models existing within an organization for estimating particular software development projects.

Product Owner. This role represents may not be directly involved in the estimation process, but has the power to affect the software project, including the performance of estimation. For example, a product owner (also called a Project Sponsor) may pressure estimators with respect to software project resources and, by doing so, bias estimates.

4. UNCERTAINTY

Probabilistic Uncertainty

- It addresses what is unpredictable in information. For example, change in market demand.
- It is caused by limited capability of humans to foresee a phenomenon
- It can be reduced in a number of ways, including ‘designing for change’

Possibilistic Uncertainty

- It addresses what is incomplete or unclear in information. For example, software requirements.
- It is caused by limited knowledge of or capability of humans to observe a phenomenon.
- It can be reduced in a number of ways, including ‘prototyping’.

Granulation Uncertainty

- It addresses what is inexact (and only approximate) in information. For example, decimal representation of an irrational number.
- It is caused limited capability of humans to describe a phenomenon.
- It can be reduced in a number of ways, including ‘selecting a proper level of abstraction and focusing only on relevant issues’.

5. ONION MODEL FOR PLANNING AND AGILE SOFTWARE PROJECTS

In agile software projects, the teams are usually concerned only with the three innermost layers of the planning onion, namely Release, Iteration, and Daily, each with different goals.

Release Planning: This is conducted at the start of a software project, but is updated throughout the project (usually at the start of each iteration) so that it always reflects the current expectations about what will be included in the release. The goal is to determine an appropriate answer to the questions of scope, schedule, and resources for a software project.

Iteration Planning: This is conducted at the start of each iteration. It is based on the work accomplished in the previous iteration. The goal is to identify high-priority work the software project team should address in the new iteration.

Daily Planning: This is conducted at the start of each day. It usually takes the form of some of daily stand-up meeting. The goal is to coordinate work and synchronize daily efforts, and, in doing so, look no further away than the next day.

6. CLASSIFICATION SCHEME

There are two types of models that have been used to estimate effort, depending on the mathematical approach deployed:
1. Cost Models
2. Constraint Models

COST MODELS

The cost models provide direct estimates of effort (or duration). They are usually based on empirical data that reflects factors that contribute to overall cost.

There is a single primary factor (usually, a measure of product size) and multiple secondary factors (usually, cost drivers).

The cost drivers that are characteristics of the software project, process, products, and resources that are expected to influence effort (or duration) in some manner. These cost drivers are used to adjust the preliminary estimate provided by the primary factor

CONSTRAINT MODELS

The constraint models illustrate the relationship over time between two or more parameters of effort, duration, or staffing level. The Putnam-Norden-Rayleigh Curve is used to illustrate the relationship over time.

There are three types of models that have been used to estimate effort, depending on the degree of sophistication [Jensen, Putnam, Roetzheim, 2006]:

1. First-Order Models
2. Second-Order Models
3. Third-Order Models

FIRST-ORDER MODELS The first-order model is the most rudimentary model class. It defines the production capability of the development organization in terms of arbitrary

production units multiplied by the software product effective size to obtain the development effort or cost

SECOND-ORDER MODELS The second-order model compensates for the productivity decrease in larger software projects by incorporating an entropy factor to account for the productivity change. The entropy effect demonstrates the impact of a large number of communications paths that are present in large development teams.

THIRD-ORDER MODELS The third-order model compensates for the second-order model's narrow applicability by incorporating a set of environment factors to adjust the productivity factor to fit a larger range of problems.

7. REGRESSION-BASED COST ESTIMATION

1. Data Analysis. The data from past software projects was collected, and relationships among the attributes measured are examined.

Estimation Equation. It was hypothesized that some of the factors could be related by an equation.

Adjustment. Upon the definition of the basic equation, the estimate was adjusted by other, secondary cost factors

8. COCOMO

COCOMO I is based on software projects following a Waterfall Model in a mainframe computing environment, the prevalent software development process and computing environment in the 1970s.

COCOMO II is an evolution of COCOMO I to address software development practices, such as desktop computing, source code reusability, and the use of commercial-off-the-shelf (COTS) software components, in the 1990s and 2000s. It also has an updated project database, which continues to evolve

COCOMO I applies to three types of development modes (software projects): organic, semi-detached, and embedded.

- Organic
- Semi-Detached
- Embedded

Basic COCOMO. The Basic COCOMO is intended for quick, early, and rough order of magnitude estimates of software costs. It can be used at the beginning of software development when little knowledge is available. The Basic COCOMO does not take into account product, hardware, personnel, and software project attributes (that is, the 'cost drivers'). Therefore, its accuracy is limited

Intermediate COCOMO. The Intermediate COCOMO takes cost drivers into account.

Detailed COCOMO. The Detailed COCOMO, in addition to cost drivers, takes into account the influence of individual software project phases.

WEEK 8: Introduction to Software Project Teams

1. DEFINITION

> Definition [Team] [Tabaka, 2006]. A small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.
>
> Definition [Team] [So, 2010]. A collective of two or more individuals who (1) socially interact, (2) possess one or more common goals, (3) are brought together to perform organizationally relevant tasks, (4) exhibit interdependencies with respect to workflow, goals, and outcomes, (5) have different roles and responsibilities, and (6) are together embedded in an encompassing organizational system, with boundaries and linkages to the broader system context and task environment.

2. CROSS-FUNCTIONAL TEAMS IN AGILE METHODOLOGIES AND DEVOPS

The following characteristics of cross-functional teams are given:

Autonomy.

- The team self-organizes, self-directs, and self-manages on a day-to-day basis working towards a common goal with minimum external interference. For

example, the client/customer/organization defines the goal, and the team decides the delivery.

Requisite Variety.

- The team uses a variety of approaches for problem solving. For example, these could include Product Backlog, Kanban Board, Architectural Spike, and so on.

Learning to Learn.

- The team reflects regularly. For example, these could include Stand-Up Meeting, Retrospective Workshop, Pair Programming, and so on.

Cross-Fertilization.

- The team is heterogeneous, as shown in Figure 3. It has a diverse group of people who bring different specializations, skills, views (towards the problem and/or the solution), and learn from each other. For example, these could include subject matter experts, marketers, developers, testers, operators, and so on.

Self-Evaluation.

- The team evaluates itself regularly. For example, this is done for improving productivity or software product quality.

3. ‘AUTONOMY’ IN TEAMS

The following characteristics of autonomous teams are given:

Independent. The team is self-directed, with clear goals and boundaries they can operate in.

Empowered. The team expects that its decisions are not questioned constantly because it is comfortable and confident of its decision making.

Accountable. The team feels responsible for the outcomes it has agreed to.

Collaborative. There is a high level of trust within the team, and team members feel comfortable speaking their minds and working together.

Interdisciplinary. The team encourages diverse perspectives and experiences as it believes they contribute to better decisions and better solutions.

Transparent. The team allows broad distribution and open sharing of information (as opposed to narrow distribution of or withholding information).

4. TUCKMAN MODEL

1. Forming: The team members meet for the first time, getting to know each other.

2. Storming: The relationships among team members are created and tasks are assigned. The team members compete with each other for status and for acceptance of their ideas.

3. Norming: The team creates the rules of internal behavior that allow achieving the goals and developing mutual trust and reliance. The team members take responsibility, respect each other's opinions, and value their differences.

4. Performing: The roles and rules are established. The team works as a wellorganized, high-performing unit.

5. Adjourning: The team is at the end of the project. The relationships among team members are such that it can accelerate the development of the team (through the initial few stages) in case of future cooperation.

5. 'PERSONALITIES' OF TEAMS

- Neuroticism
- Extraversion
- Agreeableness
- Openness
- Conscientiousness

Definition [Team Climate] [Soomro, Salleh, Mendes, Grundy, Burch, Nordin, 2016]. A combination of its team members' interactions in order to share their perceptions of the team's work procedures and practices

- Participative Safety
- Support for Innovation
- Team Vision

- Task Orientation

6. CONFLICT RESOLUTION MODEL

This model measures behavior in conflict situations based on two dimensions:

1. Vertical Dimension

2. Horizontal Dimension
   - Avoiding
   - Competing
   - Accommodating
   - Collaborating
   - Compromising

7. BLOOM'S TAXONOMY

The following provides a summary of each level:

- Remembering
- Understanding
- Applying
- Analyzing
- Evaluating
- Creating

8. THE FOUR STAGES OF LEARNING MODEL

- Unconscious Incompetence
- Conscious Incompetence
- Conscious Competence
- Unconscious Competence

WEEK 9: Introduction to Software Risk Management

1. DEFINITION

Definition [Event] [ISO/IEC, 2006; ISO, 2009a]. The occurrence of a particular set of circumstances.

Definition [Probability] [ISO/IEC, 2006]. The extent to which an event is likely to occur.

Definition [Probability] [ISO, 2009a]. [The] measure of the chance of occurrence expressed as a number between 0 and 1, where 0 is impossibility and 1 is absolute Certainty.

Definition [Likelihood] [ISO, 2009a]. [The] chance of something happening. NOTE 1 In risk management terminology, the word “likelihood” is used to refer to the chance of something happening, whether defined, measured or determined objectively or subjectively, qualitatively or quantitatively, and described using general terms or mathematically (such as a probability).

Definition [Consequence] [ISO/IEC, 2006]. An outcome of an event.

Definition [Risk] [ISO/IEC, 2006]. The combination of the probability of an event and its consequence.

Definition [Risk Item] [Kontio, 2001]. A risk that has not been analyzed (and is therefore a “raw risk”).

Definition [Risk Source] [ISO, 2009a]. [The] element which alone or in combination has the intrinsic potential to give rise to risk.

Definition [Party] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. An organization, company, person, group, or other body who values an asset and on whose behalf a risk assessment is conducted.

Definition [Asset] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. [It is] anything of value to a party (person, group, or organization).

Definition [Risk] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. The likelihood of an incident and its consequence for an asset.

Definition [Likelihood] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. The chance of something to occur.

Definition [Incident] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. An event that harms or reduces the value of an asset.

Definition [Consequence] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. The impact of an incident on an asset in terms of harm or reduced asset value.

Definition [Threat] [Alberts, Dorofee, 2010]. A circumstance with the potential to produce loss.

Definition [Consequence] [Alberts, Dorofee, 2010]. The loss that will occur when a threat is realized.

Definition [Impact] [Alberts, Dorofee, 2010]. A measure of the loss that will occur if the threat is realized.

Definition [Continuous Risk Management] [IEEE, 2021]. [A] continuous process, which may be automated, that identifies, applies, and monitors controls to treat risks for a planned activity, project, or program, to achieve a desired outcome

Definition [Risk Management Process] [ISO/IEC, 2006]. A continuous process for systematically identifying, analyzing, treating, and monitoring risk throughout the life cycle of a product or service.

2. RISKS UNIQUE TO SOFTWARE

The emergence of software risk management as a discipline in its own right is motivated by the following risks unique to software:

Connecting. There is a risk in connecting a software system to a network. The negatives include the potential for negative uses, such as a cyberattack (for example, (distributed) denial-of-service).

Distributing. There is a risk in distributing parts of a software system over a network. The negatives include unacceptable delays in transmission of data, especially during high network traffic.

Archiving. There is a risk in archiving data on a software system. The negatives include corruption or loss of data during a system failure, or unreadability of data due to technological obsolescence.

3. SCOPE OF SOFTWARE RISK MANAGEMENT

- Organization Risks
- Project Risks
- People Risks
- Process Risks
- Product Risks

4. PM3

The PM Solutions/PM3 has ten Knowledge Areas:

1. Project Integration Management
2. Project Scope Management
3. Project Time Management
4. Project Cost Management
5. Project Quality Management
6. Project Human Resource Management
7. Project Communications Management
8. Project Risk Management
9. Project Procurement Management
10. Project Stakeholder Management

The PM Solutions/PM3 has five Levels:

1. Level 1 Initial Process
2. Level 2 Structured Process and Standards
3. Level 3 Organizational Standards and Institutionalized Process
4. Level 4 Managed Process
5. Level 5 Optimizing Process

5. RISK ASSESSMENT ASPECT

- Risk Identification
- Risk Analysis

- Risk Prioritization

6. RISK CONTROL ASPECT

- Risk Planning
- Risk Resolution
- Risk Monitoring

7. RISK IDENTIFICATION

- Analogy
- Brainstorming
- Causal Mapping
- Checklist Analysis

8. STRATEGIES FOR RISK TREATMENT

Definition [Risk Level] [Refsdal, Solhaug, Stølen, 2015, Section 2.1]. The magnitude of a risk as derived from its likelihood and consequence.

Definition [Risk Treatment] [Refsdal, Solhaug, Stølen, 2015, Section 2.4.5]. An appropriate measure to reduce risk level.

- Acceptance
- Avoidance
- Reduction
- Mitigation
- Transference
- Contingency Measures