Cal State Fullerton

CPSC 254

Software Development With Open Source Systems - Project Development

Instructor: Tejaas Mukunda Reddy

9 STEPS TO A HASSLE FREE AND EFFECTIVE SOFTWARE DEVELOPMENT PROJECT

- Following these nine steps may be the ultimate secret weapon to winning business and successfully delivering new easy-to-use software that meets and exceeds expectations.
- Has your company developed entirely new software or added to software already in use throughout the organization and found the process cumbersome, frustrating, and sometimes not living up to expectations or meeting organizational goals? If so, the solution to a smooth and effective development program may be as easy as staffing a well-qualified project manager and adopting a proven development process.
- For any software development or other project initiative your company may be considering, it is critical to have in place and practice a set of effective and proven guidelines to ensure project success and delivery of the expected results: taking into consideration the role and responsibilities of a well-qualified project manager, knowledge of important business and financial aspects, and a step-by-step process that all contribute to the solid foundation and implementation of an effective project plan.

Developing a Practical Approach: The Role of the Project Manager

- When undertaking a software development project, the first element to consider is the establishment of a comprehensive yet practical approach to the initiative that ultimately will lead to a successful end result.
- The in-house project manager has a key role in ensuring each phase of the project is carried out as planned. The project manager is responsible for considering the potential risks involved with the project and how to avoid and resolve them, establishing and maintaining momentum throughout the project, ensuring individual project team member tasks are assigned appropriately and carried out according to specifications, and successfully addressing and resolving any conflicts that may arise during the length of the development project.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

Project Manager

- A well-qualified project manager is able to address what may seem to be an overwhelmingly complex process by developing an organized approach where the process is broken down into manageable individual tasks and understanding how to keep those involved in the project dedicated to the ultimate goal of meeting and even exceeding the expected end result.
- If the project manager dedicates the necessary time, effort, and resources to the preparation of an efficient, comprehensive, and practical approach, then the project team may progress with ease and confidence as they deliver on their individual tasks, having a solid foundation and strategic framework at the outset. Far too often, however, failures with such projects are the result of not only a poorly executed plan, but one that ultimately lacked the fundamental elements of a well-though-out approach rooted in adequate preparation and commitment from the project manager and project team.
- Designing a strategic plan means taking into consideration all aspects that can contribute to success or potential failure.

Embarking on the Initiative: Key Steps to Consider

- With a comprehensive approach and a competent project manager in place to guide the new software development initiative, there is another important element your organization may find helpful as you embark on the project: establishing specific steps that can be followed to project completion that are based on proven industry experience in such a project environment.
- Following are a set of practical guidelines to approach a software development project, established by two university professors and business consultants with specialised expertise in the computing, engineering, and general business environments.
- Dr. Gordon Scott Gehrs is an adjunct instructor at the Illinois Institute of Technology (IIT) and a business consultant for the Jules F. Knapp Entrepreneurship Center at IIT. Dr. Dorota Huizinga is associate dean of the College of Engineering and Computer Science and a professor in the Computer Science Department at California State University, Fullerton, as well as a frequent business seminar speaker, a business consultant, and co-author of Automated Defect Prevention: Best Practices in Software Management.
- Read on for nine key steps to consider as you embark on a software development project.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

9 Steps

Step #1: Conduct Feasibility Analysis

- According to Dr. Gehrs, a critical first step is to interview stakeholders in order to uncover whether a specific need exists, identify this exact need, and determine whether the proposed project can feasibly deliver the expected software development. Many times, this is the point at which an ROI study will be carried out in order to determine project costs and benefits, says Dr. Gehrs.

Step #2: Analyze and Determine Requirements

- When it comes to the next step of determining requirements, Dr. Gehrs believes a proper analysis should consist of interviews with end users and others who will be associated with the new software system. In addition, a thorough review and a keen understanding of user documents, business rules, and processes are keys to determining appropriate and necessary features and functionality. This is a valuable and significant step in the development process and the point at which such deliverables as those documents outlining the scope of the project and those detailing the software product requirement will be produced.
- Dr. Huizinga notes the importance of having the minimum technology infrastructure in place before beginning a software project, which include:
    - Desktops for development with an advanced integrated development environment suite
    - A server with a configuration management system for document tracking and version control
    - A staging server for integration testing and a production server for deployment of the final product
    - A requirement/task/defect tracking tool
    - An automated build system
    - A regression testing tool
    - An automated reporting system
    - Investing in the proper infrastructure is essential and will pay back quickly, asserts Dr. Huizinga. There are three key elements the proper infrastructure provides:
    - Product and project visibility
    - Automation of repetitive and mundane tasks
    - Facilitation of collaboration

Step #3: Consider Industry Best Practices

- When defining a software development process, consider proven industry best practices. Dr. Huizinga recommends a good, customized Agile process with emphasis on pictorial documentation both for requirements and technical documentation. It is important to follow a standard template and all activities should be traceable through a requirements/task/defect tool and shared document repository.

Step #4: Design

- During the design phase, the software architect, programmer, and/or developer may put together a detailed design document outlining exactly how the software will meet the specified requirements. Dr. Gehrs recommends the use of mock-ups to accompany the design document as a way of illustrating user-interface elements.
- In some cases, customization is required in order to meet specific, individual project needs. For example, Dr. Huizinga notes that this might include the use of specialized COTS (Commercial Off-The-Shelf) hardware and software components. The wide spectrum of products from databases to game engines is dictated by the market shift to customization of existing commercial applications to fit project needs rather than in-house development of such systems. According to Dr. Huizinga, COTS can offer higher quality because they are developed by vendors who specialize in systems that provide the required functionality and are well-tested by many users.

Step #5: Measuring and Tracking Progress

- Without the proper technology infrastructure in place, it is difficult to collect and measure key project data. Consequently, the software projects cannot be managed effectively, says Dr. Huizinga. Project indicators can help to ensure the prompt identification of potential or existing problems, therefore allowing them to be recognised and remedied in a timely manner. When observed over an extended period, notes Dr. Huizinga, these indicators can be used to determine product quality and deployment readiness.

Step #6: Development

- At the development phase, the design document is translated into a real piece of software. When prior careful planning has been executed, the software will match the requirements of the business driver that initiated the need for the project. Dr. Gehrs points out that development cycles may produce several versions of the software:
    - Alpha: preliminary feature/functionality only
    - Beta: used for internal testing or usability testing
    - Release Candidates: usually a very stable build that may need minor tweaks
    - Production Build or Gold Master: ready for release
    - Project managers need feedback on the user's navigational experience, task-completion times, ease of use, and other information related to the user interface and user-centric elements.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

Step #7: Addressing Automation

- Another key step is to ensure the automation of repetitive tasks:
    - Code builds;
        - Static code analysis scans
        - Regression tests
        - Collection of project, and product, related measures
- Dr. Huizinga believes that taking such measures reduces the error-prone human influence when the software is implemented. It also facilitates the use of best practices and collection of project-related data. All repetitive and mundane tasks should be automated whenever possible in any portion of the software life cycle, she adds.

Step #8: Testing

- As the project continues on through each phase and on to testing, a general progression of action is as follows: software features are laid out in some sort of list, scripts are written for each task the user might perform, and those features are tested to ensure they function properly. Dr. Gehrs points out that testing also may vary quite widely depending on the individual testing procedures adopted by the organisation. Testing can consist of several sub-stages as well, such as quality assurance and staging.
- Once the software is in general use, any bugs found at this point are addressed based on a criticality scale: urgent fixes are scheduled to be carried out as soon as possible. In addition, feature enhancements/changes may be slated for future upgrade versions.

Step #9: Gradual Implementation Practices

- Incremental implementation of the above practices is critical to success. The approach of gradually introducing change group by group and practice by practice is essential to achieving the desired organizational culture change, as change is unsettling, and there will always be some degree of resistance, points out Dr. Huizinga. Because of the complex nature of software projects and the technology involved, new software development warrants this systematic approach.
- Understanding the role of the project leader and importance of having well-thought-out development processes in place may be a company's only real competitive advantage in an increasingly competitive marketplace. It is the ultimate secret weapon to winning business and successfully delivering new easy-to-use software.
- With workable and disciplined software project guidelines and well-qualified project managers, your organization can't lose.

Agile

- Executive Brief, the technology management resource for business leaders, offers proven tips, techniques, and action plans that companies can use to better manage people, processes and tools - the keys to improving their business performance. To learn more, please visit: SoftServe United Blog
- © Executive Brief 2008 https://www.projectsmart.co.uk/9-steps-to-a-hassle-free-and-effective-software-development-project.php

Agile software development

- Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end users(s).[1] It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.[2]
- The term agile (sometimes written Agile)[3] was popularized, in this context, by the Manifesto for Agile Software Development.[4] The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.[5][6]
- While there is significant anecdotal evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations; some empirical studies have found no scientific evidence.

Agile Manifesto

- In 2001, Agile software development values[edit]
- Based on their combined experience of developing software and helping others do that, the seventeen signatories to the manifesto proclaimed that they value:[4]
  - Individuals and Interactions over processes and tools
  - Working Software over comprehensive documentation
  - Customer Collaboration over contract negotiation
  - Responding to Change over following a plan
- That is, while there is value in the items on the right, they value the items on the left more.
- As Scott Ambler elucidated:[19]
  - Tools and processes are important, but it is more important to have competent people working together effectively.
  - Good documentation is useful in helping people to understand how the software is built and how to use it, but the main point of development is to create software, not documentation.
  - A contract is important but is no substitute for working closely with customers to discover what they need.
  - A project plan is important, but it must not be too rigid to accommodate changes in technology or the environment, stakeholders' priorities, and people's understanding of the problem and its solution.
- Some of the authors formed the Agile Alliance, a non-profit organization that promotes software development according to the
- manifesto's values and principles. Introducing the manifesto on behalf of the Agile Alliance, Jim Highsmith said,
- The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term hacker.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

Agile Manifesto 12 Principles

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development
- Working software is delivered frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the primary measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

- **Pair programming**, an agile development technique used by XP. Note information radiators in the background.
- Iterative, incremental and evolutionary: Most agile development methods break product development work into small increments that minimize the amount of up-front planning and design. Iterations, or sprints, are short time frames (timeboxes) that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the product to adapt to changes quickly.[22] An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.[23] Multiple iterations might be required to release a product or new features.
- Working software is the primary measure of progress.
- **Efficient and face-to-face communication:** No matter which development method is followed, every team should include a customer representative (Product Owner in Scrum). This person is agreed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer questions throughout the iteration. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the return on investment(ROI) and ensuring alignment with customer needs and company goals.
- In agile software development, an information radiator is a (normally large) physical display located prominently near the development team, where passers-by can see it. It presents an up-to-date summary of the product development status.[24][25] A build light indicator may also be used to inform a team about the current status of their product development.
- Very short feedback loop and adaptation cycle
- A common characteristic in agile software development is the daily stand-up (also known as the daily scrum). In a brief session, team members report to each other what they did the previous day toward their team's iteration goal, what they intend to do today toward the goal, and any roadblocks or impediments they can see to the goal.
- **Quality focus:** Specific tools and techniques, such as continuous integration, automated unit testing, pair programming, test-driven development, design patterns, behavior-driven development, domain-driven design, code refactoring and other techniques are often used to improve quality and enhance product development agility. The idea is that the quality is built into the software.
- **Philosophy:** Compared to traditional software engineering, agile software development mainly targets complex systems and product development with dynamic, non-deterministic and non-linear characteristics. Accurate estimates, stable plans, and predictions are often hard to get in early stages, and confidence in them is likely to be low. Agile practitioners will seek to reduce the "leap-of-faith" that is needed before any evidence of value can be obtained.[29] Requirements and design are held to be emergent. Big up-front specifications would probably cause a lot of waste in such cases, i.e., are not economically sound. These basic arguments and previous industry experiences, learned from years of successes and failures, have helped shape agile development's favor of adaptive, iterative and evolutionary development.[30]

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

- **Adaptive vs. predictive**
- Development methods exist on a continuum from adaptive to predictive.[31] Agile software development methods lie on the adaptive side of this continuum. One key of adaptive development methods is a "Rolling Wave" approach to schedule planning, which identifies milestones but leaves flexibility in the path to reach them, and also allows for the milestones themselves to change.[32]
- Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team has difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method is about what will happen on that date. An adaptive team cannot report exactly what tasks they will do next week, but only which features they plan for next month. When asked about a release six months from now, an adaptive team might be able to report only the mission statement for the release, or a statement of expected value vs. cost.
- Predictive methods, in contrast, focus on analyzing and planning the future in detail and cater for known risks. In the extremes, a predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction. Predictive teams often institute a change control board to ensure they consider only the most valuable changes.
- Risk analysis can be used to choose between adaptive (agile or value-driven) and predictive (plan-driven) methods.[33] Barry Boehm and Richard Turner suggest that each side of the continuum has its own home ground, as follows:
- **Iterative vs. waterfall**
- One of the differences between agile software development methods and waterfall is the approach to quality and testing. In the waterfall model, there is always a separate testing phase after a build phase; however, in agile software development testing is completed in the same iteration as programming.
- Because testing is done in every iteration—which develops a small piece of the software—users can frequently use those new pieces of software and validate the value. After the users know the real value of the updated piece of software, they can make better decisions about the software's future. Having a value retrospective and software re-planning session in each iteration—Scrum typically has iterations of just two weeks—helps the team continuously adapt its plans so as to maximize the value it delivers. This follows a pattern similar to the PDCA cycle, as the work is planned, done, checked (in the review and retrospective), and any changes agreed are acted upon.
- This iterative approach supports a product rather than a project mindset. This provides greater flexibility throughout the development process; whereas on projects the requirements are defined and locked down from the very beginning, making it difficult to change them later. Iterative product development allows the software to evolve in response to changes in business environment or market requirements.[35]
- Because of the short iteration style of agile software development, it also has strong connections with the lean startup concept.

- **Code vs. Documentation**
- In a letter to IEEE Computer, Steven Rakitin expressed cynicism about agile software development, calling it "yet another attempt to undermine the discipline of software engineering" and translating "Working software over comprehensive documentation" as "We want to spend all our time coding. Remember, real programmers don't write documentation."[36]
- This is disputed by proponents of agile software development, who state that developers should write documentation if that is the best way to achieve the relevant goals, but that there are often better ways to achieve those goals than writing static documentation.[37] Scott Ambler states that documentation should be "Just Barely Good Enough" (JBGE),[38] that too much or comprehensive documentation would usually cause waste, and developers rarely trust detailed documentation because it's usually out of sync with code,[37] while too little documentation may also cause problems for maintenance, communication, learning and knowledge sharing. Alistair Cockburn wrote of the Crystal Clear method:
- Crystal considers development a series of co-operative games, and intends that the documentation is enough to help the next win at the next game. The work products for Crystal include use cases, risk list, iteration plan, core domain models, and design notes to inform on choices...however there are no templates for these documents and descriptions are necessarily vague, but the objective is clear, just enough documentation for the next game. I always tend to characterize this to my team as: what would you want to know if you joined the team tomorrow.
- Alistair Cockburn.
- Agile software development methods
- Agile software development methods support a broad range of the software development life cycle.[40] Some focus on the practices (e.g., XP, pragmatic programming, agile modeling), while some focus on managing the flow of work (e.g., Scrum, Kanban). Some support activities for requirements specification and development (e.g., FDD), while some seek to cover the full development life cycle (e.g., DSDM, RUP).

- Agile software development practices[edit]
- Agile software development is supported by a number of concrete practices, covering areas like requirements, design, modeling, coding, testing, planning, risk management, process, quality, etc. Some notable agile software development practices include:
- Certain practices are described at the end of this document. They are identified by *
    - Acceptance test-driven development (ATDD)
    - Agile modeling * & Agile testing
    - Backlogs (Product and Sprint)
    - Behavior-driven development (BDD)
    - Business analyst designer method (BADM)[41]
    - Continuous integration (CI) & Cross-functional team
    - Domain-driven design (DDD)
    - Information radiators (scrum board, task board, visual management board, burndown chart)
    - Iterative and incremental development (IID)
    - Kanban *
    - Low-code development platforms
    - Pair programming & Planning poker
    - Refactoring & Retrospective
    - Scrum events (sprint planning, daily scrum, sprint review and retrospective) *
    - Story-driven modeling
    - Test-driven development (TDD)
    - Timeboxing
    - User story & User story mapping
    - Velocity tracking

- The Agile Alliance has provided a comprehensive online guide to applying these and other practices.[42]
- **Method tailoring:** In the literature, different terms refer to the notion of method adaptation, including 'method tailoring', 'method fragment adaptation' and 'situational method engineering'. Method tailoring is defined as:
- A process or capability in which human agents determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments.
- Mehmet Nafiz Aydin et al., An Agile Information Systems Development Method in use[43]
- Situation-appropriateness should be considered as a distinguishing characteristic between agile methods and more plan-driven software development methods, with agile methods allowing product development teams to adapt working practices according to the needs of individual products.[44][43] Potentially, most agile methods could be suitable for method tailoring,[40] such as DSDM tailored in a CMM context.[45] and XP tailored with the Rule Description Practices (RDP) technique.[46][47] Not all agile proponents agree, however, with Schwaber noting "that is how we got into trouble in the first place, thinking that the problem was not having a perfect methodology. Efforts [should] center on the changes [needed] in the enterprise".[48] Bas Vodde reinforced this viewpoint, suggesting that unlike traditional, large methodologies that require you to pick and choose elements, Scrum provides the basics on top of which you add additional elements to localise and contextualise its use.[49]
- **Large-scale, offshore and distributed:** Agile software development has been widely seen as highly suited to certain types of environments, including small teams of experts working on greenfield projects, and the challenges and limitations encountered in the adoption of agile software development methods in a large organization with legacy infrastructure are well-documented and understood.
- In response, a range of strategies and patterns has evolved for overcoming challenges with large-scale development efforts (>20 developers)[52][53] or distributed (non-colocated) development teams, amongst other challenges; and there are now several recognized frameworks that seek to mitigate or avoid these challenges.
    - Scaled agile framework (SAFe),[56] Dean Leffingwell inter alia
    - Disciplined agile delivery (DAD), Scott Ambler inter alia & Large-scale scrum (LeSS), Craig Larman and Bas Vodde
    - Nexus (scaled professional Scrum),[57] Ken Schwaber
    - Scrum at Scale,[58] Jeff Sutherland, Alex Brown & Enterprise Scrum,[59] Mike Beedle
    - Setchu (Scrum-based lightweight framework),[60] Michael Ebbage
    - Xscale[61] & Agile path
    - Holistic Software Development

- There are many conflicting viewpoints on whether all of these are effective or indeed fit the definition of agile development, and this remains an active and ongoing area of research.[52][64]
- When agile software development is applied in a distributed setting (with teams dispersed across multiple business locations), it is commonly referred to as distributed agile development. The goal is to leverage the unique benefits offered by each approach. Distributed development allow organizations to build software by strategically setting up teams in different parts of the globe, virtually building software round-the-clock (more commonly referred to as follow-the-sun model). On the other hand, agile development provides increased transparency, continuous feedback and more flexibility when responding to changes.

Regulated domains

- Agile software development methods were initially seen as best suitable for non-critical product developments, thereby excluded from use in regulated domains such as medical devices, pharmaceutical, financial, nuclear systems, automotive, and avionics sectors, etc. However, in the last several years, there have been several initiatives for the adaptation of agile methods for these domains.
- There are numerous standards that may apply in regulated domains, including ISO 26262, ISO 9000, ISO 9001, and ISO/IEC 15504. A number of key concerns are of particular importance in regulated domains:
    - Quality assurance (QA): Systematic and inherent quality management underpinning a controlled professional process and reliability and correctness of product.
    - Safety and security: Formal planning and risk management to mitigate safety risks for users and securely protecting users from unintentional and malicious misuse.
    - Traceability: Documentation providing auditable evidence of regulatory compliance and facilitating traceability and investigation of problems.
    - Verification and Validation (V&V): Embedded throughout the software development process (e.g. user requirements specification, functional specification, design specification, code review, unit tests, integration tests, system tests).
- The Scrum framework in particular has received considerable attention. Two derived methods have been defined: R-Scrum (Regulated Scrum)[65] and SafeScrum.[68][69]

Experience and adoption

- Although agile software development methods can be used with any programming paradigm or language in practice, they were originally closely associated with object-oriented environments such as Smalltalk and Lisp and later Java. The initial adopters of agile methods were usually small to medium-sized teams working on unprecedented systems with requirements that were difficult to finalize and likely to change as the system was being developed. This section describes common problems that organizations encounter when they try to adopt agile software development methods as well as various techniques to measure the quality and performance of agile teams.[70]

Measuring agility: The best agile practitioners have always emphasized sound engineering principles. As a result, there are a number of best practices and tools for measuring the performance of agile software development and teams.

Internal assessments: The Agility measurement index, amongst others, rates developments against five dimensions of product development (duration, risk, novelty, effort, and interaction).

- Other techniques are based on measurable goals[73] and one study suggests that velocity can be used as a metric of agility.[74]
- There are also agile self-assessments to determine whether a team is using agile software development practices (Nokia test,[75] Karlskrona test,[76] 42 points test[77]).

Public surveys: One of the early studies reporting gains in quality, productivity, and business satisfaction by using agile software developments methods was a survey conducted by Shine Technologies from November 2002 to January 2003.[78]

- A similar survey, the State of Agile, is conducted every year starting in 2006 with thousands of participants from around the software development community. This tracks trends on the benefits of agility, lessons learned, and good practices. Each survey has reported increasing numbers saying that agile software development helps them deliver software faster; improves their ability to manage changing customer priorities; and increases their productivity.[79] Surveys have also consistently shown better results with agile product development methods compared to classical project management.[80][81] In balance, there are reports that some feel that agile development methods are still too young to enable extensive academic research of their success.[82]

Common agile software development pitfalls: Organizations and teams implementing agile software development often face difficulties transitioning from more traditional methods such as waterfall development, such as teams having an agile process forced on them.[83] These are often termed agile anti-patterns or more commonly agile smells. Below are some common examples:

Lack of overall product design: A goal of agile software development is to focus more on producing working software and less on documentation. This is in contrast to waterfall models where the process is often highly controlled and minor changes to the system require significant revision of supporting documentation. However, this does not justify completely doing without any analysis or design at all. Failure to pay attention to design can cause a team to proceed rapidly at first but then to have significant rework required as they attempt to scale up the system. One of the key features of agile software development is that it is iterative. When done correctly design emerges as the system is developed and commonalities and opportunities for re-use are discovered.

Adding stories to an iteration in progress: In agile software development, stories (similar to use case descriptions) are typically used to define requirements and an iteration is a short period of time during which the team commits to specific goals.[85] Adding stories to an iteration in progress is detrimental to a good flow of work. These should be added to the product backlog and prioritized for a subsequent iteration or in rare cases the iteration could be cancelled.[86]

- This does not mean that a story cannot expand. Teams must deal with new information, which may produce additional tasks for a story. If the new information prevents the story from being completed during the iteration, then it should be carried over to a subsequent iteration. However, it should be prioritized against all remaining stories, as the new information may have changed the story's original priority.

Lack of sponsor support: Agile software development is often implemented as a grassroots effort in organizations by software development teams trying to optimize their development processes and ensure consistency in the software development life cycle. By not having sponsor support, teams may face difficulties and resistance from business partners, other development teams and management. Additionally, they may suffer without appropriate funding and resources.[87] This increases the likelihood of failure.[88]

Insufficient training: A survey performed by VersionOne found respondents cited insufficient training as the most significant cause for failed agile implementations[89] Teams have fallen into the trap of assuming the reduced processes of agile software development compared to other methodologies such as waterfall means that there are no actual rules for agile software development. Agile software development is a set of prescribed methodologies, and training/practice is a requirement.

Product owner role is not properly filled: The product owner is responsible for representing the business in the development activity and is often the most demanding role. A common mistake is to have the product owner role filled by someone from the development team. This requires the team to make its own decisions on prioritization without real feedback from the business. They try to solve business issues internally or delay work as they reach outside the team for direction. This often leads to distraction and a breakdown in collaboration.

Teams are not focused: Agile software development requires teams to meet product commitments, which means they should focus only on work for that product. However, team members who appear to have spare capacity are often expected to take on other work, which makes it difficult for them to help complete the work to which their team had committed.[92]

Excessive preparation/planning: Teams may fall into the trap of spending too much time preparing or planning. This is a common trap for teams less familiar with agile software development where the teams feel obliged to have a complete understanding and specification of all stories. Teams should be prepared to move forward only with those stories in which they have confidence, then during the iteration continue to discover and prepare work for subsequent iterations (often referred to as backlog refinement or grooming).

Problem-solving in the daily standup: A daily standup should be a focused, timely meeting where all team members disseminate information. If problem-solving occurs, it often can only involve certain team members and potentially is not the best use of the entire team's time. If during the daily standup the team starts diving into problem-solving, it should be set aside until a sub-team can discuss, usually immediately after the standup completes.[93]

Assigning tasks: One of the intended benefits of agile software development is to empower the team to make choices, as they are closest to the problem. Additionally, they should make choices as close to implementation as possible, to use more timely information in the decision. If team members are assigned tasks by others or too early in the process, the benefits of localized and timely decision making can be lost.[94]

Being assigned work also constrains team members into certain roles (for example, team member A must always do the database work), which limits opportunities for cross-training.[94] Team members themselves can choose to take on tasks that stretch their abilities and provide cross-training opportunities.

Scrum master as a contributor: Another common pitfall is for a scrum master to act as a contributor. While not prohibited by the Scrum methodology, the scrum master needs to ensure they have the capacity to act in the role of scrum master first and not working on development tasks. A scrum master's role is to facilitate the process rather than create the product.

Having the scrum master also multitasking may result in too many context switches to be productive. Additionally, as a scrum master is responsible for ensuring roadblocks are removed so that the team can make forward progress, the benefit gained by individual tasks moving forward may not outweigh roadblocks that are deferred due to lack of capacity.

Lack of test automation: Due to the iterative nature of agile development, multiple rounds of testing are often needed. Automated testing helps reduce the impact of repeated unit, integration, and regression tests and frees developers and testers to focus on higher value work.[96]

- Test automation also supports continued refactoring required by iterative software development. Allowing a developer to quickly run tests to confirm refactoring has not modified the functionality of the application may reduce the workload and increase confidence that cleanup efforts have not introduced new defects.

Allowing technical debt to build up: Focusing on delivering new functionality may result in increased technical debt. The team must allow themselves time for defect remediation and refactoring. Technical debt hinders planning abilities by increasing the amount of unscheduled work as production defects distract the team from further progress.

- As the system evolves it is important to refactor as entropy of the system naturally increases.[98] Over time the lack of constant maintenance causes increasing defects and development costs.

Attempting to take on too much in an iteration: A common misconception is that agile software development allows continuous change, however an iteration backlog is an agreement of what work can be completed during an iteration.[99] Having too much work-in-progress (WIP) results in inefficiencies such as context-switching and queueing.[100] The team must avoid feeling pressured into taking on additional work.

Fixed time, resources, scope, and quality: Agile software development fixes time (iteration duration), quality, and ideally resources in advance (though maintaining fixed resources may be difficult if developers are often pulled away from tasks to handle production incidents), while the scope remains variable. The customer or product owner often push for a fixed scope for an iteration. However, teams should be reluctant to commit to the locked time, resources and scope (commonly known as the project management triangle). Efforts to add scope to the fixed time and resources of agile software development may result in decreased quality.

Developer burnout: Due to the focused pace and continuous nature of agile practices, there is a heightened risk of burnout among member of the delivery team.

Agile management:

The term "agile management" is applied to an iterative, incremental method of managing the design and build activities of engineering, information technology and other business areas that aim to provide new product or service development in a highly flexible and interactive manner, based on the principles expressed in the Manifesto for Agile Software Development.

Agile X techniques may also be called extreme project management. It is a variant of iterative life cycle[105] where deliverables are submitted in stages. The main difference between agile and iterative development is that agile methods complete small portions of the deliverables in each delivery cycle (iteration),[106]while iterative methods evolve the entire set of deliverables over time, completing them near the end of the project. Both iterative and agile methods were developed as a reaction to various obstacles that developed in more sequential forms of project organization. For example, as technology projects grow in complexity, end users tend to have difficulty defining the long-term requirements without being able to view progressive prototypes. Projects that develop in iterations can constantly gather feedback to help refine those requirements.

Agile management also offers a simple framework promoting communication and reflection on past work amongst team members.[107] Teams who were using traditional waterfall planning and adopted the agile way of development typically go through a transformation phase and often take help from agile coaches who help guide the teams through a smooth transformation. There are typically two styles of agile coaching: push-based and pull-based agile coaching. Agile management approaches have also been employed and adapted to the business and government sectors. For example, within the federal government of the United States, the United States Agency for International Development (USAID) is employing a collaborative project management approach that focuses on incorporating collaborating, learning and adapting (CLA) strategies to iterate and adapt programming.

Agile methods are mentioned in the Guide to the Project Management Body of Knowledge (PMBOK Guide) under the Project Lifecycle definition:

Adaptive project life cycle, a project life cycle, also known as change-driven or agile methods, that is intended to facilitate change and require a high degree of ongoing stakeholder involvement. Adaptive life cycles are also iterative and incremental, but differ in that iterations are very rapid (usually 2-4 weeks in length) and are fixed in time and resources.

Applications outside software development

- Agile Brazil 2014 conference: According to Jean-Loup Richet (Research Fellow at ESSEC Institute for Strategic Innovation & Services) "this approach can be leveraged effectively for non-software products and for project management in general, especially in areas of innovation and uncertainty." The end result is a product or project that best meets current customer needs and is delivered with minimal costs, waste, and time, enabling companies to achieve bottom line gains earlier than via traditional approaches.[110]
- Agile software development methods have been extensively used for development of software products and some of them use certain characteristics of software, such as object technologies.[111] However, these techniques can be applied to the development of non-software products, such as computers, motor vehicles,[112] medical devices, food, clothing, and music;[113]see Flexible product development. Agile software development methods have been used in non-development IT infrastructure deployments and migrations. Some of the wider principles of agile software development have also found application in general management[114] (e.g., strategy, governance, risk, finance) under the terms business agility or agile business management.
- Under an agile business management model, agile software development techniques, practices, principles and values are expressed across five domains.[115]
    - Integrated customer engagement: to embed customers within any delivery process to share accountability for product/service delivery.
    - Facilitation-based management: adopting agile management models, like the role of Scrum Master, to facilitate the day-to-day operation of teams.
    - Agile work practices: adopting specific iterative and incremental work practices such as Scrum, Kanban, test-driven development or feature-driven development across all business functions (from sales, human resources, finance[116] and marketing).
    - An enabling organisational structure: with a focus on staff engagement, personal autonomy and outcomes based governance.
    - Applications of agile process (along with DevOps and lean manufacturing), to data analytics, business intelligence, big data, and data science is called DataOps
- Agile software development paradigms can be used in other areas of life such as raising children. Its success in child development might be founded on some basic management principles; communication, adaptation, and awareness. Bruce Feiler has claimed that the basic agile development paradigms can be applied to household management and raising children. In his TED Talk "Agile programming – for your family",[117] these paradigms brought significant changes to his household environment, such as the kids doing dishes, taking out the trash, and decreasing his children's emotional outbreaks, which inadvertently increased their emotional stability.

Criticism

- Agile practices can be inefficient in large organizations and certain types of developments.[118] Many organizations believe that agile software development methodologies are too extreme and adopt a Hybrid approach [119] that mixes elements of agile software development and plan-driven approaches.[120] Some methods, such as dynamic systems development method (DSDM) attempt this in a disciplined way, without sacrificing fundamental principles.
- The increasing adoption of agile practices has also been criticized as being a management fad that simply describes existing good practices under new jargon, promotes a "one size fits all" mindset towards development strategies, and wrongly emphasizes method over results.[121]
- Alistair Cockburn organized a celebration of the 10th anniversary of the Manifesto for Agile Software Development in Snowbird, Utah on 12 February 2011, gathering some 30+ people who had been involved at the original meeting and since. A list of about 20 elephants in the room ("undiscussable" agile topics/issues) were collected, including aspects: the alliances, failures and limitations of agile software development practices and context (possible causes: commercial interests, decontextualization, no obvious way to make progress based on failure, limited objective evidence, cognitive biases and reasoning fallacies), politics and culture.[122] As Philippe Kruchten wrote:
- The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant. But I have no doubts that it will mature further, become more open to the outside world, more reflective, and therefore, more effective.
- Philippe Kruchten[122] https://en.wikipedia.org/wiki/Agile_software_development
- **Agile modeling (AM)** is a methodology for modeling and documenting software systems based on best practices. It is a collection of values and principles, that can be applied on an (agile) software development project. This methodology is more flexible than traditional modeling methods, making it a better fit in a fast changing environment.[1] It is part of the agile software development tool kit.
- Agile modeling is a supplement to other agile development methodologies such as Scrum, extreme programming (XP), and Rational Unified Process (RUP). It is explicitly included as part of the disciplined agile delivery (DAD) framework. As per 2011 stats, agile modeling accounted for 1% of all agile software development.[2]
- https://en.wikipedia.org/wiki/Agile_modeling

Kanban (development)

- From Wikipedia, the free encyclopedia
- This article is about the process-management and improvement method. For the lean-manufacturing process, see Kanban. Kanban is a lean method to manage and improve work across human systems. This approach aims to manage work by balancing the demands with available capacity, and improving the handling of system level bottlenecks.
- Work items are visualized to give participants a view of progress and process, from start to finish usually via a Kanban board. Work is pulled as capacity permits, rather than work being pushed into the process when requested.
- In knowledge work and software development, this provides a visual process management system which aids decision-making about what, when and how much to produce. Although the underlying Kanban method originated in lean manufacturing[1] (inspired by the Toyota Production System[2]) it is used mainly for software development and technology related work. However Kanban can be applied to any area of work, and it can even be combined with other methods or frameworks such as Scrum.[3]
- https://en.wikipedia.org/wiki/Kanban_(development)

- **"RAD tool"** and **"Rapid Application Development Tool"** redirect here. For development tools focused on making graphical user interfaces, see graphical user interface builder. Rapid-application development (RAD) is both a general term, used to refer to adaptive software development approaches, as well as the name for James Martin's approach to rapid development. In general, RAD approaches to software development put less emphasis on planning and more emphasis on an adaptive process. Prototypes are often used in addition to or sometimes even in place of design specifications.
- RAD is especially well suited for (although not limited to) developing software that is driven by user interface requirements. Graphical user interface builders are often called rapid application development tools. Other approaches to rapid development include the adaptive, agile, spiral, and unified models.
- https://en.wikipedia.org/wiki/Rapid_application_development

Scrum (software development)

- Scrum is an agile framework for managing work with an emphasis on software development. It is designed for teams of three to nine developers who break their work into actions that can be completed within timeboxed iterations, called sprints (typically two-weeks) and track progress and re-plan in 15-minute stand-up meetings, called daily scrums.[1][2] Approaches to coordinating the work of multiple scrum teams in larger organizations include Large-Scale Scrum, Scaled Agile Framework (SAFe) and Scrum of Scrums, among others.

Key ideas:

- Scrum is an iterative and incremental framework for managing product development.[3][4] It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal",[5] challenges assumptions of the "traditional, sequential approach"[5] to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.
- A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need (often called requirements volatility[6]) and that there will be unpredictable challenges—for which a predictive or planned approach is not suited. As such, Scrum adopts an evidence-based empirical approach—accepting that the problem cannot be fully understood or defined up front, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions.
- Many of the terms used in Scrum (e.g., scrum master) are typically written with leading capitals (i.e., Scrum Master) or as conjoint words written in camel case(i.e., ScrumMaster). To maintain an encyclopedic tone, however, this article uses normal sentence case for these terms—unless they are recognized marks (such as Certified Scrum Master). This is occasionally seen written in all-capitals, as SCRUM.[7] The word is not an acronym, so this is not correct; however, it likely arose due to an early paper by Ken Schwaber which capitalized SCRUM in its title.
- While the trademark on the term Scrum itself has been allowed to lapse, so that it is deemed as owned by the wider community rather than an individual,[9] the leading capital is retained—except when used with other words (as in daily scrum or scrum team). - https://en.wikipedia.org/wiki/Scrum_(software_development)

Pair programming

- Two co-workers pair programming
- Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator,[1] reviews each line of code as it is typed in. The two programmers switch roles frequently.
- While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address. This is intended to free the driver to focus all of their attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

Economics

- Pair programming increases the man-hours required to deliver code compared to programmers working individually. Experiments yielded diverse results, suggesting increases of between 15% and 100%.[2] However, the resulting code has about 15% fewer defects.[3] Along with code development time, other factors like field support costs and quality assurance also figure in to the return on investment. Pair programming might theoretically offset these expenses by reducing defects in the programs.
- Design quality: A system with two programmers possesses greater potential for the generation of more diverse solutions to problems for three reasons:
    - the programmers bring different prior experiences to the task;
    - they may access information relevant to the task in different ways;
    - they stand in different relationships to the problem by virtue of their functional roles.
- In an attempt to share goals and plans, the programmers must overtly negotiate a shared course of action when a conflict arises between them. In doing so, they consider a larger number of ways of solving the problem than a single programmer alone might do. This significantly improves the design quality of the program as it reduces the chances of selecting a poor method.[4]
- Satisfaction: In an online survey of pair programmers, 96% of them stated that they enjoyed their work more than when they programmed alone and 95% said that they were more confident in their solutions when they pair programmed.[5]
- Learning: Knowledge is constantly shared between pair programmers, whether in the industry or in a classroom, many sources suggest that students show higher confidence when programming in pairs,[5] and many learn whether it be from tips on programming language rules to overall design skill.[6] In "promiscuous pairing", each programmer communicates and works with all the other programmers on the team rather than pairing only with one partner, which causes knowledge of the system to spread throughout the whole team.[3] Pair programming allows programmers to examine their partner's code and provide feedback which is necessary to increase their own ability to develop monitoring mechanisms for their own learning activities.[6]

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

Pair Programming

Team-building and communication

- Pair programming allows team members to share problems and solutions quickly making them less likely to have hidden agendas from each other. This helps pair programmers to learn to communicate more easily. “This raises the communication bandwidth and frequency within the project, increasing overall information flow within the team.”

Studies: There are both empirical studies and meta-analyses of pair programming. The empirical studies tend to examine the level of productivity and the quality of the code, while meta-analyses may focus on biases introduced by the process of testing and publishing.

- A meta-analysis found pairs typically consider more design alternatives than programmers working alone, arrive at simpler more maintainable designs, and catch design defects earlier. However, it raised concerns its findings may have been influenced by "signs of publication bias among published studies on pair programming". It concluded that "pair programming is not uniformly beneficial or effective".[7]
- Although pair programmers may complete a task faster than a solo programmer, the total number of man-hours increases.[2] A manager would have to balance faster completion of the work and reduced testing and debugging time against the higher cost of coding. The relative weight of these factors can vary by project and task.
- The benefit of pairing is greatest on tasks that the programmers do not fully understand before they begin: that is, challenging tasks that call for creativity and sophistication, and for novices as compared to experts.[2] Pair programming could be helpful for attaining high quality and correctness on complex programming tasks, but it would also increase the development effort (cost) significantly.[7]
- On simple tasks, which the pair already fully understands, pairing results in a net drop in productivity.[2][8] It may reduce the code development time but also risks reducing the quality of the program.[7] Productivity can also drop when novice–novice pairing is used without sufficient availability of a mentor to coach them.[9]

Indicators of non-performance: There are indicators that a pair is not performing well:

- Disengagement may present as one of the members physically withdraws away from the keyboard, accesses email, or even falls asleep.
- The "Watch the Master" phenomenon can arise if one member is more experienced than the other. In this situation, the junior member may take the observer role, deferring to the senior member of the pair for the majority of coding activity. This can easily lead to disengagement.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn