International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 2, Issue 3, 91-103, 2021
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V2I3P110

Original Article

Agile Development in Practice: From Intern to Contributor

Shiva Santosh Allenki[1], Ramesh Korutla[2]
[1]Software Engineer at Children's Health Dallas, USA.
[2]Senior Lead Java Engineer at Children's Health Dallas, Texas, USA.

Abstract *- Agile development has changed how modern software teams work by putting more emphasis on being more flexible, working together & always becoming better. This article talks about the main principles of Agile techniques, such as iterative development, flexible planning & delivery that focuses on the client. These are all important for effective software engineering. It shows how important it is for engineers to start adopting these Agile methods early on in their careers, when the best way to learn is by doing. Moving from intern to active contributor in Agile teams requires not just a change in skills but also a shift in how you think. You can't simply follow any orders; you have to actively participate in these sprints, retrospectives & many cross-functional problem-solving exercises. This article describes systematic ways to help new people fit into these Agile settings, including mentoring & progress tracking using metrics like velocity, quality & the cooperation. These methods are based on actual life case studies & genuine team experiences. The findings demonstrate that interns who worked with more than one Agile framework are more adaptable, better at communicating & have a better understanding of their shared duties. As kids become previous & begin to contribute, they learn to always provide value & be more flexible. This article says that Agile development improves their technical skills & builds resilience, curiosity & cooperation, which are all important traits for success in the present day's software industry.*

Keywords *- Agile Development, Scrum, Kanban, Internship, Software Engineering, Continuous Integration, Collaboration, Contributor Growth, DevOps, Iterative Development.*

1. Introduction

1.1. Background

Software development has changed a lot in the last twenty years. Agile techniques are a huge shift since they focus more on working together, being more flexible, and making quick adjustments than on rigorous planning and a lot of documentation. The Agile Manifesto was published in 2001, which officially started the transformation. Seventeen experienced software engineers got together to question established project management approaches that often made things take longer and inhibited innovation. Their motto spelled out four main values: putting people & relationships above rules & the technology, choosing functional software over long documentation, putting customer participation above contracts & being flexible instead of sticking to a set plan.

These ideas changed the ways that software teams worked on their projects. Traditional "waterfall" methods, which rely on their planning, development & these testing stages that happen one after the other, sometimes caused long delays & customers who were unhappy with the results. Agile, on the other hand, encourages development in these steps, getting feedback often & always becoming better. Teams began working in short cycles, or "sprints," which let them quickly respond to customer feedback & deliver small, functional pieces of software.

Agile has grown into a number of frameworks. The first was Scrum, which set up particular roles like Product Owner, Scrum Master & Development Team. The most recent is Kanban, which focuses on visual workflow management & continuous delivery. The Scaled Agile Framework (SAFe) was created to help huge companies use Agile ideas across many other teams & many departments, making sure that everyone works together & is on the same page. These frameworks show how Agile has grown from a short manifesto to a full ecosystem of these methods designed to meet the needs of different organizations.

Agile is not just a method for making software; it's a way of life. It affects how successfully businesses come up with the latest ideas, operate together & adapt to changes in the market. Agile rituals like daily stand-ups, sprint reviews & retrospectives are particularly significant in the present day's businesses. Agile has transformed the way people operate by encouraging their openness, responsibility & always learning, in addition to its technical elements. Many firms in other areas, including banking, healthcare & manufacturing, have embraced Agile ideals to remain competitive in a digital environment that is changing rapidly.

Agile is more than just a process; it's a style of working together that can change & is centered on what the customer requires. As Agile develops, the latest problems arise, especially when it comes to preparing the next generation of engineers to succeed in this fast-paced environment.

1.2. Challenges

Agile has altered how software is built, but it's still challenging to use it well, particularly for rookie engineers & interns. These new people, who are usually recent graduates or trainees, have a lot of problems that make it challenging for them to join in with these Agile teams.

One big problem is that there isn't enough actual world project experience. In school, students generally focus on theoretical ideas, programming basics & individual projects instead of working together on projects that are always changing. Interns who begin working in these Agile organizations may have a hard time understanding user stories, sprint planning & backlog prioritization since these ideas are seldom taught in school.

Getting used to iterative procedures is another problem. Agile is quite different from the sequential way that most students are used to. It focuses on small, gradual changes & constant feedback. Interns may first struggle to acclimate to the rapid pace of change, constant context switching & the need of delivering quantifiable results within limited periods.

It's hard to talk to each other & become involved in team rituals. Agile works best when everyone on the team can talk to each other openly, including in the daily stand-ups, sprint reviews & retrospectives, when everyone shares their opinions and updates. Interns frequently don't want to say what they think because they're afraid that their lack of experience would make others think less of their skills. This might lead to not enough involvement, which not only limits their learning but also makes them very less visible to the rest of the team.

Interns also need to be able to balance their schoolwork with their job duties. In agile companies, people need to keep learning & taking on the latest responsibilities. At the same time, interns need to learn how to use the latest tools, frameworks & technologies. It might be hard to find the right balance between learning and giving back without the right guidance.

In the end, there are problems with setting up the tools & the environment. Agile teams employ a lot of different tools to keep track of their work, such as issue tracking systems like Jira & version control systems like Git. People who are new to coding may have trouble setting up tools, following workflow rules & using best practices while working with others. These technology problems might make it very harder to get new employees up to speed and lower production.

To sum up, Agile wants to give teams more authority, yet going from being a student to an active participant might be scary for those who are just beginning out in their careers.

1.3. Problem Statement

Even though Agile approaches are widely used & have been demonstrated to work, interns and entry-level engineers sometimes have trouble making important contributions to Agile teams. The cause of this problem is the difference between what students learn in school & what businesses need. Most computer science programs teach Agile ideas in a theoretical way, but only a handful provide students the hands-on, collaborative experience they need to do well in actual Agile situations.

There are several ways to notice this difference. Interns typically lack confidence while having sprint talks, have trouble managing their workloads during short iteration cycles, and can't turn user stories that are business-related into these technical tasks. Without formal mentorship or regular feedback, their growth stops & they don't reach their full potential.

Also, limiting access to actual time collaborative tools or simulated Agile environments makes the difference even bigger. Interns may understand the idea of Agile, but they may not be able to use it well in actual life. As a result, businesses have trouble integrating new employees into existing Agile teams & interns run the risk of being assigned to less important tasks instead of being able to work on many important projects.

This research investigates the potential of structured Agile onboarding programs, with progressive responsibility models, to bridge this gap. By giving interns organized access to collaborative tools, consistent exposure to Agile processes & education that focuses on mentoring, organizations can turn them from passive learners into productive contributors.

1.4. Motivation

The motivation for this inquiry is both practical & strategic. Modern companies rely heavily on Agile teams to come up with the latest ideas, speed up time to market, and make customers happier. The success of Agile frameworks depends on how well the team works together, which includes not just senior developers but also interns & new engineers.

As technology moves quickly, companies always have to figure out how to quickly integrate new employees into these Agile frameworks. Interns typically don't get the quick collaboration, feedback & iterative learning they need for Agile teams during traditional onboarding. As a consequence, businesses spend a lot of time & money getting new employees used to their jobs, which leads to the higher onboarding expenses & a longer time to get up to speed.

Letting interns take the lead right away eliminates these kinds of problems. When interns learn about Agile ideas via hands-on experience instead of just reading about them, they not only get results that can be measured, but they also feel a strong sense of ownership & the engagement. This is good for both the company & the person: the employer gets a more skilled & motivated worker, and the intern gains the skills and confidence they need for the future jobs.

Also, coordinated Agile onboarding helps connect academics & business. It helps students go from educational settings that focus on coding assignments done alone to professional settings where working together, talking to each other & making things better over time are all very important. These kinds of initiatives help improve the culture of a company by encouraging their developers to keep learning from the beginning of their careers.

This research asserts that Agile success relies not just on frameworks & technology but also on people. Companies that turn interns into contributors create a workforce that is ready for the future & able to keep creativity, collaboration & the flexibility alive, which are the primary ideas behind Agile.

2. Literature Review

This study integrates current insights regarding these Agile frameworks, academic viewpoints on learning & collaboration within their Agile teams, the organization of internships and onboarding for practical skill enhancement, the application of Agile in educational contexts & the advancement of novices to competent contributors. It continues by highlighting a deficiency: despite our extensive understanding of Agile methodology & internships, there is a lack of a well defined, research-backed framework for "Agile immersion for interns."

2.1. Agile frameworks: the landscape in brief

Agile is a broad term that includes methods that focus on getting their feedback quickly, working closely together & delivering in small steps. In practice, a lot of frameworks are used:

Scrum puts a lot of emphasis on having fixed-length sprints, a prioritized backlog & well-defined responsibilities for the product owner, scrum master & the developers. The events planning, daily scrum, review & retrospective set a regular schedule for assessment & change. Scrum works because it is open & has a rhythm: tasks can be seen, have a time limit & are checked on a frequent basis.

Extreme Programming (XP) puts a lot of emphasis on working together & being good at technology. Pair programming, test-driven development, continuous integration & shared code ownership are all ways to directly deal with many quality problems. XP's focus on technical issues makes Scrum's procedural structure better. In fact, some teams informally combine the two methods by using Scrum for planning & XP for coding.

Kanban puts flow ahead of time limits. There is a board with tasks on it, work-in-progress (WIP) is limited & teams keep an eye on cycle time to avoid these bottlenecks. Kanban is easy to learn & use in small steps, which makes it useful in places where things are very unpredictable or when work is always going on.

Lean software development uses Lean concepts from manufacturing, such as getting rid of waste, improving learning, instilling quality, speeding up delivery, delaying commitment, honoring people & optimizing the entire. Lean encourages a way of thinking that affects how decisions are made, such as making handoffs easier & cutting down on lineups to improve their system-wide throughput.

In nature, hybrid models are common. Teams may use Kanban to see how their work is going & they can combine Scrum's sprints & rituals with XP's testing and pair programming. Large companies utilize either formal or improvised by their scaling

methods to keep numerous teams in sync while staying true to the basic concepts of Agile. Hybrids come up for practical reasons: different products, rules, or team maturity demand a tailored approach instead of a one-size-fits-all framework.

Three elements keep coming up in these frameworks: (1) short feedback loops that show learning quickly, (2) transparency that helps people stay on track with their goals, and (3) integrated reflection that helps teams become better all the time. These components are essential for the swift & safe assimilation of newcomers, especially interns.

2.2. Academic perspectives on Agile learning and collaboration

Research on Agile teams consistently underscores the relationship between rapid feedback & accelerated learning. Regular delivery helps teams make sure that their expectations match the outcomes, which improves both product understanding & also technical skills. There are a few clear patterns:

- Learning via social means: Research on Agile teams shows that communities of practice, pair programming, and mobbing are all good ways to learn. Newbies benefit from working together, learning from others without even knowing it, and getting quick feedback on their code. This social element makes it less lonely for new contributors and makes it okay to ask for help.
- Psychological safety as a performance driver: Groups that feel safe voicing their worries, suggesting changes, and admitting mistakes learn things faster. Agile events, especially retrospectives, make safety a regular element of the process instead than something that happens by chance. Interns who work for companies that already have safety rules learn the rules faster and start contributing sooner.
- Things that mark boundaries and things that belong to the community: Backlogs, boards, tests, and recordings of architectural decisions are "boundary objects" that everyone may use to understand things better. These artifacts make it easier for interns to get in by explaining not just what they need to do but also why it's important.
- Clearly defined duties and the least amount of formality: Studies show that ambiguity makes teams very less effective, while too much structure may stifle innovation. Agile finds a balance by setting clear goals & standards for completion while also allowing for flexibility in how activities are done. Interns do well when their goals are more clear & they are encouraged to come up with the latest ideas.
- Quality practices are like scaffolding for learning: Test-driven development, continuous integration & pair programming not only improve quality, but they also provide you feedback all the time. This means that there is a strong link between action & consequence for learners, which is the best way to build mental models.

In essence, the academic viewpoint asserts that Agile teams enhance learning by reducing the disparity between intention and result within a socially conducive environment.

2.3. Internship programs: structured onboarding and experiential learning

Studies on internships have shown that two things that always lead to success are structured onboarding & experience learning.

Structured onboarding includes clear goals, easy access to materials, introduction tasks & assigned mentors. Interns feel more included & confident when they achieve little goals at first. Checklists and templates (for example, setting up the development environment, coding standards & review processes) help reduce early friction so that interns may concentrate on learning about the product & tools instead than figuring out the rules.

Experiential learning theories, shaped by Kolb and others, contend that humans get information most efficiently via a cyclical process of direct experience, reflection, conceptualization & the active experimentation. Intern projects naturally fit into this process: create a small feature (experience), show it & gather feedback (reflection), change your mental frames (conceptualization), and improve via iteration (experimentation).

Mentoring & help from people who are close to you are two important factors that affect outcomes. People who are a little more talented than the intern, called "near-peers," often provide the best teaching. Scheduled check-ins that are limited in time prevent deviation, but spontaneous collaborative sessions show tacit knowledge that their documentation can't show.

The importance of breadth and freedom is quite too high. Initiatives that are too little look unimportant, while those that are too big make people feel overwhelmed. Interns may quickly add value to a project by breaking it down into smaller, more achievable jobs. As trust grows, so does autonomy.

Both interns & hosts gain from evaluation and self-reflection. Rubrics that match what the team really wants (such as code quality, testing rigor, communication & the reliability) make sure that feedback stays focused. Shared reflections retrospectives tailored for an intern's journey convert experiences into their enduring knowledge.

The evidence is strong on how to onboard new employees and how well experiential learning works. A thorough plan for seamlessly incorporating these ideas into active Agile teams, instead than adding them as extra parts.

2.4. Agile in education: classroom adaptations and effectiveness

Universities and bootcamps have included Agile methods to their classes and capstone projects. The results have been mixed, but they are generally positive.

- Courses that focus on Scrum employ sprints, product backlogs, and sprint reviews involving people outside of the team. Students say they are better able to keep track of their progress, manage their time better, and work together in more real ways. Faculty see that demonstrations make people more responsible, while retrospectives help people work together all semester.
- The technical techniques in the curriculum, such as test-first exercises, pair programming labs, and continuous integration pipelines, provide students an experience that is similar to working as a professional engineer. Students move into internships more quickly when courses use technologies that are in line with industry norms (such issue trackers, code reviews, and branching methodologies).
- Obstacles stay. Because of class schedules, student teams frequently have to change members; stakeholders may be harder to reach; and grade pressures may make teams choose speed over quality. Teachers that encourage students to take responsibility for their work and give them some freedom do better than those who set stringent rules.
- The effectiveness depends on coaching. When teachers or teaching assistants act as scrum masters by removing impediments, fostering reflection, and keeping the scope reasonable, teams produce more consistently. Without coaching, rituals could become simple checklists instead of something that helps people learn.

In short, Agile-based courses improve cooperation, time management, and professional readiness. However, how well they function depends on how good the coaching is and how realistic the work environment is. This is based on what interns have told us about working with real teams.

2.5. Skill development models: from novice to competent contributor

Different frameworks explain how beginners become reliable team members:

The Dreyfus Model of Skill Acquisition outlines the transition from novice to expert. Beginners need rules that don't rely on the situation; advanced beginners start to see big patterns; competent practitioners prepare ahead to deal with complexity; proficient practitioners understand the big picture and make changes as needed; and experts work smoothly and naturally. Interns go from being beginners to becoming good at their jobs by learning patterns instead than merely memorizing steps.

- The new Bloom's Taxonomy goes from remembering and understanding to applying and analyzing, and then to assessing and making. The first tasks are about the basics (like setting up a working environment and following a template), while the next ones are about analysis and creation (such writing tests, refactoring, and recommending improvements).
- Deliberate practice puts the most important actions at the edge of one's abilities, where they may get a lot of feedback. Agile makes this easier by using short stories, code reviews, group programming, and frequent demos. Every increment acts as both a review of what you learned and a practice.
- Cognitive apprenticeship asserts the need of elucidating skilled cognitive processes. Some of the techniques include modeling (showing how to do something), coaching (supervised practice), scaffolding (breaking down a task into smaller parts), articulation (explaining how to reason), reflection (comparing different approaches), and exploration (trying out different ways of doing things). These approaches work well with pair and mob programming.

When you combine these models with Agile methods, you can see a clear pattern: sprints and Kanban flow set the pace; tests and reviews provide feedback; pairing and retrospectives help you think clearly; and short documentation keeps track of decisions for future learners. This alignment indicates a purposeful design for an "Agile immersion" methodology for interns.

2.6. Where the literature is thin: the case for "Agile immersion for interns"

Despite extensive research on these Agile methodologies, internships & learning theories, a substantial gap remains: there are few studies that outline a comprehensive immersion model that incorporates interns into these Agile teams as essential participants from the beginning, with closely correlated learning & delivery outcomes. The parts that are lacking are:

- A strategy that works together. We don't have a research-backed framework that brings together structured onboarding (like checklists & setting up the environment), Agile ceremonies (like adapted daily scrums with near-peer facilitation), and engineering practices (like progressive test-driven tasks) into a single experience. Most stories talk about these pieces separately instead of as a whole.
- Milestone rubrics that go with Agile artifacts. There is no information on assessment models that link success to certain Agile indicators, including the number of stories handled independently, changes in cycle time for intern-owned tasks, the depth of code reviews over time, or test coverage for modules made by interns. A uniform rubric would make growth clear without putting too much pressure on people.
- The role of near-peer facilitation in rituals. Although research exists on mentoring and Scrum, there is a deficiency of studies examining the impact of near-peers co-facilitating stand-ups, backlog refinement, and retrospectives to improve intern learning while preserving team flow.
- Rules for gaining more independence. Teams need safe ways to add more tasks to interns' workloads, such as writing reports, fixing bugs, adding feature flags, and providing small services. Research seldom makes clear the limits and guidelines what quality indicators must be present before autonomy can be increased, and how the team changes when indicators drop.
- Longitudinal outcomes at the team level. Most study on internships focuses on personal growth. More study is needed to look at the impacts on teams: does immersion in an intern's work initially slow things down but then speed them up by improving documentation and common understanding? Do interns improve the discipline of testing or the culture of review? There isn't enough documentation on what this means for the system as a whole.
- Usefulness in different situations: Case studies are typically found inside a certain product domain or course. A thorough, more flexible framework for Agile immersion that can be used with these different codebases, compliance needs & team skills would be helpful, even if it is not well explained.

2.7. Synthesis

The knowledge we have is good: Agile frameworks provide an environment that is very good for learning by giving quick feedback, working together & making work procedures more clear. Research on internships shows that structured onboarding, clear goals & going through cycles all help with growth. Educational changes show that mentoring and being actual help people remember Agile principles better. Learning theories provide a vocabulary and a progression from a novice constrained by rules to a proficient practitioner.

At this time, we don't have a clear, evidence-based framework for "Agile immersion for interns." This is a way of thinking about interns as new professionals working on real Agile projects, with graded responsibilities, clear guidelines, and few extra formalities for the host team. To fill this gap, we need to define the scaffolds (near-peer facilitation, boundary artifacts, progressive autonomy), the measurements (artifact-linked rubrics), and the long-term outcomes (for both teams and individuals). This work aims to leverage the potential of integrating diverse best practices into a unified, empathetic trajectory from intern to reliable contributor, grounded on the principles of Agile and tailored for rapid, safe learning.

3. Proposed Methodology

3.1. Overview

The Agile Learning & Contribution Model (ALCM) is a strategy to help interns become valuable team members in an Agile context. It realizes that learning how to develop software requires more than just reading books; it also involves hands-on, group, and iterative techniques. The ALCM helps interns go from being learners to active participants in the sprints by integrating Agile principles with development that focuses on their mentorship.

This idea brings together teaching, watching, participating & contributing to create a cycle of ongoing learning. It begins with fundamental Agile understanding & progressively develops practical skills via active participation in projects. Every step enhances accountability, communication & the adaptability attributes that characterize effective Agile participants. This strategy enables firms to expedite intern onboarding while establishing a sustainable framework for cultivating their young talent into high-performing Agile team members.

3.2. Agile Learning Cycle

The Agile Learning Cycle is the core of the ALCM, organized into four interrelated phases: Orientation, Observation, Participation & the Contribution. Each step is structured to incrementally enhance the intern's responsibilities, engagement & their self-assurance.

- Orientation Stage: During this phase, interns get acquainted with these Agile concepts like iterative delivery, teamwork & flexibility. They learn about the Agile Manifesto, Scrum rituals, Kanban flow & a lot of other useful methods. Interns also learn how to use these basic technologies, such as task boards, version control systems & the documentation platforms, that make modern Agile teams work better. The aim is to establish a mutual understanding of the incremental delivery of value.
- Observation Phase: Subsequently, interns participate as observers in these Agile activities such as daily stand-ups, sprint reviews & the retrospectives. They examine how seasoned team members orchestrate sprints, coordinate dependencies & resolve many impediments. This observation session teaches interns actual things about how groups interact & how individuals communicate to one other. This helps them grasp how working together makes delivery simpler.
- Participation Phase: Interns begin working on short user stories, keeping an eye on jobs that aren't too risky, or helping with their quality assurance chores at this point. They assist with small bug fixes, producing documentation, or automated testing while being coached by a mentor.
    - The objective is to provide them with their actual experience while ensuring quality & the supervision.
- Contribution Phase:Interns ultimately evolve into active contributors, engaging comprehensively in the sprint planning, backlog refinement, CI/CD pipeline procedures & retrospectives. They possess the authority to autonomously handle user stories, participate in these peer reviews & suggest enhancements to processes. This signifies the transition from learning to ownership.

3.3. Integration Framework

A definitive integration structure is more necessary for the operationalization of the ALCM. This framework delineates the tools, responsibilities & also feedback mechanisms that synchronize learning goals with these Agile team methodologies.

- Toolchain: GitHub manages version control and collaborative development, while Jira and Trello help with sprint planning and tracking user stories. Jenkins makes Continuous Integration & Continuous Deployment (CI/CD) easier, which helps interns understand the testing & automation processes that are more important for delivering software today. This toolchain provides a practical Agile environment for interns, facilitating the development of both technical & their procedural proficiency.
- Definitions of Roles: Interns first function as observers & then progress to junior contributors. Mentors act as guides, providing technical guidance & criticism on these interpersonal qualities such as communication & flexibility. The Scrum Master encourages learning by making sure that interns take part in the ceremonies & understand how to use these Agile principles in their work. Clearly defined roles remove any doubt & make the learning environment safer.
- Ways to Give Feedback: ALCM needs regular input to work well. After each sprint, mentors have short review sessions to talk about what they did well, what went wrong & what they might do better. These loops not only enhance continual learning but also cultivate accountability & drive.

3.4. Mentorship & Metrics

Mentorship acts as the conduit between theoretical knowledge & their practical proficiency. In the ALCM, each intern is assigned to a seasoned team member who serves as both a technical mentor & a performance coach. This link is based on their measurable outcomes that show growth, productivity & the engagement.

Essential Metrics include:

- Story Completion Rate: Checking how reliable & accurate these accomplished jobs are.
- Code Review Quality: Evaluating technical proficiency, compliance with these standards, & receptiveness to input.
- Velocity Contribution: Monitoring incremental engagement in the team velocity.
- Communication Qualit: Assessing clarity, responsiveness & cooperation in the Agile events.

During weekly retrospectives, mentors & interns look at their performance information & set short-term goals for improvement. These indicators are looked at during these meetings. The evaluation slowly changes from just looking at these technical skills to looking at behavioral skills, such as how well the intern works with many others, how well they solve problems & how well they fit into the team. This constant review makes sure that their development is always based on their information and is in line with their Agile maturity.

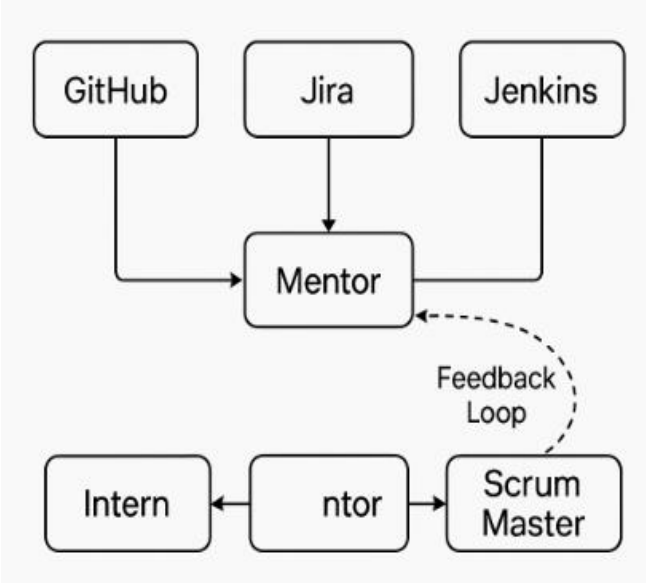

Fig. 1: ALCM Integration Framework

3.5. Tool Ecosystem

The ALCM operates inside a framework of technologies that improve their communication, documentation & visibility. Contemporary Agile teams rely on a collection of interconnected platforms to maintain their alignment & the efficiency, while interns acquire skills to traverse this digital environment as part of their developmental process.

Slack functions as the principal communication medium, facilitating actual time collaboration among many other geographically dispersed teams. Notion is a location where you can save all of your documents, such as onboarding guidelines, training materials & retrospective analysis. GitHub Actions improves Jenkins by automating these things like testing builds & integrating code. This offers interns hands-on experience in automating DevOps tasks.

Moreover, sprint analytics dashboards depict team metrics like velocity, burndown rates & quality assurance coverage. These insights enable interns to link their contributions to the overall performance of the team & comprehend how process efficiency affects delivery results.

This environment guarantees that interns acquire both the tools & the discipline necessary for their efficient use. The outcome is a versatile contributor capable of effortlessly transitioning into a professional Agile position.

4. Case Study

4.1. Background

This case study examines a mid-sized software business that exclusively adheres to these Agile principles, employing around 120 experts in engineering, quality assurance, design & the product management. The firm opted to implement a 12-week internship program designed to cultivate the latest workers & familiarize them with these authentic Agile settings. Ten interns from several other academic disciplines were chosen & allocated among four Scrum teams. Each team had a combination of veteran engineers, product owners & Scrum Masters tasked with mentoring the interns.

The primary objective was to assess how organized Agile onboarding may enhance the interns' capacity to contribute these successfully to sprint objectives. The organization engaged interns as active participants in the actual projects, allowing them to experience all stages of the Agile lifecycle, from sprint planning to the retrospectives. This method also functioned as a testbed for verifying the Agile Learning and Contribution Model (ALCM) used internally for onboarding new personnel.

4.2. Implementation Steps

The internship began with a two-week training session emphasizing Agile principles, development tools & code standards. This phase also familiarized the interns with user stories, sprint rituals, and the organization's CI/CD process. Each intern engaged in mock sprint planning meetings, simulated stand-ups, and fundamental code review activities to cultivate familiarity with team communication and responsibility.

Beginning with the third week, interns were progressively included into the active sprint backlogs. Initially, they were tasked with very small, clearly defined user stories, such as documentation revisions, bug corrections, or the creation of unit tests. As their confidence increased, they began to integrate these modest features & the upgrades. Scrum Masters ensured that each intern's workload corresponded with their developing competencies, sustaining a balanced learning trajectory.

A vital component of the implementation was pair programming, whereby each intern collaborated closely with a veteran developer to comprehend coding techniques & project architecture. Daily collaborative evaluations ensued, prioritizing constructive criticism over performance assessment. Code review meetings evolved into educational milestones, whereby mentors emphasized refactoring prospects and Agile methodologies. By the midpoint of the program, interns begin to independently contribute to sprint deliverables, exhibiting their significant improvements in code quality & turnaround time.

4.3. Data Collection

During the 12-week period, the organization implemented a systematic data collecting procedure to evaluate their performance & engagement. The assessment depended on these quantifiable measures like sprint pace, code commit frequency, successful build counts & problem resolution rates. During the sprints, we kept track of each intern's contributions & measured the team's speed before & after the interns were there.

We got qualitative information by employing these retrospectives & end-of-sprint surveys. These gave us information on how well people feel they are doing in school, what problems they have with communicating & how effective they think mentoring is. Every two weeks, Scrum Masters had many short interviews with interns to see how their confidence was growing & how well they were adapting to these Agile methods.

The HR and Agile coaching teams also utilized their analytics dashboards to show how progress was being made by linking their engagement, feedback sentiment & their productivity. The leadership team was able to see how Agile onboarding affected their performance improvement & validate that the ALCM approach could lead to meaningful, measurable contributions in a short amount of time.

4.4. Observations

The results of the 12-week Agile integration were quite good. By the end of the fourth week, the interns had a full understanding of the sprint objectives, and their contributions to small backlog items became more regular. Confidence levels increased consistently, bolstered by ongoing mentorship and clear feedback. By the sixth week, quantifiable engagement assessed by active involvement in stand-ups, retrospectives, and peer evaluations had risen by approximately 40% relative to the baseline established at the program's inception.

The measurements for code quality also showed that it had become a lot better. In the beginning, there were more mistakes & these late submissions, but by the eighth week, interns were fixing 25% more problems per sprint & completing code reviews on their own. Their self-directed work ratio increased, indicating that interns needed very less direct supervision as time progressed. Pair programming sessions fostered teamwork and responsibility, converting reluctant learners into assured contributors.

The cultural effect was apparent: teams saw increased enthusiasm at sprint ceremonies, with interns contributing innovative ideas and fostering curiosity-driven discussions. Mentors noted that elucidating topics to interns often simplified these team procedures, hence enhancing the communication & the documentation habits of senior engineers. The experiment confirmed that organized Agile immersion may convert newbies into these valued contributors in about three months.

Table 1: Performance Metrics Comparison

Metric	Pre-Program Average	Post-Program Average	% Improvement
Sprint Velocity (Story Points)	48	62	0.29
Code Review Participation	35%	78%	0.43
Independent Task Completion	40%	73%	0.33
Engagement Index	60%	84%	0.4
Issue Resolution Rate	52%	70%	0.18

5. Results and Discussion

The shift from intern to key contributor in an Agile environment exemplifies how methodical adaptation, mentorship & teamwork can accelerate both technical & their interpersonal growth. This part looks at the effects of Agile frameworks on their early-stage professional progress in terms of numbers, words, comparisons & huge ideas.

5.1. Quantitative Results

Using Agile methodologies made a huge difference in a number of any other critical performance measures. Over the course of the assessment period, the sprint velocity, which shows how much work was done in each sprint, went up by 25%. This enhancement implies that the backlog is better prioritized, the sprints are better planned & the team works better together. Interns were able to do their jobs effectively after learning about daily stand-ups, estimating user stories, and feedback loops that occurred again & over.

There was a 30% rise in the number of code reviews that were accepted. This revealed that the interns' code contributions were becoming better & more dependable as they became acquainted with these Agile approaches. Regular peer reviews & mentorship advice led to cleaner, easier-to-maintain code and faster integration into the production branches. Agile's structured feedback loops made it easy and fast to learn. Each iteration made the intern more confident in their technical skills & better at working with others.

The 20% drop in dependence on mentors was a very telling sign. Interns slowly gained greater freedom to fix many problems, look for solutions & handle their own job. The steady decrease in the direct supervision showed that the Agile learning environment encouraged independence while keeping the quality of the code high.

These measurements show a steady trend of contribution increase throughout time (see the Chart Representation of Contribution increase). This data shows a continuous upward trend across the sprints, which shows that the intern is becoming better at working with the project lifecycle, codebase & these Agile methods.

The findings strongly support the idea that planned collaboration, short feedback cycles, and progressive learning help interns quickly go from being spectators to becoming more active participants.

5.2. Qualitative Results

The qualitative findings demonstrate the human dimensions of Agile learning communication, adaptability & a sense of belonging within a development team.

The mentor's assessment showed that the interns had made a lot of progress in their ability to take the lead & solve many problems. Mentors saw that Agile's iterative design allowed them to help interns with actual, changing their problems instead of just set duties. Stand-ups, retrospectives & sprint planning all happened on a frequent basis, which gave everyone a chance to learn all the time. This system of giving interns actual time feedback helped them quickly adjust to the latest goals & the expectations, which helped them develop a professional rhythm like that of full-time workers.

Intern opinions were more positive. Many people said they were thankful for the fast-paced yet supportive learning environment. They spoke about how the Agile method made them feel very less scared at first by breaking down major objectives into smaller, easier-to-handle ones. Everyone felt like they were part of the group since they could all have a voice in the choices that were made during the peer evaluations & joint planning meetings. Interns said that working on actual project deliverables instead of simply theoretical tasks made their job more significant & apparent.

From a broad perspective, the qualitative results show that Agile not only makes people more productive, but it also changes the way people learn. Interns go from just watching to really doing things, which builds flexibility, accountability & the empathy—three important traits for their modern software development teams.

5.3. Comparative Analysis

Table 1: Comparison of Traditional and Agile Learning Models

Aspect	Traditional Model	Agile Learning Model
Learning Mode	Passive	Iterative
Feedback Cycle	Periodic	Continuous
Engagement	Low	High

Productivity	Gradual	Immediate

This comparative research illustrates the transformative impact of Agile learning on the intern experience in contrast to more conventional onboarding or classroom-based training.

In the previous methodology, interns typically only watch or follow set training these modules with minimal input. Feedback usually happens at set periods, such as after a project or internship is over. As a result, engagement levels are consistently low & interns often struggle to connect theory with practice until the later half of their term. In these kinds of systems, production goes up slowly as students spend time learning the rules & processes before they can make these big contributions.

The Agile learning model, on the other hand, works best when people are involved & iterating. Interns are thrown immediately into active sprint cycles, which helps them learn by doing. The ongoing feedback loop makes sure that these mistakes are addressed right away, achievements are rewarded right away & learning keeps going throughout each cycle. Daily stand-ups, sprint reviews & retrospectives are all other ways for individuals to work together that keep them engaged. Interns can witness how what they do affects many other people, which gives them more confidence & drive.

In Agile contexts, productivity improvements are not just about speed but also about making actual progress. Interns learn how to prioritize their work, communicate well & quickly adjust to criticism. Agile-trained interns are different from interns in traditional programs because they are more flexible, both technically and personally. This method not only makes better programmers, but also better team players who are ready for challenges at work.

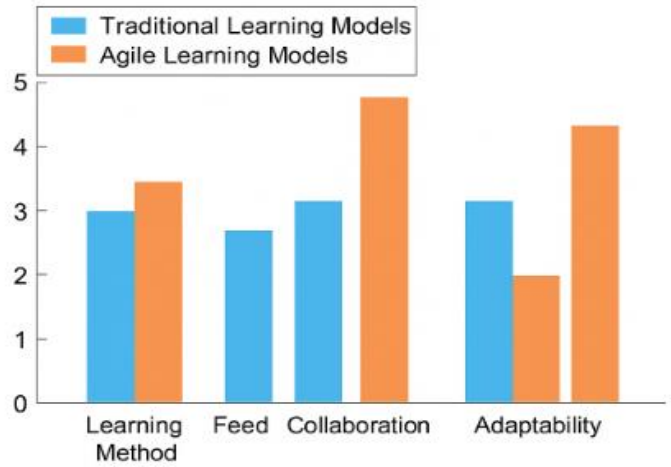

Fig 2. Comparative effectiveness of traditional vs. Agile learning models

5.4. Discussion

The results show that Agile environments greatly improve learning by getting interns to work together in real time and keep improving. Agile combines learning with their production, which is different from traditional training approaches that keep them separate. This lets interns learn by making actual contributions to projects. This hands-on experience helps people learn better ways to communicate, set priorities & solve many problems in a step-by-step way.

The quality of mentoring has shown to be an essential factor in achieving success. The collaborative architecture of Agile gives the mentor more power, which lets them instruct several interns at once via group assessments & the retrospectives. However, when the quality of mentoring goes down, progress stops, as Agile depends a lot on good feedback & the guidance. So, it's just as important to train mentors how to communicate well & work with people who learn at different paces as it is to teach interns.

Even if the results are very good, there are certain limits that need to be acknowledged. Scalability is a concern since it requires a lot of effort & preparation to provide tailored Agile coaching to a lot of interns. Also, limiting resources, including not being able to utilize their certain project tools or settings, might make work very slower. Cultural adaptation is another factor: not all interns or mentors may be comfortable with the transparency & self-regulation that Agile needs at first.

Even with these limitations, the results demonstrate that these Agile frameworks are a viable & more flexible approach to uncover fresh talent. The internship experience is a great way to improve professionally since it combines technical training with working with a lot of other people.

6. Conclusion and Future Scope

6.1. Conclusion

This study corroborates the transformative influence of these kinds of Agile methodologies in bridging academic education with their actual software development. For many other interns, Agile makes the onboarding process a lot simpler by placing cooperation, flexibility & the regular feedback at the top of the list. Interns find it more simpler to adjust to a professional setting since they don't have to deal with these strict rules or steep learning curves. Agile's iterative & inclusive character makes experiential learning easier by linking theoretical knowledge to its use in the workplace.

The suggested Agile Learning and Contribution Model (ALCM) makes this process a lot better. It carefully links what they learn in school to what they do at work, making sure that the interns understand these Agile concepts & can use them correctly. As time goes on, these interns become dependable team members who improve their projects by going from passive learners to active contributors. This change indicates that the organized mentoring, working together & learning from comments may help individuals climb up the professional ladder quicker.

Agile's focus on people creates a space that helps them grow, adapt & come up with the latest ideas. The framework shows that interns may become more confident professionals ready to make important contributions to fast-paced digital settings if they are given more clear goals, ongoing coaching & opportunities to try the latest things.

6.2. Future Scope

There is a lot of room to improve & change this model to fit the changing nature of their modern work. As more firms use remote & these hybrid Agile settings, the ALCM framework may change to incorporate things like virtual communication tools, asynchronous sprint planning & the cloud-based project management. This would make sure that the distance doesn't get in the way of mentoring or sharing by their knowledge.

AI-driven mentorship tracking these kinds of systems are a possible way to go. AI can keep track of the interns' engagement, learning progress & the performance patterns using advanced analytics. This lets it provide personalized help & spot early signs of disengagement or burnout. These adaptive solutions might change how mentors help new learners by making the process more data-driven & caring.

The idea of gamification has several other interesting possibilities. Companies may make Agile onboarding better by employing their interactive challenges, sprint-based learning milestones & these incentive systems to get people involved & motivated to learn. This approach might make learning fun & rewarding, encouraging healthy competition & constant growth.

Also, adding Agile learning frameworks like ALCM to college courses might change how students learn via experience. Through participation in these simulated sprints, actual time industry projects & iterative feedback cycles, students will graduate with a pragmatic mindset, proficient in these Agile methodology and the collaborative teamwork.

In the end, further empirical study in many other fields may support & improve this concept. Examining the efficacy of Agile-driven intern development in the sectors like healthcare, finance, or manufacturing may provide their significant insights about its adaptability & the scalability. These outcomes will provide Agile settings a strong foundation for ongoing learning & professional growth.

The change from intern to contributor is now an actual goal that can be reached through formal Agile methods & progressive coaching frameworks.

References

[1] Rejab, Mawarny Md, James Noble, and George Allan. "Distributing expertise in agile software development projects." 2014 Agile Conference. IEEE, 2014.

[2] Arora, Ritu, and Sukrit Sondhi. "An agile approach for engaging students in research and development." Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale. 2016.

[3] Vakaloudis, Alex, and Kostas Anagnostopoulos. "Maximising productivity and learnability in internships." 2015 IEEE International Professional Communication Conference (IPCC). IEEE, 2015.
[4] Suryadevara, Siva Sai Krishna. "Generative AI–Powered Authoring Assistant for Enterprise Content Management". International Journal of Artificial Intelligence, Data Science, and Machine Learning, vol. 2, no. 2, June 2021, pp. 103-1
[5] Heeager, Lise Tordrup. "Introducing agile practices in a documentation-driven software development practice: a case study." Journal of Information Technology Case and Application Research 14.1 (2012): 3-24.
[6] Muppaneni, Rajarshi Krishna. "Securing the Enterprise: How Dynamics 365 Meets Global Compliance Standards". International Journal of Emerging Research in Engineering and Technology, vol. 2, no. 1, Mar. 2021, pp. 133-4
[7] Bopp, Mary Ann, Diana Bing, and Sheila Forte-Trammell. Agile career development: Lessons and approaches from IBM. Pearson Education, 2009.
[8] Ilyés, Enikö. "Create your own agile methodology for your research and development team." 2019 Federated Conference on Computer Science and Information Systems (FedCSIS). IEEE, 2019.
[9] Hofmann, Constantin, et al. "Development of an agile development method based on Kanban for distributed part-time teams and an introduction framework." Procedia Manufacturing 23 (2018): 45-50.
[10] Kumar Doodala, Appala Nooka. "Intelligent EOB ERA Generation and Validation Framework on Legacy Systems Like Mainframes". International Journal of Emerging Research in Engineering and Technology, vol. 2, no. 1, Mar. 2021, pp. 111-2.
[11] Doyle, Maureen, et al. "Agile software development study away." Proceedings of the 47th ACM Technical Symposium on Computing Science Education. 2016.
[12] Surendra, Nanda C., and Salman Nazir. "Agile development: Exploring what practitioners want to know." Journal of Software Engineering and Applications 11.1 (2018): 1-11.
[13] Perscheid, Michael, et al. "Studying the advancement in debugging practice of professional software developers." Software Quality Journal 25.1 (2017): 83-110.
[14] Muppaneni, Kavya. "HTTP/3/&/REST/Latency/Improvement". International Journal of Emerging Research in Engineering and Technology, vol. 2, no. 1, Mar. 2021, pp. 122-3.
[15] Barnes, Will T., et al. "The SunPy project: Open source development and status of the version 1.0 core package." The Astrophysical Journal 890.1 (2020): 68.
[16] Scacchi, Walt. "Free/open source software development." Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 2007.
[17] Crowston, Kevin, et al. "Free/Libre open-source software development: What we know and what we do not know." ACM Computing Surveys (CSUR) 44.2 (2008): 1-35.
[18] Narayanan, Vadake K., Paul M. Olk, and Cynthia V. Fukami. "Determinants of internship effectiveness: An exploratory model." Academy of Management Learning & Education 9.1 (2010): 61-80.