Information Technology and Management (2022) 23:193–211
https://doi.org/10.1007/s10799-022-00370-y

Adapting agile development practices for hyper-agile environments: lessons learned from a COVID-19 emergency response research project

Salman Nazir[1] · Brad Price[1] · Nanda C. Surendra[1] · Katherine Kopp[2]

Accepted: 21 June 2022 / Published online: 30 July 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
Agile development is known for efficient software development practices that enable teams to quickly develop software to cope with changing requirements. Although there is evidence that agile practices are helpful in such environments, the literature does not inform us as to whether agile practices can also be beneficial in hyper-agile environments. Such environments are characterized by an extremely fast pace of change with fluid requirements. COVID-19 vaccine distribution is one such problem that governments have had to deal with. To solve this problem, governments need to come up with robust responses by formulating teams that have the capability to provide software solutions enabling information visibility into the vaccine distribution process. Such emergent teams need to quickly understand the distribution process, oftentimes define the process itself because it might be non-existent, and build software systems to solve the problem in a matter of days. Not much is known about how systems can be developed at such a fast pace. We adopt a clinical research methodology and employ agile software development practices to develop such a mission-critical system. In the process of building the system, we learn important lessons that can be used to adapt and extend agile methodologies to be used in hyper-agile development environments. We offer these lessons as important first steps to understanding the best practices needed to develop software systems that have the capability to provide visibility into the unprecedented health challenge of distribution of life-saving COVID-19 vaccine.

Keywords Hyper-agile · Agile · COVID-19 · Software development

1 Introduction

Agile development practices continue to grow in popularity as organizations adopt them to meet changing software requirements. Agile practices are known to enable short iterations, constant communication with customers and flexibility to adapt to changing requirements [1]. Even though agile practices espouse change in requirements and are valued by organizations for their superior speed of delivery and flexibility relative to alternative methodologies [2, 3], there is no research that informs us as to whether agile practices can be employed in highly complex, hyper-agile, mission-critical environments. Hyper-agile environments can be defined as contexts having extremely volatile requirements in which the inability to rapidly understand and adapt to the constantly evolving requirements can lead to severe consequences, including loss of lives. We know that traditional approaches do not lend themselves to be used in such environments, but we do not know whether agile development practices

Salman Nazir and Brad Price: Co-first authors.

✉ Salman Nazir
salman.nazir@mail.wvu.edu

Brad Price
brad.price@mail.wvu.edu

Nanda C. Surendra
nanda.surendra@mail.wvu.edu

Katherine Kopp
katherine.kopp@datarobot.com

1 John Chambers College of Business and Economics, West Virginia University, Morgantown, WV 26506-6025, USA

2 AI Academic Partnerships Director. DataRobot, Inc., Boston, MA, USA

can help us achieve the required speed and flexibility that is needed for hyper-agile environments. Without this knowledge, we would need to resort to expensive trial-and-error exercises that can translate to loss of lives. This research describes how a team of researchers adapted agile practices to develop a system tasked with managing the complex problem of COVID-19 vaccine distribution.

The Sars-CoV-2 virus (COVID-19) caught the world by surprise. As scientists scrambled to understand the virus and its behavior to come up with effective vaccines, and governments shut down schools and businesses, the virus claimed close to 4.6 million lives as of September 2021 [4]. Once vaccines were developed, it fell upon government agencies to come up with effective and efficient processes and supply chains to administer the vaccines in order to save precious lives. A major stumbling block in this effort was to build the capability to understand the information coming from multiple sources [5]. This information is vital and there is an urgent need to quickly develop processes that can be employed to absorb and understand this information on a real-time basis [6] to supply the right amount of vaccine to the right place at the right time. If vaccine is sent to the wrong location there is a chance that it may be wasted. At the same time, not providing vaccines where they are urgently needed is also an equally disastrous scenario. The key issue is gaining visibility of information to tackle the vaccine distribution and administration problem. With no processes in place and no previous experience with handling such a scenario, governments find themselves in a difficult situation. As is customary, emergent teams may be put together [7] and these teams might resort to solving the problem by attempting to develop a software solution that provides visibility into processes that are developed for vaccine distribution.

The systems development literature is silent as to how emergent teams that have no previous experience of working together can come together and develop software solutions that can handle such a crucial information visibility problem. Emergent teams are characterized by great urgency and fickle operating environments that are in a state of constant flux as information about needs and resources comes at a high pace and from a plethora of sources [7]. As team members are unaware of each other's capabilities, they also lack a shared understanding of the problem at hand. Coupled with this is the fact that the information coming from multiple sources might often be contradicting and might need to be sorted out before it becomes actionable. These teams also need to adopt a software development methodology that meets the need of the context and the emergent nature of the team. One possible route is adopting agile development methodologies that are suited for fast system development efforts [1, 2, 8–10]. However, agile methodologies have their own specific requirements. Although they are suited to fast paced development efforts, members of the team must gel together and develop communication routines that enable a shared understanding of the problem at hand [11, 12]. There is also a need to follow formal team organizational practices that may go against the fluid nature of emergent teams. There is no research that exemplifies and explains how agile methodologies can be effectively applied to such fast-paced scenarios where the delivery times are in days, rather than weeks or months, and errors can result in loss of human lives.

This research describes the efforts of a team of researchers that helped their U.S. State government in solving this problem by developing a software solution. The software solution is built to enable the distribution of the COVID-19 vaccine to meet changing demands and save lives. The team adopted an agile development methodology and adapted it to suit their high-pace, high-risk context. We expect to contribute to the agile development literature by showing how agile practices can be adapted to suit a hyper-agile context in order to fulfil user requirements that are changing at an extremely fast pace. We offer lessons learned as a first step to understanding the best practices needed to evolve agile practices as applied to emergent contexts characterized by unprecedented health challenges.

2 Literature review

There is limited research regarding the use of information systems as applied to crisis management. Most of this literature has studied systems that allow situational awareness [13]. Situational awareness is the understanding of environmental elements during crises to enable decision-making processes that help first responders [14]. Indeed, information systems are key to providing situational awareness as they provide timely, accurate and complete information regarding environmental conditions, responding participants, casualties and available resources [13]. The primary information system studied in prior literature is social media (e.g. Twitter, Facebook, etc.). Research has looked at how social media use can, in some instances, promote rumor mongering which decreases population's morale and increases anxiety, leading to worsening the impact of the crisis [15]. The effect of social media has also been investigated in providing connective affordances that allow a network of loosely connected users to reorganize themselves and cope with disaster [16–18]. Finally, the literature has looked at how coordinated action can be achieved by connecting like-minded individuals and improving their ability to self-organize by promoting situational awareness [13]. As noted, social media is the primary underlying system in all this literature. Clearly, vaccine distribution is a more complex phenomenon in which stakeholders have to look at demand side requirements and match them up with supply side flow while considering the constraints of time and availability and keeping a close eye on casualties. Such level of coordination is not found in any social media platform and there are no pre-existing systems that can be quickly employed to achieve this complex task. We also do not find any literature that suggests actionable best practices that guide the development of such an emergency response system *while the pandemic unfolds*. Since agile development methodology is the closest actionable framework that has the potential to deliver software in a short time frame, we based our development efforts on it and applied it to our emergent context. Next, we review the agile development and emergent teams' literature.

2.1 Agile development

Changing requirements pose a significant threat to software project success. In fact, a crucial challenge in responding to change is that what the software is required to do remains a moving target [19]. This incessant change in user requirements is perpetuated by continuously evolving business requirements [20, 21]. It is also increased by the frequency and speed of change in market competition that continuously require businesses to adapt their processes and products [22]. In response, organizations are increasingly adopting agile development [12]. Agile development is characterized by a flexible development process where short iterations and frequent communication among stakeholders enable the development team to respond more quickly and efficiently to changing customer requirements [23–25]. It is a lightweight and adaptable alternative to traditional, plan-driven software development methodologies [20]. Agile development values software developers as individuals and as important members of the team where interactions are considered the key elements in comparison to processes and tools. It views customers as integral members of the team rather than adversaries that cannot be trusted. The focus is on delivering value to customers by creating software that is usable rather than documentation that is cumbersome to create and maintain [26].

The team and its members have to collaborate within the software development subunit and with customers when adopting agile methodologies [27, 28]. This makes the team a crucial element of the process. Agile software development teams are essentially self-managing teams that organize themselves to assign tasks, schedule work and take action to solve problems [29, 30]. Research suggests that team members are able to effectively collaborate when they develop shared mental models [12]. The shared mental models allow team members to develop compatible expectations of the task and the team, enabling the team to understand current events, consider what may happen in the near future and understand why these events occur [30]. This level of understanding and knowledge coordination is not available in emergent teams responding to disaster scenarios.

2.2 Responding to disasters with emergent teams

The Sars-CoV-2 (COVID-19) pandemic is one scenario where there was an immediate need for emergent teams to form and spring into action to battle the pandemic. As thousands died, millions of people went into quarantine, and many businesses did not survive [31], scientists struggled to come up with vaccines, while governments scrambled to create effective and efficient supply chains to deliver those vaccines. In such disaster scenarios, there are often no established lines of communication, or pre-existing plans that can be incorporated to quickly respond to the impending needs. Information about the disaster or emergency comes from a plethora of sources and is often hard to reconcile in a timely manner because of the increased pace, complexity and level of detail [7]. Research has highlighted the importance of emergent response teams in such disaster environments (e.g. [32]). Such teams are markedly different from stable disaster response teams that are trained to respond to expected emergencies, such as police and firefighter teams [7]. Stable teams have experienced team members who are trained to encounter a multitude of potential scenarios and have experience in working with the same team members. Emergent response teams, on the other hand, have team members who are not familiar with each other's knowledge and capabilities. They may also lack shared goals, reward structure, and the time to share who knows what on the team [7]. All this unfamiliarity with the team and the context leads to difficulty in coordination of knowledge and expertise, resulting in poor response to the emergency.

Research has looked into coordination of expertise in different organizational contexts, such as coordination in software development [33] and coordination practices of medical trauma centers [34]. From software development research, we know that expertise coordination are processes that manage knowledge and skill interdependencies [33]. Expertise coordination processes help develop a common mental model that allows team members to understand the task at hand, know where expertise is located in the team, recognizing when and where it is needed and finally bringing it to bear [33]. From the coordination of medical trauma centers, we know that teams can incorporate dialogic coordination practices in addition to the expertise coordination practices to achieve coordination of expertise [34]. However, both of these contexts are characterized by teams where group memberships, tasks, roles and knowledge can be specified ex ante [7]. Emergent teams are unable to easily capitalize on these characteristics because the preconditions that facilitate coordination are almost non-existent. In emergency scenarios, such as the COVID-19 pandemic, processes are non-existent and data has to be gathered and reconciled from multiple sources. Although agile development practices are well known to deliver software at a faster pace, this pace is not fast enough to accommodate response times that span mere days with rapidly changing requirement and have life threatening consequences. The literature is silent as to how software systems can be developed at this hyper-agile pace. The problem is further exacerbated by the fact that the team most likely tasked with the development will be an emergent team where the members are unaware of each other's expertise, have no shared mental model and do not know how coordination can be achieved. Shared mental models are metastructures that establish the extent to which team members share the same understanding of the task, the tools, the team and the situation [35]. This shared understanding benefits the team members in better understanding and predicting the actions of other group members and enables better coordination of tasks [36]. This level of coordination develops over time as team members interact with each other on a regular basis [7]. Unfortunately, emergent teams do not come to the task with this level of shared understanding of the problem at hand. Such teams are at a disadvantage as there may be communications errors, and significant inefficiency in how knowledge about the task is shared and acted upon in such scenarios [37]. One interesting work that highlights the role of developing shared mental models in agile development teams is that of [12]. This research shows that some agile practices (system metaphor, stand-up meetings and on-site customer) help develop a shared understanding about the task to be completed, and also help establish shared mental models about team processes and team interactions [12]. Another study [38] looked at a team member's capability to integrate different disciplines and graft them into one (T-skill) and their capability to combine or integrate another member's skill onto one's own knowledge base (A-skill). The study reported that such skills help teammates coordinate their skills and enhance the development of shared mental models. However, the literature primarily does not inform us how software development can be achieved in emergent teams that do not have established mental models. Also, it is not known how agile practices can be extended to respond to such challenging scenarios where teams are emergent in nature.

In this study, we borrow from agile development literature and emergent teams literature to understand how teams that have no past experience in working together and have no shared mental models about the task at hand, move at a hyper-agile pace to produce a software solution that exceeds expectations in the coordination of activities related to COVID-19 vaccine distribution. With this study, we aspire to offer lessons that can help extend and evolve the agile development literature as applied to hyper-agile contexts.

3 Research methodology and context

We used a clinical research methodology for this study. Clinical research focuses on solving a client's real world problem and, in the process of solving the problem, helps advance research understanding [39, 40]. Clinical research is appropriate because it allows us to primarily focus on solving the problem of the client and the researcher plays the role of an interventionist who helps diagnose and solve the problem. When using clinical research methodology, the researcher plays the role of an "expert" or a "doctor" in a helping relationship. The researcher takes on the obligations to diagnose the problem and suggest and implement a solution, but the initiative to seek help remains with the client [39]. It is similar to action research as it administers interventions in organizations but differs in prioritizing the achievement of practical outcomes as its primary goal [41]. The client is involved in initiating and engaging the researcher and is involved throughout the intervention. There are several examples of using clinical research methodology in organizations. For instance, [42] used clinical inquiry to help improve the digital infrastructure in a large Swedish municipality. Another work used clinical inquiry to understand how digital strategy-as-practice emerges [43]. The study reported that strategy emerges in a rhizomatic order where offshoots de- and reterritorialize concepts, functions and logics in a continuous process. This process was facilitated through a combination of slack resources and adaptive governance [43]. Overall, clinical inquiry allows for engaged scholarship par excellence, where the aim is to help the client organization through sustained engagement in order to understand challenges from the client's perspective and suggesting interventions that help overcome these challenges [44].

3.1 Project background

In January 2021, we were contacted by the National Guard of a U.S. state to assist with COVID-19 vaccine distribution efforts. At the start of the conversation, the client contact did not even know the extent of their needs and was aware of this fact. He just knew that we had the skills to help get the right data to the right people in order to make strategic decisions. Working with the military on a COVID-response effort, we knew we had to be "hyper-agile" as changes were to happen frequently and often urgently. We also did not have the time to work through a typical software development lifecycle. We were going to have to build and test while the system was in production. As previous work with this client had taught us integration into analytics and analysis platforms was a key component and would be necessary to make the application useful. For this reason, we opted to use RShiny for our application(s). Shiny is an R package that allows users to create web-based applications for interactive data visualization [45].

3.2 Definitions

Operational week: The operational week for the project was decided to be from Friday to Thursday. Operational weeks are designed on a Friday to Thursday schedule, to coincide with vaccine arrival. Vaccine shipments will arrive beginning Friday afternoon. Also based on policies, vaccines can only be requested and distributed at different points during the week. Hubs are not open during the weekend, and pick-ups may not be scheduled for Friday as preparation is needed for the coming distribution days.

Distribution hubs: There are five distribution hubs in the state based on proximity to all counties in the state and ease of access via the interstate system. The hubs serve as a pick up location for various entities in the state. Each hub serves a different area, and is based on location to different areas of the state, serving a different population size and number of counties. The activities performed at each hub are the same and are to intake, store, and distribute the COVID vaccine to the counties. The hubs serve as a room where a vaccine is stored using refrigeration capacity necessary for the vaccines to be distributed.

Unallocated vaccine (UAV): Excess inventory resulting from vaccine administrators picking up fewer vials than what was requested or allotted for the week and/or planned excess inventory for upcoming mass vaccination efforts.

Joint interagency task force (JIATF): A concept typically used in military constructs that requires close working relationships between two or more organizations or agencies. Allows the ability to utilize assets from all agencies involved and is typically focused on a single goal or mission, under a single director.

Customers: Customers were anyone who was approved to administer vaccines. So think of this as your healthcare providers, hospitals, pharmacies, health departments, etc.

Users: Users were the users of the software such as the JIATF stakeholders associated with compiling orders, and the members of the State National Guard who interacted with the system as the key users in vaccine distribution.

3.3 Process details

The following timeline occurs weekly.

Friday: UAV is rolled over from previous operational week. Vaccine and related supplies are received by each hub from the CDC shipments.

Monday: Additional CDC shipments of vaccine and related supplies are received and customer pick-ups begin including interhub transfers.

Tuesday: Notification of CDC supply quantities for next operational week and customer pick-ups at each of the 5 Hubs.

Wednesday: Customer pick-ups, vaccine requests submitted and distribution plan prepared for next operational week.

Thursday: New work orders and change orders reconciled before next operational week begins. Order placed with CDC for next operational week.

Ongoing: Customer pick-ups with the majority happening on Tuesdays and Wednesdays; inter-hub transfers of vaccine inventory. Reconciliation processes (Fig. 1).

Previously the distribution process had been extremely manual. On Wednesdays customers would send spreadsheets with the demand of vaccines for their customers in a non-standard spreadsheet by UAV on Tuesday. There could be 8–10 spreadsheets per week accounting for over 200+ orders from different entities throughout the state. A distribution plan was then created which entailed assigning each request a work-order, a pick-up day, and pick-up time. From there, state leadership would reconcile each demand based on supply levels from the CDC, and assign an order quantity to each order. If supply exceeded or met demand, each order was filled at the requested levels. Once the orders were allocated, the operational team would then reconcile the number of vials of each brand of vaccine that would need shipped to each hub. As vials can only be shipped in boxes or trays (i.e. partial boxes cannot be shipped), the operational team would then decide what hubs the full allocations would go to, and then schedule inter-hub transfers. An assessment would also be made on secondary supplies (e.g. syringes, needles, gloves), as certain tool combinations maximize the return on vaccine distribution. For instance, a certain needle syringe combination is known to be able to get more vaccine doses out of a single vial. Finally, the distribution plan is finalized and sent to all stakeholders (customer entities, customers, hubs, and leadership). Throughout the week new work orders would be added as entities are unable to pick-up vaccines, or need to change the amount of requested vials leading to a dynamic distribution.

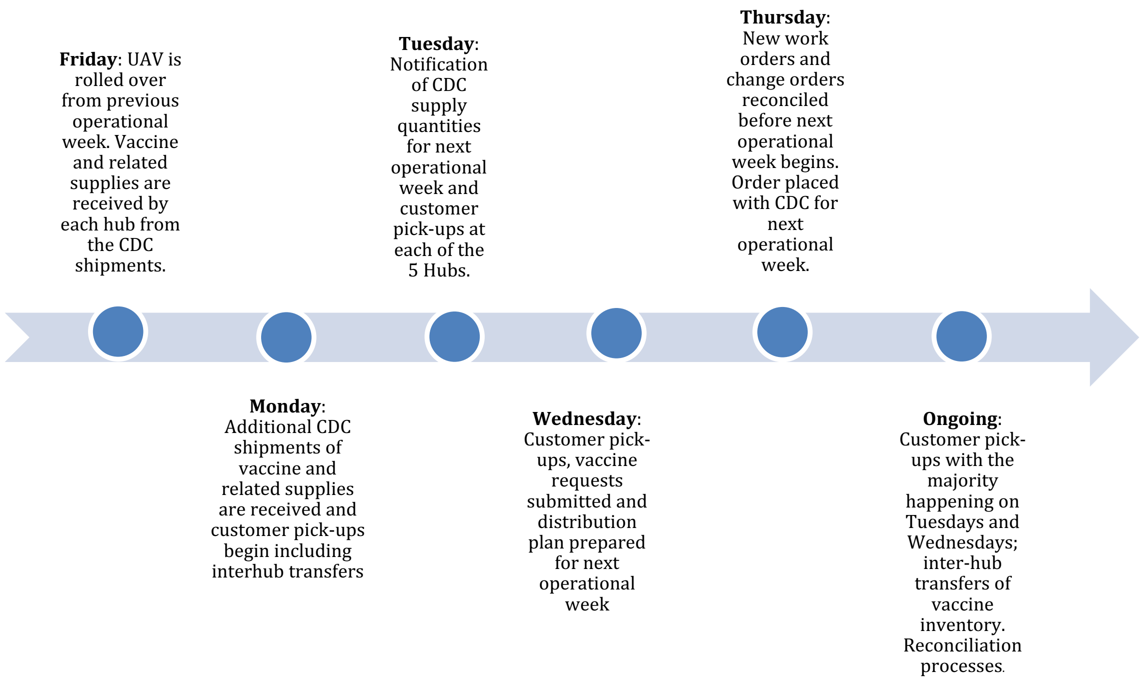
Friday: UAV is rolled over from previous operational week. Vaccine and related supplies are received by each hub from the CDC shipments.
Monday: Additional CDC shipments of vaccine and related supplies are received and customer pick-ups begin including interhub transfers
Tuesday: Notification of CDC supply quantities for next operational week and customer pick-ups at each of the 5 Hubs.
Wednesday: Customer pick-ups, vaccine requests submitted and distribution plan prepared for next operational week
Thursday: New work orders and change orders reconciled before next operational week begins. Order placed with CDC for next operational week.
Ongoing: Customer pick-ups with the majority happening on Tuesdays and Wednesdays; inter-hub transfers of vaccine inventory. Reconciliation processes.

Fig. 1 Process timeline

Before the research team was engaged, this process was completely manual and the creation of a distribution plan could take 72–96 h. The process was error prone as it required a single individual to copy, paste, and develop all content necessary. Furthermore, during distribution files changing due to the dynamic nature of changing orders created havoc between hubs and leadership as the common operating picture was muddied. After the research team was involved, the distribution plan took less than 4 h to be finalized, and a common operating picture was developed to share information and real time insights between the hubs and the operations leadership.

We chose the SCRUM framework to guide the overall process and aspired to use all SCRUM ceremonies. However, using the agile development methodology during the pandemic was a challenge. As the team had to work remotely, it wasn't a regular standup meeting, but check-ins were multiple times a day and very unscheduled as the requirements changed multiple times a day based on situation dynamics. The team would meet as requirements changed, but did meet at least once a day to continue inspecting progress towards project goals of the iteration. During these meetings, the team would discuss any problems or identify any obstacles to task progression. These meetings were also a good way to discuss any updates to the project backlog that would be needed to match the changing requirements.

The team was not able to maintain any scrum boards for items and tasks. Since things changed so rapidly that updating the board would have been a job in itself. One of the national guard members used a task board to keep features he would like added on there, but it wasn't directing the development team's processes. It was more of a wishlist that he maintained because of all the dynamics on the ground.

Overall, the team had planning on specific goals in each iteration which helped us define the project goals that would be achieved by each app. We also defined the time period this app will need to be completed in. Sprint reviews were done when we hit milestones with stakeholders, and trainings with them. The team did these at every major update or introduction of an application. The team did not have time to do the retrospectives. With the continuous changes dictated by the emergent nature of the context, the goal was simply to meet the next urgent requirement.

3.4 Data collection

In line with interpretive research, we used several data collection techniques [40]. The data was collected mainly through scheduled open-ended interviews and discussions with users and stakeholders. We also conducted several unscheduled discussions with stakeholders, which usually occurred when a new or revised functionality was needed or when users reported problems with the system. Moreover, we observed the users in their workplace setting by visiting

Table 1 Catalogue of time spent in various data collection activities

Type of data collection	Who was involved?	How long and how many times?
Scheduled interviews and discussions (scheduled for 30 min to 2 h)	The JIATF Liaisons, HUB Staff, Technical team liaison, Customer Liaisons	12 h (10 meetings)
Unscheduled interviews and discussions	JIATF Liaison, HUB Staff, Technical Team Liaisons, Customer Liaisons	100 h (80 meetings)
Observation	JIATF Floor, Technical Liaisons, Technical Team, Operations Team, HUB Workers	16 h (3 times)
Reading/reviewing documents	Documents explaining vaccine efficacy, procedures for giving a shot, necessary equipment. Other documents such as working documents used prior to research teams engagement to understand the process of distribution. Obtaining information that was not necessarily tracked in systems or formally prior to research team engagement	20 h

Table 2 Major system parameters

Parameter	Number
Number of functionalities	
Data processing functions	6
Data update/augmentation	9
Data visualization/analysis	7
Total number of applications	7
Approximate number of lines of code	5000
Approximate number of system development hours	500

the distribution hubs to gain a better understanding of the scale and complexity of the requirements. In addition, we employed participant-observation and studied any documents used by users for performing their work and created as a result of their work. Data collection for this system was chaotic at best. Many conversations happened in an ad-hoc manner, as the dynamic nature of the project did not facilitate the time to collect all necessary implementations. Table 1 catalogues the time spent on each type of data collection and Table 2 shows major parameters of the system.

4 Research analysis

We used an agile iterative approach and went through three iterations during this research project. For each iteration, we first describe the overall business need as understood by the researchers at the beginning of the iteration. Next, we describe the solution created by the development team. Finally, we describe the lessons learned during the iteration.

4.1 First research iteration

The entities involved in the vaccine delivery effort consisted of groups such as Hospitals, Local Health Departments, Higher Education, PK-12, etc. In the state's Joint Interagency Task Force (JIATF), one member was assigned as the liaison for each entity. This liaison would submit an Excel spreadsheet once weekly with details regarding customer details, point of contact details, vials requested including vaccine manufacturer and series [prime (first dose) or booster (second dose)]. These spreadsheets would be manually aggregated by another member of the JIATF and re-sorted to prepare the distribution plan for each distribution hub. Due to the manual nature of the process, it was highly susceptible to human error. The process was inefficient and would often take up to four days to complete.

To address immediate needs, we needed to help automate the process of several entities submitting vaccine requests. In turn, this would automate the creation of the distribution plan, increasing the time to plan execution of vaccine delivery. Within less than a week of our initial conversation with the State's National Guard, we had two applications in operation. The first was an interface to allow the entity liaisons a place to upload their weekly spreadsheet. We minimized the information they had to provide and validated their submission before accepting the upload. In a separate application, we allowed the management team the ability to download two reports. The first was an overall vaccine request that summarized the requests by hub, vaccine brand and series. The second was the foundation of the distribution plan. This document is what the hubs use to facilitate the distributions directly to the medical providers who give the shots. We automated the creation of new fields and created a workbook tab for each distribution hub. There was still some manual work to be completed as there were no business rules in place for some of the work, but in the first

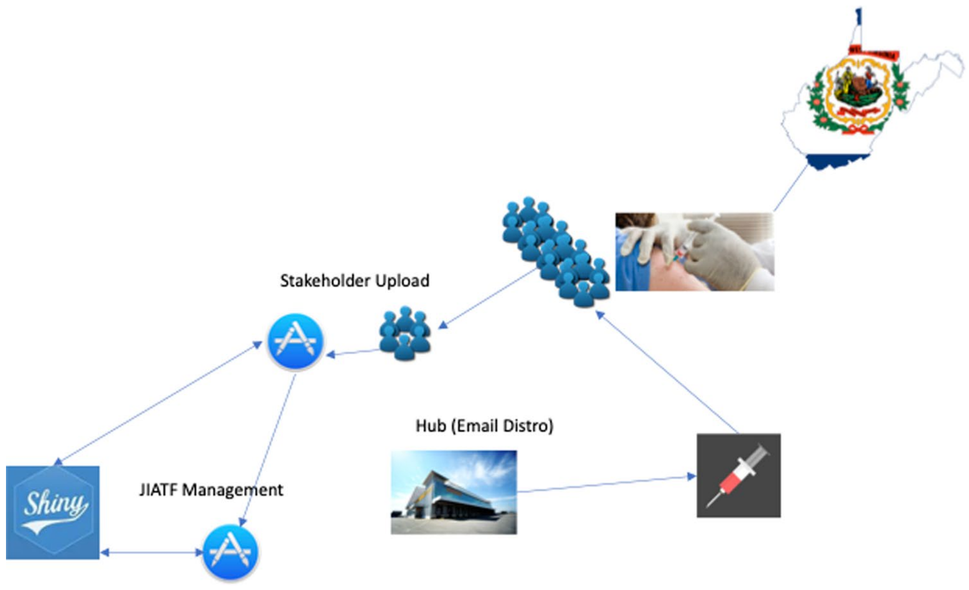

Fig. 2 Diagram of research iteration 1, consisting of two applications to process vaccine demand from stakeholders, while distribution of vaccine was based on the provided distribution plan with no real time visibility. All demand was collected by stakeholders and uploaded to the system for internal leadership review

week we cut a four day process down to a four hour process and minimized human error.

At this point in the process, communication was done predominantly via email with some phone calls and Zoom sessions when more complex questions or answers needed to be discussed. All code version control and collaboration was managed through GitHub. In the first week, it was all hands on deck from early in the morning through the late hours of the night. Necessary data collection was done via e-mail and discussion with our State National Guard liaison and stakeholder who received feedback from the entity liaisons. Furthermore, the excel spreadsheet used for upload was not standard so entity liaisons needed to be trained on the system as we were developing it. The diagram of applications and interactions is shown in Fig. 2. It shows that the first iteration's focus was on developing a better understanding of weekly vaccine demand.

4.2 Lessons learned during the first iteration

(1) The best agile practices in an emergency are the practices that quickly adapt to changing user requirements to deliver a usable product. We focused on delivering a minimum viable product to meet the immediate needs and then continued building from there. The data came from a plethora of sources and was hard to reconcile due to the pace and complexity of the data [7]. With this amount of data, it is easy to inflict information overload on the decision makers by trying to provide all possible reports that could be generated. With a targeted focus on delivering the minimal viable product, we were able to maintain just enough visibility for all stakeholders without overloading them with information that was not useful for their decision-making needs.

(2) As this emergency had the potential to impact millions of lives across the state, each vaccine dose was extremely precious. There were risks of not meeting demand due to misallocated doses, under-supplying the need, or simply not allocating the federally allotted doses. We were aware that we will not have the luxury of using the agile development recommendation of iterations that span one or two weeks. We adapted each agile iteration such that it should be done as fast as needed without adhering to the one or two week iteration recommendations. The iteration was made to be as short as possible to release a minimum viable functionality. In many cases, there were two releases per day for meeting evolving user requirements. This allowed us to move at a hyper-agile pace.

(3) Use the lightest weight tools and software possible to build the required functionalities. Many of the lightest weight tools tend to be open source tools—such as R and Shiny, and Shiny Server Studio—that were used in developing the applications. Not only are the lightweight tools easy to use but there are often open source communities that respond in an agile manner to add or extend packages that might help developers in an emergency. RStudio actually donated a piece of their non-open source system called RStudio Connect that allows for some of the additional security features we wanted beyond what we have now. The part of this too is that visualization and analysis are the key components to building in a language built to analyze data and make life easier. Also, the functional programming nature of the development language made life a bit easier, as we

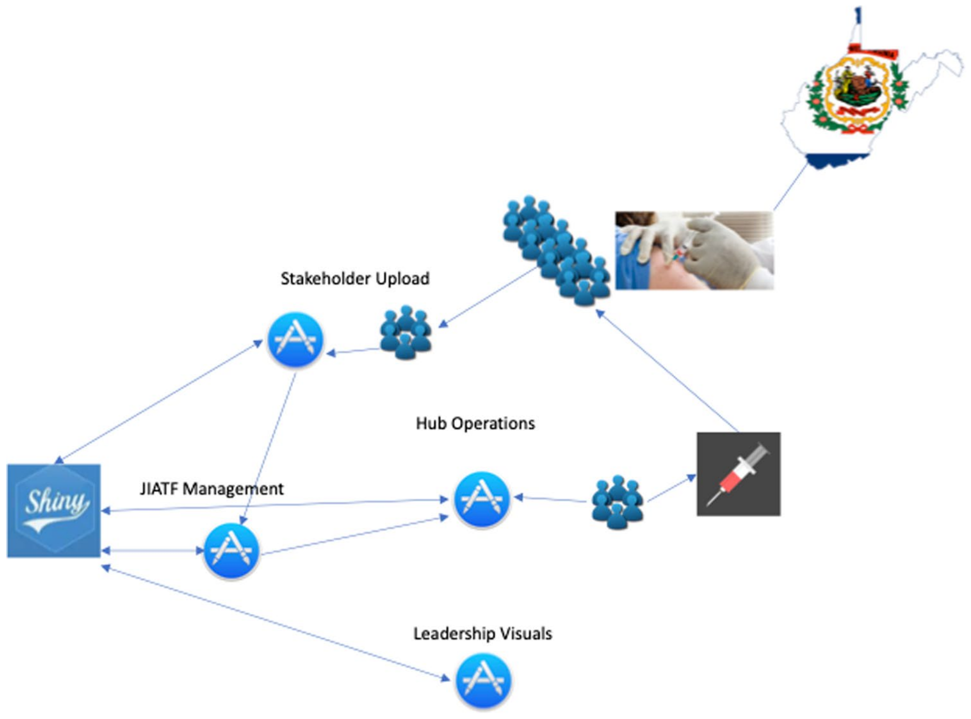

Fig. 3 Research Iteration 2, which adds real time visibility into hub operations and distribution of vaccine, while simultaneously assessing upcoming vaccine demand. The stakeholders are still responsible for acquiring demand information in this iteration, but real time changes to distribution plan can be made, and real-time insights into distribution can be given so that decision making can be dynamic in nature

were able to use pre-built components to develop the application because minimum viability and functionality, not "style", were the key.

4.3 Second research iteration

The second iteration of this work, diagrammed in Fig. 3, focused on providing visibility into the actual distribution process from the hubs. This required understanding challenges the hubs faced with distribution such as scheduling, and other state and federal data entry requirements on distribution. The unseen issues to our systems were the existence of systems such as the federal vaccine management system (VAMS), and state level immunization tracking systems that required information from users. As federal systems focused more on the federal pharmacy program (of which our state was the only one to originally opt out of) and no API's to immunization records systems had ever been needed, connecting to traditional medical systems was not possible. Thus, minimum standards were of major concern. The necessary requirements were to be able to track distribution, but also produce real time insights into potential issues with distribution. Three applications were designed to produce insights on behaviors of customers such as late pickups, partial pickups, or errors in the distribution files. Partial pickups occur when an incorrect amount of vaccine is ordered, or the provider is unable to pick up the full order due to capacity issues.

Within less than 9 days of the request being made three new applications were online, with appropriate considerations and adjustments. The initial production applications were done with feedback only from JIATF leadership and stakeholder liaisons, because of the timeline for the application development. A week after the product was live, the developers spent four hours at a Hub (the first exposure to actually viewing a Hub and the process performed rather than hearing second hand). This first-hand experience allowed for a simplification of the distribution hub application within hours of leaving the Hub and was in full production by the following morning. Furthermore, adjustments of visualizations had to be made to meet military guidance on visualizations such as the ability to take static quad charts and turn them into dynamic dashboards, causing additional tools to be made available to meet reporting needs.

This implementation had complexities that needed to be addressed as the three applications had many moving parts of data. The first application was designed for hubs to be able to indicate the current state of orders on the distribution report. It allowed them to confirm pickups and deliveries, which allowed a real-time account of demand for vaccine and stock levels of all vaccine types. The second application was to allow changes in the distribution report by the JIATF to account for the dynamic environment of vaccinations. For instance, if an order needed to be added, this application gave the JIATF the ability to add the order without calling the hub and it would automatically be reflected on the distribution report for the hub. Finally, the third application provided the ability to visualize the distribution and stock of vaccines. Changes in the distribution process and stocks can be monitored by leadership to make necessary decisions such as creation of vaccine clinics and where to route vaccines that had not been distributed from mobile clinics. In cases such as this, data persistence must always be a concern, the developers considered this when doing the storage, and created a methodology for allowing users to interact with the most up to date distribution plan while minimizing data persistence without the use of a production level database.

4.4 Lessons learned during the second iteration

(1) Divide and conquer: We often found it hard to encompass the set of requirements if we considered a complete application for the entire vaccine distribution system. Instead, we found it expedient to think of major functionalities as applications on their own. We decided to create multiple applications rather than one large application with several modules. This mechanism enabled us to achieve quick delivery of functionalities. With this approach, developers only needed to worry about minimal coordination among applications.

(2) Practice a minimalistic approach when testing: As time was limited, we often needed to develop and test the system as it was in production. This meant that unit tests and stress tests that are key components of agile development were not adhered to. Since we had created multiple applications, when a bug needed to be fixed or an enhancement needed to be implemented, it only impacted one application at a time and subsequently fewer users at a time. We could keep certain processes in operation without disruption. This was key, as we never had minimal testing time.

4.5 Third research iteration

As supply increased to meet demand, the ability for all providers to request vaccine in a more direct manner was of interest. For the first time during the development process, this was not "emergency based". With this said, the goal was to create our next iteration of the system based on current applications and new requirements such as the ability for each of the 2000 + approved representatives in the state's vaccination program to log-in and request vaccine for their respective entities while maintaining current capabilities. A later observed requirement was integration with existing email accounts that were registered to the state. This required new cyber-security concerns that would need to be handled. Furthermore, the addition of 2000 + users created the need for more resources. The development team took on a new developer, and the JIATF used existing resources to stand up and create a help desk. GitHub would be used to communicate errors and issues from the help desk to developers, and the JIATF would be responsible for enrolling and submitting user names and credentials to the development team. As third party developers, the development team could not be responsible for an unenrolled user getting access. Thus, it was left to the JIATF to review all users and then submit them to the application.

These new requirements not only required all current builds of every application to be changed or reconsidered but also was a major shift in distribution protocols for the state, meeting the changing needs of the public health dynamics in the state. The rebuild also allowed us to focus on streamlining applications targeted at different users as opposed to a part of a process, which had previously been the approach due to phased development. A three week timeline was given for this massive rebuild, the application was also to be rebranded and take a new name, emergency inventory management system (EIMS). New applications had to be developed to give users access to request vaccines, and for JIATF stakeholders to review these requests and integrate with any needs they may be planning, such as state run vaccination clinics. Additionally, new applications were needed to manage the approved users and align them with immunization policies at a state and federal level. With this new request network, the unification of processes from the JIATF to hubs was also to be considered.

The first step was to add a second Shiny Server that was used to account for 2000 users entering the system, but it also served as a partition between request and distribution of vaccine. The original server was to be kept to be accessed by JIATF stakeholders only. An elastic file system (EFS) would be used to bind the two servers and allow communication of data when necessary. The new applications to manage user access from the JIATF were to be hosted on the original server. The combination and reduction of capabilities from the original application can be found in Fig. 4. The new distribution process had all requests for the coming week to be put in by Tuesday 11:59PM, and then would lock users out from requesting. Request files would be created and uploaded to a new application where stakeholders could add requests, approve, disapprove or change on Wednesdays. Once approved, the distribution report would be created off the approved data and compiled appropriately. Figure 4 shows the full diagram of the system, specifically how the system is split to handle the influx of users, while maintaining security by regulating access through dual Shiny servers.

The system went live May 14, and had a two week burn in to register users. Capabilities for direct shipments to providers as well as others had modules in place, but national distribution strategies were not yet to a point where this implementation would be useful. The minimum order quantities for major vaccines do not meet requirements for rural areas to this point, given limited demand as time has moved on.

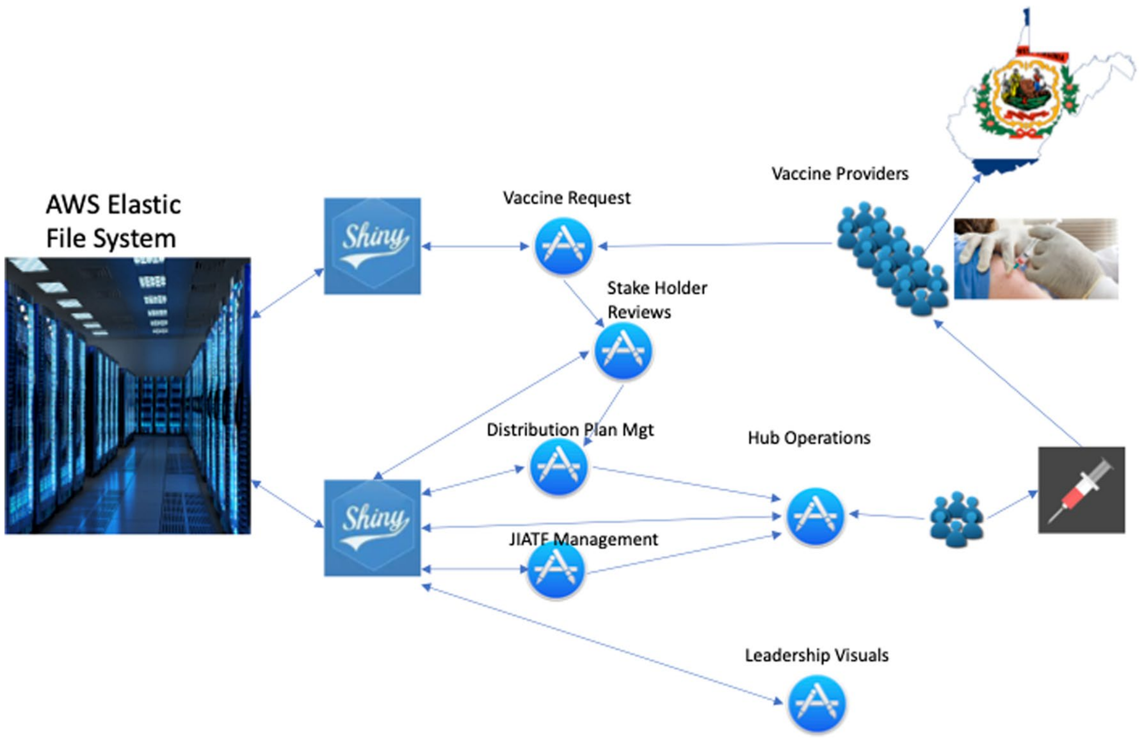

Fig. 4 Research iteration three which allows for vaccinators to directly request vaccine and stakeholders to approve each week, while maintaining visibility into hub operations. The dual servers connected by the Amazon Web Services Elastic File System, serves dual purposes: To deal with load balancing for 2000+vaccinators using the system, while segmenting user privileges to management of vaccines and requesting vaccines

As of October 2021, this approach has been responsible for 1.8 million+doses of vaccine and played a role in achieving 66% of the eligible population having received at least one dose of vaccine.

This implementation was not without some bumps in the road. The role of creating a single sign on utilizing AWS Cognito with a Microsoft hosted sign on is not a non-trivial feat. In addition, the additional developer had a 2-week burn in on the system prior to being able to be effectively deployed and even then some of the "lessons learned" from the prior implementations had to be re-learned by the new developer. Also, some of the more "cutting edge" solutions that can be found on common help sites such as stack-overflow and others in the open source community may not scale to a larger user base.

4.6 Lessons learned during the third iteration

(1) Practice the philosophy of "build to tear down." If we think of retaining the system beyond this emergency, we will create more than is necessary and cannot be as agile as needed. An example from this system would be storage of data in a file system rather than a complex database structure. Throughout the project, we had to maintain a sharp focus on which applications can be created rapidly to be put in production quickly. This often meant making choices that may not work in the long term and only consider what would work in the near future, but that constraint was part of the volatile nature of the context that we were dealing with. One basic tenet of agile development is welcoming changing requirements. We needed to expand this tenet drastically to the extent of re-imagining and redoing several moving parts of the project, often in short periods of time.

(2) Strive for stability in team membership by minimizing team composition changes. The development team realized early on that the knowledge gained during the first few iterations was complex and not easily transferable. The members also understood that they did not have the time and energy to expend on onboarding new members. Hence, conscious efforts were made to minimize core team composition changes until the project reached a steady state. Onboarding could be done once the system achieved some major milestones, but can cause chaos if the new member is not familiar with the principles and system requirements across the board. This is also true for the tools and technologies used in creating the application; for instance, the use of a package could override functionality across the system.

An example of this would be when we used dplyr (a newer version of plyr) which overwrote the namespace used by plyr, causing the functionality to fail. Overall, members of the team need to be very familiar with one another and should use a common set of tools with which they are already familiar. Avoid bringing in members who are not familiar with the core team or with the tools being used.

(3) Span the boundary beyond the core. Although the core development team needed to minimize team composition changes, it was necessary for the team to embrace change from the client side. The client side often involved representatives from the various government agencies that made up the JIATF. The dynamic situation based on stakeholders who were engaged at each time, led to individuals who would only be engaged at certain times (depending on situations) forcing the core team to adapt their communication and working strategies. Overall, the team needed to practice ambidexterity to be able to accept changing members to the overall team, while maintaining minimal composition changes to the core team. A corollary of this practice was that the core team members needed to be boundary spanners, needing to connect with new members, exploring and understanding the different facets of the distribution process continuously and bringing it back to the development team to incorporate the functionalities in the system. This could be seen as connecting with vaccinators to better understand the minimal amount of information they wanted to enter to request vaccine or with hubs to determine their process to make the system more efficient. It also corresponds with communicating with leadership about the entities that would be engaged in the distribution process. Moreover, prioritizing needs versus wants was also an important task for the development team. Clear expectations are necessary, but also understanding the outcome is more important than efficiency at first. User input matters, but in the immediate, too much user input may become a problem. Users may ask for bells and whistles and may think that their suggested feature is immediately important. They would discuss the software/application in its final state rather than the possible paths to the functional solution. As a developer charged with providing an emergency response system, it is paramount to know how to prioritize requirements over complete solutions. In such high cadence scenarios, the development team needed to decide what is an emergency and what is a want. From the outset, everything may feel like an emergency because of what is on the line. However, some things will need to wait and can have longer time lines (Table 3).

5 Discussion

This research project was undertaken to respond to an immediate need for an information system that would provide visibility of data into the COVID-19 vaccine distribution process for a U.S. state. Although we adopted an agile development methodology for the development of the system, we were aware that the pace of the project requirements was beyond the capabilities of agile development practices. We found that with carefully chosen adjustments, the agile

Table 3 Project milestones

Date	Milestone
15-Jan	Project scope and start
22-Jan	Stakeholder upload and JIATF management apps deployed
1-Feb	Requirements and enhancements for dynamic supplies
5-Feb	Capacity for additional vaccine types and doses
15-Feb	Real time hub integration to application
27-Feb	Real time visualization on inventory and hub operations added
15-Mar	Hub operations and change orders features operational
20-Mar	Receipt of vaccine shipments added
30-Mar	Begin maintenance on version 1 (protocol changes around distribution and Prioritization
15-Apr	Version 2 scoped and development begins
23-Apr	Automation of stakeholder review launched
30-Apr	Automation of distribution report of provider confirmations
10-May	Integration with state systems and provider enrollment/training begins
16-May	Update distribution restrictions for providers and introduce pediatric doses
30-May	Version 2 launches

methodology can be employed to meet the hyper-agile pace of the project. These adjustments allowed us to distill important lessons at the system, process and team levels.

5.1 System-level lessons

In the course of the development, we found that the information required to make important decisions came from a plethora of sources and at a rapid pace. Timely and accurate information about demand and supply are crucial for efficient flow in supply chains [46]. While providing visibility to the stakeholders has always been an objective of paramount importance in any information system and especially needed in supply chains [47], the development team had to walk a thin line between providing just enough functionality and overloading the end-users with more features than needed. With information overload, research suggests that irrelevant information may cause useful information to be ignored due to a deluge of unnecessary facts and figures, leading to poor decision making [48]. The team often needed to carefully assess how to develop modules and reports that had just enough functionality. At this crucial stage, the team had to be extremely vigilant about making the process easy through useful graphs that had drill-down capability, but at the same time were cautious to hold back features that had the potential to cause confusion for the end-user who had limited experience using a system like this one.

In emergency situations, it is very likely that the requirements are so convoluted that they cannot be easily attained by creating a complex system that has several perfectly integrated modules working in perfect harmony. Instead, we found it prudent to think of major requirements as systems on their own. This enables the development team to focus on solving one major requirement and then move on to another. Each major requirement is attained through an individual system. The current agile development approach is to build systems that are continuously integrated and have a high degree of connectivity [24]. Our approach was contrary to this agile practice. We strived to create loose coupling in systems such that each system handles its own major requirement while maintaining minimal level of connectivity. This promotes flexibility in the overall set of applications. The different systems allowed each unit to take responsibility for sub-processes (activities) that formed a complex higher-level process. Each application helped the stakeholders manage their own chunk of the complex overall process while maintaining some requisite coordination [49, 50]. This mechanism of handling major requirements with different applications allowed a combination of capabilities that manifested greater responsiveness to the environment [51].

Finally, at the system level, we found it judicious for the development team to play the role of gatekeepers with respect to the requirements that will be of utmost importance and those that could wait. As the development team engaged with different stakeholders in the process of gathering requirements, they were given ideas for several functionalities. It fell upon the development team to decide what constituted the most judicious usage of their resources. They had to analyze whether a requirement was really a need or a want. Agile development methods stress the customers' role during the entire development process by involving them in discussing product features and prioritizing the feature list [52]. The customer is assigned the role of the product owner who tells the development team what is important to deliver [53]. However, in an emergency situation, we found that the development team has to assume the role of the product owner. This role fell upon the development team because they had a complete picture of all the parts of the process and thus were better able to decide how best to utilize the capabilities to meet the need.

5.2 Process-level lessons

Agile development practice espouses short iterations [9]. The development team needed to extend this practice to the level of completing iterations in weeks and sometimes even days. Due to the nature of the urgency, the development team had to resolve to adopt an all-hands-on-deck approach to make sure that development did not stop until the application was up and running. The team also needed to work with lightweight tools. Agile development practices do not give any suggestions concerning the tools that would be better suited for development tasks. It has been our observation, however, that tools which are lightweight and can be quickly put together are best suited for systems that need to be developed at an urgent pace. Research suggests that agile development practices and open source software development share several principles and practices that may help support the utilization of best practices from both [54]. Lightweight tools are often developed with the goal of frugal use of resources to maximize output that is immediately useful for the end-user [55, 56]. These lightweight tools often have supporting online communities working on a voluntary basis that help resolve any functional issues. The added support through these communities may help overcome any technical hurdles in building the system [57, 58]. Often this level of support is non-existent for more established industry standard, proprietary software. The support networks for such tools are often slow to respond and issues with the capabilities of software are often resolved in the next iteration of updates that can come in months, if not years.

When responding to an emergency, development teams may think that they have to test the system thoroughly to make sure every functionality is operational. The agile practice of comprehensive unit testing is key to ensuring that each functionality is extensively tested [59]. Although that should be the goal for an ideal scenario, emergencies cannot wait for perfection in response. The development team had to resort to minimal testing and primarily testing only the major functionalities. This is one possible mechanism to enable quick iterations that match the pace of the emergency [60]. This does not mean that the possibility of failure should not be assessed. We worried about failure every morning we woke up. Partially it was imposter syndrome, partially because we knew what was at stake and what would happen if it failed. We tried to build it as robust and independent as possible to the point where all data is recorded and stored, and logs are tracked so that even if something fails the data is recorded and we can restore it. There are redundancies and version control is of utmost importance when building. There were times the system went off line for hours to bug hunt, because it was critical. This is where the lack of testing was a problem. We could not stress test, but partially because it was a new process on all sides. Overall, testing had to be relegated to a back seat due to the minimalistic approach that we had to espouse.

The development team also needed to be flexible with respect to the functionality provided by the current applications as they faced changing requirements. They realized that there was often a need to rethink the purpose and scope of a previously built application. They had to be open to the idea of completely tearing down an application to build another one. As the requirements changed, the functionality provided by a certain application would not match the change in requirements. This meant rebuilding the applications to match the new needs. Although this may seem to be a waste of time, with requirements that are fickle, holding on to previously built applications was infeasible.

5.3 Team-level lessons

We also learned important lessons at the team level. First, we found that the core development team had to maintain stability in the core, which meant that new members were not allowed to join even when there was a need for helping hands. The team was put together in an emergent manner and there were no formal processes followed. As the core team came together, they strived to keep the core stable. The core team members realized that they did not have the capacity to allow new members as that would have put a greater strain on the time constraints due to on-boarding and training needs of new members. Team instability due to addition of new members can severely impact team cognitive structures and can be harmful to team effectiveness [61, 62]. At the same time, membership from the stakeholder side varied markedly. This goes against the agile development practice of maintaining stable teams. The core team needed to maintain stability at the core while embracing changing membership at the periphery. In order to thrive in this dual natured context, the development team had to strive to attain a balance between the two conflicting team structures. The

Table 4 Extensions needed for adapting agile practices to hyper-agile contexts

Agile practice	Needed extension for hyper-agile context
Continuous integration	Divide and conquer: Requirements can be more easily managed when different applications are created to meet the major needs. A single, completely integrated application is a much more complex and time-consuming target to achieve
On-site customer	Assume gatekeeper role: Development team assumes the role of a gatekeeper to decide which functionalities are provided by the systems. This is important because it is the development team that has the complete picture of the overall process
Development focuses on functionalities identified for a sprint	Functionality to meet emergency needs: Development must maintain a minimalistic, frugal approach to provide the functionalities that are just enough to meet emergencies that emerge on a daily and sometimes hourly basis
Short iterations (a few weeks)	Extremely short iterations (days): Iterations of the agile process should be kept as short as possible in hyper-agile contexts. The team can also adjust the iteration duration as needed, contrary to the fixed duration as per standard agile practice
No recommendations for tools	Use open source tools: Open source tools are often lightweight and thus can be very useful in quickly achieving the needed functionality when working in hyper-agile contexts
Testing first	Minimal testing: Contrary to the agile standard of testing every feature before release, teams will need to resort to minimal testing. The team will need to build redundancies in the system to prevent data loss
Refactoring and retrospectives	Build to tear down: As requirements change drastically, there may be need to completely tear down existing applications to create new ones
Stable team	Ambidextrous team: The development team will need to strive for stability at the core but be fluid at the periphery

team adopted an ambidextrous attitude and strove to gain efficiency at the core while combining it with a sharp focus on the changing needs at the periphery of the team. Research shows that teams that often need to work in conflicting environments adopt ambidexterity. This enables teams to form explorative and exploitative routines, where the former promotes flexibility while the latter focuses on attaining goals related to stability, routinization and efficiency [63, 64]. The team members played the role of boundary spanners beyond the core to be able to make sense of the changing nature of the requirements and the distribution process which varied with changes in supply and demand as well as the capabilities of the client. A summary of all the extensions needed for adapting agile practices is provided in Table 4.

One might think that adapting agile methodologies by simply increasing the pace of iterations might be sufficient to meet the needs of hyper-agile environments. However, we found that just increasing the pace of the iterations was insufficient to effectuate a robust response to this hyper-agile environment. We had to adapt and go against several established agile practices. For instance, one cornerstone practice of agile development is the use of continuous integration. Continuous integration (CI) is a software development practice that requires developers to integrate software into the production environment often during development [65]. Agile teams often consider this as a useful practice that results in significantly reduced integration problems and allows the development of cohesive software rapidly [66]. Our context required us to take a divide-and-conquer approach instead. We found that requirements can be more easily managed when different applications are created to meet the major needs. Aiming for integration with previously created applications would have added an additional layer of complexity as integration points would need to be careful decided upon. Research shows that CI poses challenges in identifying component dependencies during development and integration [67] which could potentially cost more time or break a working application by introducing bugs into the production environment. Overall, component interfaces need to be clearly defined, more failures can be experienced during integration and there is a need to wait until other components are completed before integrating work [68].

Another agile practice that we had to forego was that of having an on-site customer representative. On-site customer and its extension, active stakeholder participation, are important agile practices that promote close interaction between the development team and the customer. Increasing the pace of agile would have meant getting the user to join the team more often to make sure that requirements are prioritized and developed to provide value at a faster pace. This would have also meant getting the user to give requirements faster than the usual pace. However, the emergent nature of the context characterized by a hyper-agile pace of requirements resulted in the customer being not confident of what the process was supposed to be. The process was actually non-existent. The development team had to become gatekeepers of the project so as to be able to define the process and prioritize the requirements to meet the needs. Oftentimes, the development team needed to turn down requirements requests from the customer as they were deemed not functionally critical. The development team had to make decisions as to which functionality is a crucial need with high priority rather than a mere want that the users may consider good to have. Another reason for assuming the gatekeeper role was that the users were not aware of the entire business process and often understood only a subset of the system needs. Educating the user would have also cost time which was a critical resource in this project.

Testing is another key recommended activity during agile development. Agile development suggests integrating testing into the development process instead of having it as a separate phase [69]. Testing continues even as development of the entire application has not completed and more features are being added. It can consist of a plethora of different types of tests, such as unit tests, integration tests, UI tests, acceptance tests and exploratory tests, to name a few [70]. Speeding up the standard agile process would have meant accelerating the iteration cycle to have shorter, more frequent sprints. This would have resulted in compacting more testing in the iteration as more functionality needs to be built, tested and deployed in a small amount of time. This would have delayed the deployment of major mission critical features needed for vaccine distribution. For this reason, we decided to keep testing to a bare minimum level. Our multi-application design enabled us to put in needed enhancements with minimal testing. If a bug was discovered in an application at a later point in time, it impacted only that application. The other applications and related processes were able to continue operation without disruption.

Refactoring and retrospectives are two distinct activities in agile software development. Refactoring is related to making minor adjustments to code such that its internal structure improves but overall functionality stays the same [71]. It is undertaken to improve the non-functional attributes such as design, structure or implementation of code, while preserving overall functionality. This has the potential to improve the code's simplicity, clarity, internal structure or extensibility. Retrospectives is the practice of inspecting how the past Sprint went. Essentially, this is an exercise to determine what went well, what problems were encountered, and how those problems were or were not resolved [72]. Research suggests that coordinating emergency responses often requires breaking established protocol if it may negatively affect the outcome [7]. Speeding up sprints in the iterations would mean that refactoring and retrospective would have to be undertaken more often, costing more time in development and distracting the development team's attention from developing new functionalities to perfecting existing functionality. Foreseeing the disadvantage of this, the development team consciously chose to revisit an application's functionality only when it was unavoidable, such as when it severely constricted the process by producing bottlenecks or producing erroneous outcomes.

Another beneficial practice of agile development is that of maintaining stable development teams. Stable teams allow team members to spend time together and have frequent interactions which translates to strong cohesion among the members making them high-performing [73]. Research confirms that high team stability leads to higher amount of work completed at the end of each iteration by the team [74]. Stability in teams also allows for a higher capability to build knowledge and gain mastery of the problems and tradeoffs in a business domain [75]. Emergent teams, however, do not have the luxury to spend time and energy to build stable relationships that translate to all the above-mentioned benefits [7]. We had to resort to maintaining a team that was stable at the core but that allowed members to move in and out at the periphery. This created a team that was ambidextrous as it maintained focus at the core while maintaining a fluid membership at the periphery.

The remaining three extensions (minimal functionality, extremely short iterations and using open source tools) are related to adopting the concept of frugality throughout the development process. In agile development minimal functionality is a key practice. However, the main goal is to gain further feedback from the customer and add more features as per the priorities decided upon in the set of features of the overall project. In our hyper-agile context the goal was not to add these bells and whistles at a later stage. As a matter of fact, requests for any bells and whistles were discouraged by the development team. This practice allowed the team to maintain a sharp focus on the frugal list of functionalities that it had come up with. The concept of frugality was also relevant to the tools selected for development. As functional viability was the only metric for success, we chose lightweight tools that enabled us to use pre-existing components that could be quickly deployed. Lightweight tools are often developed with the goal of frugal use of resources to maximize output that is immediately useful for the end-user [55, 56]. None of the agile methodologies (XP, Scrum, etc.) provide any suggestions about any development tools to be used. Keeping iterations short is a key suggestion when working in an agile environment. The standard suggestion can range from weeks to months and the duration is set as a standard across all iterations. For our hyper-agile context, we suggest pushing this suggestion further by using iterations over only a few days. We also suggest that the team should be able to adjust the durations as needed, contrary to the standardized duration across all iterations in standard agile development.

In any flavor of agile development (e.g. Scrum, XP, DSDM, etc.) features are developed quickly, tested frequently and integrated continuously into the main code base. Feedback is encouraged at every step, clients are involved throughout the process, and development continues until the client is satisfied. For hyper-agile development, we believe the suggested extensions are crucial as simply speeding up the iterations in Scrum, XP or another agile flavor would not make the team hyper-agile. Instead, it may have the contrary effect of delaying development. This is primarily due to the emergent nature of the context where the process is unknown, the team members have no past experience of working together and the client is not available to test out each feature to perfection. Simply speeding up the iterations would have forced the development team to repeatedly undertake practices that were not supportable. For instance, having an on-site customer who was unaware of the overall process and the different sources of data would have wasted the development team's time as they would need to first educate the customer about the process and its complexities. Similarly, the standard practices of refactoring and retrospectives would have forced the team to take up a continuous cycle to reevaluating the code base which would have put additional time constraints on the development team. For these reasons, we believe that existing agile frameworks are deficient in nature to cater to the needs of this hyper-agile context. The suggested extensions are crucial as they help better match the emergent, hyper-agile context of this project.

6 Contributions and conclusion

The development of a software system at an extremely fast pace by an emergent team while using agile development methodology is a complex task. This research reported the process followed and the lessons learned during the development of a system which was quickly put together to achieve efficient COVID-19 vaccine distribution capability. We offer several important contributions with this project. First, we offer important lessons for the application and adaptability of the agile development process at the system, process and team levels. Although agile development is meant to offer fast-paced results, the pace of this project was beyond any that we find in the software development literature. With this project we now know that the agile process can be adapted but with careful adjustments. At the system level, the development team has to strive for minimal functionality, adopt a divide and conquer approach while assuming the role of the gatekeeper of the system. At the process level, the team has to maintain the shortest possible iterations, use open source tools and perform minimal testing flanked by carefully managed redundancies. At the same time, they have to assume a build-to-teardown approach to be pliant with process changes. At the team level, the team has to adopt an ambidextrous stance by striving for a stable core set of members but being flexible to allow changes at the periphery. Although some research does show that agile methodologies can be tailored by software development teams as they prefer to use selective agile practices [30, 31], the literature does not show whether the practices themselves can be tailored. Our research contributes to this literature by showing how individual agile practices can be adapted to match the need of a highly mission-critical system. We were able to adapt agile practices by extending them to achieve minimal functionality, dividing and conquering, gatekeeping, building-to-teardown, and achieving ambidexterity in team.

Second, we contribute to the emergent teams' literature by studying how emergent teams can take on the challenge of developing a system that has the potential to save lives. The team achieved a dual nature as an emergent team that strove for stability at the core while embracing flexibility at the periphery. The emergent nature of the team meant that team memberships and member roles could change, but at the same, to achieve stability at the core, the membership of the main developers had to be preserved. Moreover, we show that in the absence of any metastructures guiding the team's coordination efforts [7], the team evolved dialogic coordination practices [34] by impromptu sense-making practices (unscheduled discussions and interviews, hub visits) which allowed members to develop a sufficient level of shared understanding of the problem. The lessons and contributions from this research serve as a first step to understanding best practices to follow in emergent contexts characterized by unprecedented health challenges.

Generalizability is a concern in a study like this one. Although we recognize that our context is somewhat unusual, we do believe that there are findings that can be generalized to other contexts and organizations. Even though many organizations use standardized processes, there are scenarios within organizations where organizational members need to make teams that are emergent in nature. For instance, we offer important lessons for disaster recovery situations caused by natural disasters like flooding, hurricanes and tornadoes which call for swift action by emergent teams. Moreover, as businesses recover from COVID-19 related lockdowns, there is a need to reflect on past experiences to gain opportunities to inculcate resilience that enables quick recovery from disturbances in the future [76, 77]. The lessons learned from our hyper-agile vaccine distribution context can facilitate gaining this understanding. Finally, our suggested extensions to the agile development methodology may also be applicable to other fast-response medical and non-medical settings where there is an urgent need to gain accurate visibility into supply and demand processes. Future research is needed to corroborate our findings in other contexts where emergent nature of teams play a crucial role in the road to recovery from disturbances.

Funding The project described was supported by the National Institute of General Medical Sciences, 2U54GM104942-02. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH. Research reported in this publication was also supported by the National Institute On Minority Health And Health Disparities of the National Institutes of Health under Award Number 1U01MD017419-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

1. Elbanna A, Sarkar S (2016) The risks of agile software development: learning from adopters. IEEE Softw 33:72–79
2. De Cesare S, Lycett M, Macredie RD, Patel C, Paul R (2010) Examining perceptions of agility in software development practice. Commun ACM 53:126–130
3. Dingsoyr T, Nerur S, Balijepally V, Moe NB (2012) A decade of agile methodologies: towards explaining agile software development. J Syst Softw 85:1213–1221
4. John Hopkins University (nd.) Covid-19 dashboard by the Center for Systems Science and Engineering (CSSE) Johns Hopkins University. https://coronavirus.jhu.edu/map.html. Accessed 9 Sept 2021
5. Comito C (2021) How COVID-19 information spread in US? The role of Twitter as early indicator of epidemics. IEEE Trans Serv Comput. https://doi.org/10.1109/TSC.2021.3091281
6. Rowe F, Ngwenyama O, Richet J-L (2020) Contact tracing apps and alienation in the age of COVID-19. Eur J Inf Syst 29:545–562
7. Majchrzak A, Jarvenpaa SL, Hollingshead AB (2007) Coordinating expertise among emergent groups responding to disasters. Organ Sci 18:147–161. https://doi.org/10.1287/Orsc10600228
8. Harvie DP, Agah A (2016) Targeted scrum: applying mission command to agile software development. IEEE Trans Softw Eng 42:476–489
9. Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48:73–78
10. Holmstrom H, Fitzgerald B, Agerfalk PJ, Conchuir EO (2006) Agile practices reduce distance in global software development. Inf Syst Manage 23:7–18
11. Mishra D, Mishra A, Ostrovska S (2012) Impact of physical ambiance on communication, collaboration and coordination in agile software development: an empirical evaluation. Inf Softw Technol 54:1067–1078
12. Yu X, Petter S (2014) Understanding agile software development practices using shared mental models theory. Inf Softw Technol 56:911–921. https://doi.org/10.1016/j.infsof.2014.02.010
13. Bonaretti D, Piccoli G (2018) Effective use of information systems for emergency management: a representation theory perspective. In: Proceedings of 38th International Conference on Information Systems, ICIS 2018, San Francisco, CA

14. Silvius G (ed) (2016) Strategic integration of social media into project management practice: advances in it personnel and project management. IGI Global, Hershey
15. Oh O, Kwon KH, Rao H R (2010) An exploration of social media in extreme events: rumor theory and twitter during the Haiti earthquake 2010. In 31st International Conference on Information Systems, Saint Louis, MO
16. Leong CML, Pan SL, Ractham P, Kaewkitipong L (2015) ICT-enabled community empowerment in crisis response: social media in Thailand flooding 2011. J Assoc Inf Syst 16(3):174
17. Tim Y, Pan SL, Ractham P, Kaewkitipong L (2017) Digitally enabled disaster response: the emergence of social media as boundary objects in a flooding disaster: social media for disaster response. Inf Syst J 27(2):197–232
18. Vaast E, Safadi H, Lapointe L, Negoita B (2017) Social media affordances for connective action: an examination of microblogging use during the Gulf of Mexico oil spill. MIS Q 41(4):1179–1206
19. Lee G, Xia W (2005) The ability of information systems development project teams to respond to business and technology changes: a study of flexibility measures. Eur J Inf Syst 14:75–92
20. Hoorn JF, Konijn EA, van Vliet H, van der Veer G (2007) Requirements change: fears dictate the must haves; desires the won't haves. J Syst Softw 80:328–355
21. Maruping LM, Venkatesh V, Agarwal R (2009) A control theory perspective on agile methodology use and changing user requirements. Inf Syst Res 20:377–399
22. MacCormack A, Verganti R, Iansiti M (2001) Developing products on "Internet time": the anatomy of a flexible development process. Manage Sci 47:133–150
23. Sutherland J, Sutherland JJ (2014) Scrum: the art of doing twice the work in half the time. Random House, New York
24. Wang X, Conboy K, Pikkarainen M (2012) Assimilation of agile practices in use. Inf Syst J 22:435–455. https://doi.org/10.1111/j.1365-2575.2011.00393.x
25. Fitzgerald B, Hartnett G, Conboy K (2006) Customizing agile methods to software practices at Intel Shannon. Eur J Inf Syst 15:197–210
26. Surendra NC, Nazir S (2019) Creating 'informating' systems using Agile development practices: and action research study. Eur J Inf Syst 28:549–565
27. Kude T, Mithas S, Schmidt CT, Heinzl A (2019) How pair programming influences team performance: the role of backup behavior shared mental models and task novelty. Inf Syst Res 30:1145–1163. https://doi.org/10.1287/isre.2019.0856
28. Waizenegger L, McKenna B, Cai W, Bendz T (2020) An affordance perspective of team collaboration and enforced working from home during COVID-19. Eur J Inf Syst 29:429–442
29. Kirkman BL, Shapiro DL (2001) The impact of cultural values on job satisfaction and organizational commitment in self-managing work teams: the mediating role of employee resistance. Acad Manage J 44:557–569
30. Mohammed S, Ferzandi L, Hamilton K (2010) Metaphor no more: a 15-year review of the team mental model construct. J Manage 36:876–910. https://doi.org/10.1177/0149206309356804
31. Ågerfalk PJ, Conboy K, Myers MD (2020) Information systems in the age of pandemics: COVID-19 and beyond. Eur J Inf Syst 29:203–207. https://doi.org/10.1080/0960085X20201771968
32. Drabek TE, McEntire DA (2003) Emergent phenomena and the sociology of disaster: lessons trends and opportunities from the research literature. Disaster Prev Manage 12:97–112
33. Faraj S, Sproull L (2000) Coordinating expertise in software development teams. Manage Sci 46:1554–1568
34. Faraj S, Xiao Y (2006) Coordination in fast-response organizations. Manage Sci 52:1155–1169
35. Blickensderfer E, Cannon-Bowers JA, Salas E (1997) Theoretical bases for team self-correction: fostering shared mental models. In: Beyerlein MM, Johnson DA, Beyerlein ST (eds) Advances in interdisciplinary studies of work teams: team implementation issues, vol 4. JAI Press, London
36. Mohammed S, Dumville BC (2001) Team mental models in a team knowledge framework: expanding theory and measurement across disciplinary boundaries. J Organ Behav 22:89–106
37. DeSanctis G, Monge P (1999) Communication processes for virtual organizations. Organ Sci 10:693–703
38. Yang HD, Kang HR, Mason R (2008) An exploratory study on meta skills in software development teams: antecedent cooperation skills and personality for shared mental models. Eur J Inf Syst 17:47–61. https://doi.org/10.1057/palgrave.ejis.3000730
39. Schein EH (1995) Process consultation action research and clinical inquiry: are they the same? J Manage Psychol 10:14–19
40. Baskerville RL, Myers MD (2004) Special issue on action research in information systems: making IS research relevant to practice—foreword. MIS Q 28:329–335
41. Baskerville R, vom Brocke J, Mathiassen L, Scheepers H (2020) special issue call for papers: clinical research from information systems practice. Eur J Inf Syst
42. Khisro J, Lindroth T, Magnusson J (2021) Mechanisms of constraint: a clinical inquiry of digital infrastructuring in municipalities. Transform Gov 16(1):81–96
43. Magnusson J, Nilsson A, Lindroth T, Khisro J, Norling K (2022) Rhizomatic strategizing in digital transformation: a clinical field study. In: Annual Hawaii International Conference on System Sciences.
44. Heracleous L (2022) Helping at NASA: guidelines for using process consultation to develop impactful research. Inf Organ 32(1):100388
45. Schlüter J, Vetter F (2020) An interactive visualization of Google Books Ngrams with R and Shiny: exploring a(n) historical increase in onset strength in a(n) huge database. J Data Mining Digit Humanit 7:1–25
46. Mohr J, Spekman K (1994) Characteristics of partnership success: Partnership attributes communication behavior and conflict resolution techniques. Strateg Manage J 15:135–152
47. Wang ETG, Wei H-L (2007) Interorganizational governance value creation: coordinating for information visibility and flexibility in supply chains. Decis Sci 38:647–674. https://doi.org/10.1111/j.1540-5915.2007.00173.x
48. Xie C, Wu D, Luo J, Hu X (2010) A case study of multi-team communications in construction design under supply chain partnering. Supply Chain Manage 15:363–370
49. Gosain S, Malhotra A, El Sawy OA (2005) Coordinating for flexibility in e-business supply chains. J Manage Inf Syst 21:7–45
50. Orton DJ, Weick KE (1990) Loosely coupled systems: a reconceptualization. Acad Manage Rev 15:203–223
51. Christopher M (2000) The agile supply chain competing in volatile markets. Ind Mark Manage 29:37–44
52. Hoda R, Noble J, Marshall S (2012) The impact of inadequate customer collaboration on self-organizing agile teams. Inf Softw Technol 53:521–534
53. Schwaber K, Beedle M (2002) Agile Software Development with SCRUM. Prentice-Hall, Hoboken
54. Gandomani TJ, Zulzalil H Ghani AA, Sultan AB (2012) A systematic literature review on relationship between agile SD and open source SD International review on computers and software (IRECOS) 7:1602–1607. Accessed from https://arxiv.org/abs/1302.2748
55. Anders G (2014) How to launch a billion dollar startup on a shoestring. Forbes. https://www.forbes.com/sites/georgeanders/2012/05/02/thrifty-startup/. Accessed 10 Sept 2021

56. Ahuja S, Chan Y (2014) The enabling role of IT in frugal innovation. Proceedings of ICIS.
57. Chou S-W, He M-Y (2011) The factors that affect the performance of open source software development – the perspective of social capital and expertise integration. Inf Syst J 21:195–219. https://doi.org/10.1111/j.1365-2575.2009.00347.x
58. Scacchi W, Feller J, Fitzgerald B, Hissam S, Lakhani K (2006) Understanding free/open source software development processes. Softw Process Improve Pract 11:95–105
59. Gary K, Enquobahrie A, Ibanez L, Cheng P, Yaniv Z, Cleary K, Kokoori S, Muffih B, Heidenreich J (2011) Agile methods for open source safety-critical software. J Softw 41:945–962. https://doi.org/10.1002/spe.1075
60. Mäntylä MV, Adams B, Khomh F, Engstrom E, Peterson K (2015) On rapid releases and software testing: a case study and a semi-systematic literature review. Empir Softw Eng 20:1384–1425. https://doi.org/10.1007/s10664-014-9338-4
61. Davern M, Shaft T, Te'eni D (2012) Cognition matters: enduring questions in cognitive IS research. J Assoc Inf Syst 13:273–314
62. Kude T, Bick S, Schmidt CT, Heinzl A (2014) Adaptation patterns in agile information systems development teams. In: Proceedings of the 22nd European Conference on Information Systems. pp 1–15
63. Gibson CB, Birkinshaw J (2004) The antecedents consequences and mediating role of organizational ambidexterity. Acad Manage J 47:209–226
64. Vinekar V, Slinkman CW, Nerur S (2006) Can agile and traditional systems development approaches coexist? An ambidextrous view. Inf Syst Manage 23:31–42
65. Dingsøyr T, Lassenius C (2016) Emerging themes in agile software development: Introduction to the special section on continuous value delivery. Inf Softw Technol 77:56–60. https://doi.org/10.1016/j.infsof.2016.04.018
66. Fowler (2006) Continuous integration. https://martinfowler.com/articles/continuousIntegration.html. Accessed 10 May 2022
67. Olsson H, Alahyari H, Bosch J (2012) Climbing the "stairway to heaven"—a multiple case study exploring barriers in the transition from agile development towards continuous deployment of software. In: 38th EUROMICRO Conference on Software Engineering and Advanced Applications, 392–399 (September)
68. Debbiche A, Dienér M, Berntsson Svensson R (2014) Challenges when adopting continuous integration: a case study. In: Jedlitschka A, Kuvaja P, Kuhrmann M, Männistö T, Münch J, Raatikainen M (eds) Product-focused software process improvement. PROFES 2014. Lecture notes in computer science, vol 8892. Springer, Cham
69. Prasad GP, Hamsini R, Smitha GR (2016) Agile development methodology and testing for mobile applications—a survey. Int J New Technol Res 2(9):263424
70. Vocke H (2018) The practical test pyramid. https://martinfowler.com/articles/practical-test-pyramid.html. Accessed 11 May 2022
71. Fowler M (nd) https://refactoring.com. Accessed 12 May 2022
72. What is a sprint retrospective? https://www.scrum.org/resources/what-is-a-sprint-retrospective. Accessed 15 May 2022
73. Verwijs C (2022) In-depth: stable or fluid teams? What does the science say? https://www.scrum.org/resources/blog/depth-stable-or-fluid-teams-what-does-science-say. Accessed 17 May 2022
74. Scott E, Charkie KN, Pfahl D (2020) Productivity, turnover, and team stability of agile teams in open-source software projects. In: 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA): 124–131, doi: https://doi.org/10.1109/SEAA51224.2020.00029.
75. Narayan S (2018) Products over projects. https://martinfowler.com/articles/products-over-projects.html. Accessed 19 May 2022
76. Boh WF, Constantinides P, Padmanabhan B, Viswanathan S (2020) Call for papers MISQ special issue on digital resilience. MIS Quarterly. Accessed from https://misq.org/skin/frontend/default/misq/pdf/CurrentCalls/DigitalResilience.pdf
77. Sakurai M, Chughtai H (2020) Resilience against crises: COVID-19 and lessons from natural disasters. Eur J Inf Syst 29(5):1–10. https://doi.org/10.1080/0960085X.2020.1814171

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.