Tom Lancaster
CS 510 – Agile
Midterm
Summer 2019

Short Answer (5 – 10 sentences)

1. Agile is always the best possible software development process. True or False? Why?

False. Agile is not a perfect software development process for every company. Agile takes an incremental process of building up software. This works fine if you are making a product or a service that can be tested several times before deployment such as a finance program or a signup application. For industries where it is critical to achieve success on the first try, then Agile wouldn't work for them. For example, a pacemaker must work the first time it is placed in a patient because if it doesn't function properly, then the patient is dead. Another example is launching a space shuttle. You cannot have the space shuttle blow up or fail to launch due to the astronauts' lives at stake and the money spent on the project. Therefore, Agile is *not* always the best possible software development process.

2. What are the two roles in pair programming? What is each role responsible for?

The two roles for pair programming are the *driver* and the *navigator*. The driver is the person coding the program and has control over the computer. The navigator is the person not typing and thinks about what the driver will do next. The driver should not be switching between modules unless agreed upon by both parties. Navigators shouldn't be pointing small coding mistakes or the driver's programming choices such as missing semicolons and while versus do-while. Each pair of programmers should not be the driver for more than an hour and should consider switching roles every half an hour or so.

3. Describe the benefits of timeboxing. What are its drawbacks?

The benefits of timeboxing are saving time, working efficiently and avoiding the trap of "death by inches." By giving yourself a set time for research or discussion, you force yourself to stop, make the best decision and move on to the next task which saves time and is an efficient manner of working. Also, you avoid the trap of "death by inches" where you say you'll finish after the next small task is completed such as testing, fixing small bugs or researching a more "perfect" solution. Before you realize it, you may have lost 2 hours or even days that could have been time better spent elsewhere. The drawbacks are unsatisfied team members and potentially underdeveloped ideas. Since timeboxing forces decisions to be made at the end of the time limit, a consensus may not have been reached on an idea and the team members could be unhappy that their idea wasn't chosen which could cause the members not to give their best effort on the proceeding tasks. Another drawback is not fully developing an idea before production and therefore the team hits snags or delivers an inferior or inadequate product which affects the company's reputation and bottom line.

4. Define the terms pigs and chickens. Why do we express Agile teams with these terms? You are not allowed to use the joke in your definition.

A pig is a committed team member of the project who does the actual work to achieve the result. A chicken is a third-party who is interested in the outcome but is not a part of the day-to-day process. These terms are used to highlight the differences between being committed versus just being involved. The committed members go through the process of creating their product or service and putting time and energy into the project. The involved members monitor the outcome since they have a stake in the project, but don't have the same sweat equity as the committed team members. These differences matter organizationally and therefore could affect the result of the project.

5. Describe why the iterative life cycle is better suited to software development than the standard waterfall lifecycle. If you disagree with the previous statement, explain why.

An iterative life cycle is better suited to software development than the waterfall cycle because the release schedule is shorter, it gives a quicker return on investment, and there is more time for testing and any significant changes in the software. The iterative life cycle goes through all the same steps as waterfall - plan, analysis, design, code, test, and deploy - but in one-to-three months iterations. This allows quicker releases and therefore sooner realized revenue and return on investment. The quicker turnaround also provides more time for testing which tends to be skipped as deadlines draw near in the waterfall model. Any bugs and software issues can, therefore, be dealt with promptly. Some issues or bugs can even require a partial or complete restart of the project which sinks the time and money spent up to that point - an expensive mistake businesses do not want happening. Also if the project's stakeholders decide to change something midstream, then this process can handle the change better than the waterfall model. Overall, the iterative life cycle works better for software development than the waterfall lifecycle.

6. Discuss why the gatekeeper responsibility is so important for the Scrum master to have. How does it help to ensure the team is successful?

The gatekeeper responsibility of the Scrum master ensures no outside interference for each sprint. This means any queries or concerns about the project are shared to the Scrum master and not the team as to not take away concentration and therefore value from the project and company. If the Scrum master doesn't keep outside influence at bay, the team would then continuously be addressing concerns and questions, debating the merits of ideas, and changing design midstream which decreases productivity. By assuming the gatekeeper role, the Scrum master helps the team deliver their project which means success for the company. Any project delivered is better than no project delivered (or a partially functioning project that can serve no one.)

7. Why do we allow anyone one to come to and Scrum meeting but only allow pigs to talk? What would we risk by allowing chickens to talk during these meetings? And why allow chickens to talk during the demo?

Chickens, or anyone interested in the project but not on the team, may come to Scrum meetings to get updates on the project and see how the team is operating. They are not allowed to speak because they can disrupt the smooth flow of the meeting and they may not understand the entire context of the discussion. For example, if a chicken comes and asks for more details, then that is lost work time for most of the team. The chickens can talk during demos because they have had input on the vision of the project and normally are

Multiple Choice

1. Which of the following is not an agile value as specified in the Agile Manifesto?

a. Working software over comprehensive documentation

b. Immediate feedback over delayed responses

c. Individuals and interactions over processes and tools

d. Responding to change over following a plan

e. Customer collaboration over contract negotiation

Essay - (1 or more pages, single spaced)

1. The book spends a great deal of time talking about trust. Discuss why it is important to foster trust not only within the team but with management as well. Are there other groups/individual who are should be included when fostering trust? What are the best ways, in your opinion to create and maintain this trust?

According to Patrick Lencioni, the author of The Five Dysfunctions of a Team, trust is "the confidence among team members that their peers' intentions are good, and that there is no reason to be protective or careful around the group." (pg. 195) It is the foundation of effective teamwork (which is better than individual work.) Without trust within a team and an organization, dysfunction ensues in the forms of lagging sales and revenue and inferior products or services. To highlight the difference between a team that trusts one another and a team that distrusts one another, I am going to use the 2004 NBA Finalists Detroit Pistons and Los Angeles (LA) Lakers. The Detroit Pistons won the NBA championship that year not because of just talent, but also because they worked together as a team that trusted one another. The LA Lakers definitively had more talent but did not have trust. They had four future Hall-of-Famers in Gary Payton, Kobe Bryant, Karl Malone, and Shaquille O'Neal and a host of solid role players who had won at high-levels on previous teams. Trust can be the difference between success and failure and it all starts at the top.

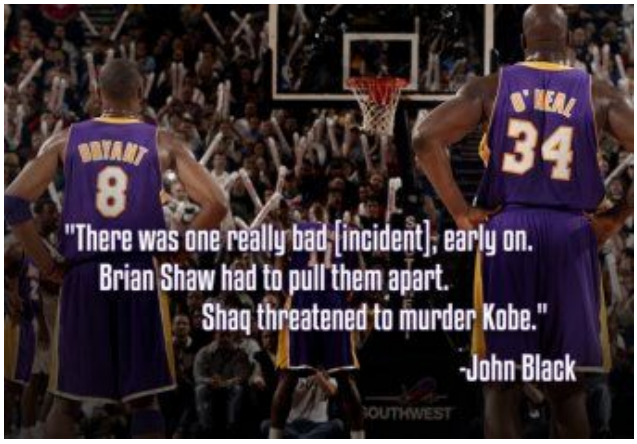
Fostering trust is critical from the top-down within the team and management. Management supports the broad, overall goals and must be able to rely on their product or service teams to do their jobs and do them well. If management distrusts the teams they manage, this can lead to a negative chain reaction. Employees conceal their weaknesses and mistakes from one another. They waste time and energy to monitor their behaviors and can resent upper management. They find ways to avoid spending more time at the office. For example, the 2004 Lakers' key players especially Kobe Bryant resented upper management's handling of the team and internalized all the issues to the point of apathy. Bryant was even quoted as saying, "I don't like (Phil Jackson) as a person, but I like him as a coach." (See legendary basketball coach Phil Jackson's The Last Season) The Pistons team, on the other hand, trusted the management's decisions. Larry Brown earned the trust and praise of players by holding everyone accountable. Even after the Pistons traded away popular locker room players Chucky Atkins and Lindsay Hunter for another key star player in Rasheed Wallace, the team continue to have faith in their coach and front office. The Pistons had trust in their management and the Lakers didn't. Management must understand their team and figure out ways to bring the best out of them. The lower-level teams must be able to trust that management has the overall organization's best interest at heart.

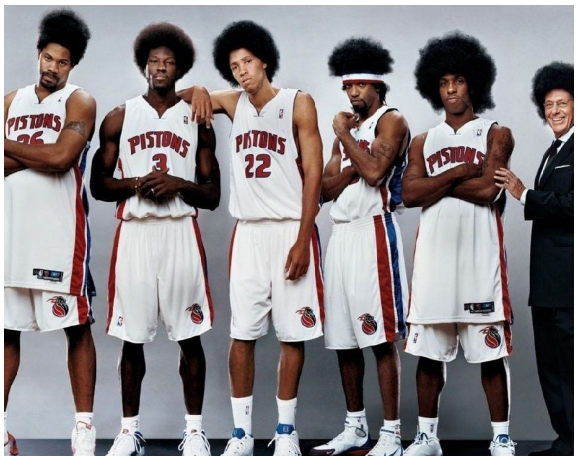
The trust within the team itself is also critical to success. Trusting teams ask for help, accept responsibility for their actions, focus on important issues, offer and accept apologies without hesitation and look forward to working together as a group. The Lakers did not do any of these particularly well especially due to star player Kobe Bryant. Bryant actively avoided teammates, would get caught up in the politics of the team and didn't accept responsibility for his failures. This created a negative dynamic within the team, but they advanced to the NBA

finals anyhow due to sheer talent. The Pistons did do all of these things well. No one was above the team and everyone was focused on one goal - to win an NBA championship. They had better team harmony and cohesion and just enough talent to beat the Lakers. Other ways teams like the Pistons build or foster trust is to admit weaknesses or mistakes. give each team member the benefit of the doubt, take risks in offering feedback, and look forward to meetings and time spent together. All of these actions lead to team unity and success.

Lacking in the 2004 NBA finalists example is the customers of the product. The fans cheer their NBA teams but don't have input in roster moves. In the software development world, the customer's input is essential to good software. If the customer does not like the product, then the bottom line of the company is affected. The programmers must have empathy for the customers and vice versa. The programmers must understand the customers' needs and the customers must trust the programmer's way of arriving at their solutions. Overall, trust is built through openness, good intentions and the ability to move past all the non-essential issues to focus on the main goal of delivering an awesome product or service. The 2004 Detroit Pistons are an example of what it means to have trust within a team and are now forever written as champions in NBA history books.

Which of these teams look more successful?





2. We talked briefly in class on merging an agile team with a team using the waterfall software development process. We determined it would be a bad idea. You are now tasked with implementing that bad idea. Describe what you would do to produce the best result possible.

Waterfall methodology is going through the five stages of the software development life cycle (SDLC) one stage at a time. The five stages are requirements, design, implementation, verification, and maintenance. A project team does not move onto the next stage until the previous stage is completed. Some problems with waterfall are not being able to know all future problems and customer desires, having less flexibility, the costs of redesign and implementation, and running out of time for testing due to deadlines. These and other problems have led many companies to adopt an agile methodology. Agile follows the same lifecycle as waterfall but does it in shorter iterations or "sprints" of typically two weeks. Merging a team that previously used waterfall methodology into an agile team midstream of a project is messy, but not impossible. In fact, 71% of all organizations have sometimes, often or always used the agile methodology. Since I am now tasked with merging both SDLC's to produce the best possible outcome, my plan of attack involves adopting the agile methodology in four main stages - building buy-in, investing in training, making practical changes to the teams, and being patient. (I am assuming my project would be a fit for agile methodology.)

The first phase is building buy-in. For agile to work, all pigs and chickens must be invested. A pig is a committed team member like a programmer or tester and a chicken is an interested member of the project but is not involved in the day-to-day activities. The waterfall team has become accustomed to having many months to complete a project and not moving to another stage in the SDLC until the previous stage is completed. I would first explain to the pigs and chickens the results found from 2008 QSM independent study of over 7,500 traditional and agile projects. The results showed that agile projects were 37% faster delivering their products to market and therefore accruing revenue much faster, 16% more productive than non-agile or traditional methodology, and able to maintain normal defect levels despite a compressed schedule. I would then have members from the agile team share their experiences and explain how it has positively affected their previous projects. Convincing all important parties the benefits of agile is critical to having success with the methodology. If anyone is not completely aboard, I would then weigh moving this person to another project or letting them go from the company completely. They are just not worth the hassle or negative consequences from their attitude. Once the pigs and chickens are convinced, I would move to stage two.

The second stage is investing in training for the non-agile members. Assuming I have the funds, I would require a two-to-three-day training session on agile. They will learn about standup meetings, paired programming, sprints, stories, and most importantly, the importance of teamwork and building a spirit of collaboration which is key to agile. I would also have these members read *The Art of Agile Development* by James Shore and Shane Warden. Once training is complete, we'll move onto the next important stage of merging our agile and waterfall teams.

The third stage consists of making the practical changes needed to implement agile. First, I'll assign the roles of project architect, user interface designer, programmers, testers and

scrum master. These roles would likely be filled by the agile team members for this project or until it is appropriate to name one of the newly-trained agile members. Each previously waterfall team member will be assigned an agile team member as a partner so they can work in pairs (if their work permits it.) Second, I'll find an open space with desks designed for paired programming for the team to work in so they can quickly be available and access one another. Finally, they'll build a new roadmap to successfully creating their product. They'll use physical story cards and progress charts since having a tangible view of everything helps the newcomers learn agile better. All of these changes will be slow, but make a difference in the long run.

The final phase of merging the waterfall and agile teams is being patient. As the saying goes, "Rome wasn't built in a day" and neither will the process of using agile. Agile has a learning curve that takes time to ramp up to optimal efficiency. The stand-up meetings will take some getting used to. The sprints will take time to understand, learn, and develop a good workflow. Using new and open spaces will be unfamiliar for all the newcomers. Changing old routines takes time. This is why it is so important for upper management to be patient. The results may come immediately in the form of having a minimum viable product, but full functionality may not come as quickly as management would like and they will have to be ok with that. All great things take time and agile is no different.

As the person in charge of merging both a waterfall and agile team, I would use my four-stage plan of building buy-in, investing in training, making practical changes, and being patient. These four stages will, in my opinion, produce the best result possible. Overall, we may fall behind in the short-term, but will ultimately come out ahead in the long-term.

3. Discuss the topics of Responsibility and Mindfulness. These are important traits in any software development process. What makes them so important for Agile & Scrum? How do they differ from trust discussed in the first essay question?

There are many important individual and team traits that go into any software development process. Two of the most important ones are responsibility and mindfulness. I will do a deep dive into both of these traits, talk about what makes them so important to agile and scrum, and discuss how they are different from the trust that is described in the first essay question.

Responsibility is taking accountability for every choice - good or bad - we make. If you make a mistake, own up to it. If you have a good idea, share, and follow-through with it. This is a concept not just important in agile or extreme programming (XP) but life. One of the biggest problems of my generation is a fear of admitting your not perfect and that you make mistakes. Whenever someone makes a mistake, many times they go into "blaming mode." I made this mistake because of reason *X* or reason *Y*. It's an attitude "it's not my fault but someone else's fault." In agile and life, you *will* make mistakes. A company doesn't hire you to be perfect. They hire you for your character, basic skills and work ethic. They know you will have bugs in your codes. They know you will do an imperfect manner of implementing a feature into a program. The question they want to know is whether you will always be improving. If you make a mistake, be willing to admit it and go back and fix it. Your company and teammates can accept that. What they cannot accept is a lack of responsibility. It weakens team morale and the success of a project due to time and energy being wasted on you failing to accept responsibility. Because of these reasons, this trait is critical to the team and project's success.

Another trait critical to success is mindfulness. Mindfulness is every team members' need to pay attention to the process and practices of development. Each team member must be considerate of the code quality, feedback from the product owner and coworkers, and the morale of each team member. If the code has a certain pattern and/or certain requirements, make sure to follow these standards. For example, if everyone is using camelcase ("thisIsAVariable") instead of underscores, then maintain the standard and use camelcase. Whenever someone provides feedback, don't take it personally. Your product owner and teammates have the project and your best interest at heart. If they would like you to refactor code, don't get frustrated. Accept their position (unless you have a reasonable objection) and refactor the code. It is better for your relations between the team and the overall morale of the workplace. Finally, be sensitive to your team member's moods and energy levels. If someone is tired, that may not be the best time to push them to do more of something. If someone seems to feel underappreciated, give an honest compliment of something they did recently. Unless they are a complete zero to the team, each team member has added some value to the process in the past couple weeks. Also, having good manners matter. Saying "please" and "thank you" goes a long way for people and the process of software development.

The agile or scrum process is built upon teams effectively collaborating on projects. Without responsibility and mindfulness, the team doesn't operate as well and therefore the project is less successful. It is like riding a bicycle with flat tires. You can ride it to your

destination, but it will take you a lot longer to get there and you will be miserable and exhausted by the time you do.

If lacking responsibility and mindfulness is like riding a bicycle with flat tires, then lacking trust is like riding a bicycle without a chain. The whole bicycle stops in place and it doesn't work. According to Patrick Lencioni's *The Five Dysfunctions of a Team*, trust is "the confidence among team members that their peers' intentions are good, and that there is no reason to be protective or careful around the group." Agile is team dependent and trust is the foundational piece of a team. If you don't trust your team, then petty disagreements distract you from the main goal, time and energy is wasted on watching your behavior, and you fail to provide constructive feedback. It leads to a lack of conflict which leads to a lack of commitment which leads to a lack of accountability and finally a lack of attention to detail. The wheels of the team stop in place.

Responsibility and mindfulness are key ingredients for any successful agile project. Responsibility means taking ownership of every good or bad decision you make and making up for any mistakes you made. This is ok because you are hired to always be improving and not to be perfect. Mindfulness is being considerate of the process and practices of development within the project and team. Each team member must be mindful of the standards of the team's code quality and process. They also must be sensitive to other teammates emotions so as when to press the right buttons and move the project forward. Responsibility and mindfulness are like air in the tires of a bicycle and trust is the chain. You can eventually arrive at your destination without responsibility and mindfulness, but you cannot arrive without trust. Trust is the foundational trait of any team and since agile is a team-dependent process, it doesn't work without it.