Chapter 2
The Psychology of Agile Team Leadership

For modern managers, one has to adopt a new philosophy, or psychology, for dealing with agile development teams. While process is important to ensure the team delivers quality software that meets customer requirements, it is important to understand that the Agile Method is geared around more of an informal approach to management, while putting more time, effort, and emphasis on flexibility, communication, and transparency between team members and between the team and management. It promotes an environment of less control by managers and more facilitation by managers. The role of the manager takes on a new psychological role, one of removing roadblocks, encouraging openness and communication, and keeping track of the change-driven environment to ensure that the overall product meets in goals and requirements, while not putting too much control on the ebb and flow of the agile development process. Change is no longer wrong, the lack of ability to change is now wrong. Here we discuss the new "soft" people skills required for modern managers, and how they add/detract from modern agile development. How to recognize the skills, how to utilize the skills, and how to build teams with the right "mix" of personalities and soft people skills for effective and efficient development efforts [71].

2.1 Individuals over Process and Tools

Companies have spent decades designing, creating, implementing, and executing tools required to bid and manage development projects. One major category of tools is prediction tools like *CiteSeer*© and COCOMO© (Constructive Cost Model) that have been used since the late 1900s to provide "objective" cost bids for software development. A later version of COCOMO, COSYSMO© (Constructive Systems Engineering Model), attempts to provide objective systems engineering bids also. All of them are based on the antiquated notion of Software Lines of Code (SLOC). Productivity metrics are all based on the lines of code written/unit time. They try to estimate the life-cycle cost of software, including designing, coding, testing,

J.A. Crowder, S. Friess, *Agile Project Management: Managing for Success*,
DOI 10.1007/978-3-319-09018-4_2

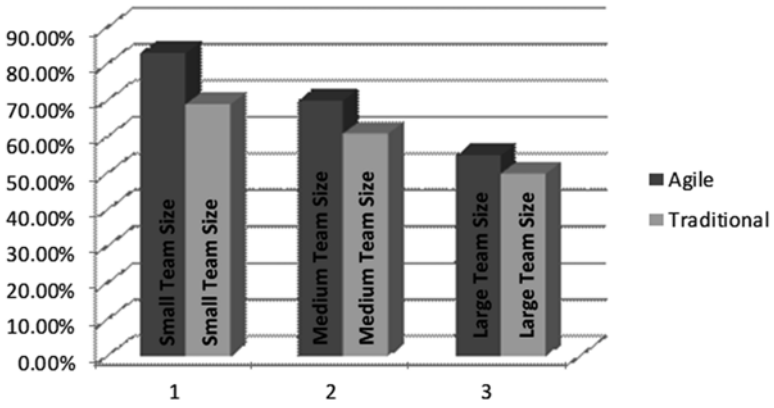

Fig. 2.1 Efficiencies between traditional and agile development

bug-fixes, and maintenance of the software. But ultimately it comes down to Software Lines of Code/Month (SLOC/Month). While many will claim these are objective tools for helping to determine the staff loading necessary for a software/systems development project. In each tool there are dozens of parameters which are input by the operator, each of which has an effect on the outcome of the cost model. Parameters like efficiency (average SLOC/Month), familiarity with the software language used, average experience level, etc., can be manipulated, and usually are, to arrive at the answer that was determined before the prediction tool was used [43].

Many other tools are utilized to measure the performance (cost and schedule) of projects once they are in execution. These measurement tools measure how the project is progressing against its preestablished cost and schedule profile, determined in the planning phase of the program/project. What none of these tools, cost estimation, performance metrics tools, etc., take into account is the actual agile team and their dynamics. The makeup of the each agile team and the facilitation of each team is as important, if not more important, than the initial planning of the project. If the Agile Manager/Leader is not cognizant of the skills necessary not to just write code, but to work cohesively as an agile team, then success is as random as how the teams were chosen (usually by who is available at the time). Grabbing the available software engineers, throwing them randomly into teams, and sending them off to do good agile things will usually result in abject failure of the project, or at least seriously reduced efficiency. This may sound like an extreme example, but you would be surprised how many agile development projects are staffed in just this fashion. Many managers point to the following graph (Fig. 2.1) as the reasons not to go to the expense of changing all their processes to accommodate agile development.

While in each category agile development produces a higher efficiency than traditional software development methods, the increase is not as dramatic as the promises made by agile advocates and zealots. Classical managers find this graph disturbing and feel smugly justified in their classical software development/execution/control methods. This is especially true for large teams. The data for this graph was taken from

Fig. 2.2 Four main components of the agile development process

50 of each size project, both agile and traditional. What are not taken into illustrated by this graph are the management methods utilized across the traditional vs. agile programs/projects: the team makeup, how the teams were chosen, or any discussion of the types of issues that were encountered during the development process. And while it's clear that under any team size agile development has increased efficiency over traditional methods, and, as expected, smaller team sizes produce better results with agile methods, understanding the true nature of the agile team process and applying the psychology of agile management can achieve even greater efficiencies.

Placing the emphasis on the individuals in the agile development teams rather than on process or tools means understanding people, recognizing their strengths (not only in terms of programming skills, but also in terms of soft people skills), and understanding the differences between people of different backgrounds and how the differences affect team dynamics. This is the first generation where it is possible to have 60-year-old software engineers in the same agile development teams with software engineers in their early 20s. The generational differences in perspectives can severely hamper team dynamics, and therefore team efficiencies will suffer greatly if they are not dealt with appropriately and the team members are not trained in how to function in an agile development team. All members of the teams need to be able to understand and come to grips with four main components of agile development, illustrated below in Fig. 2.2. While there are other components that are important,

without a good handle and agreement on these, agile development teams are in trouble from the start. These and other issues relating to team dynamics will be explored in Chap. 3.

As explained, Fig. 2.2 represents four of the major components of the Agile Development Process that must be embraced by the agile development team in order to have a successful and efficient development process. As important are the skills, or philosophies, that the manager of the program/project must embrace and practice in order for the teams to be able to function in an agile environment and have the best chance for success. Figure 1.4 provided a high-level look at the skills of the effective agile manager/leader. The descriptions of these skills are:

1. **Effective Communicator**: The effective communicator fosters and increases trust, is transparent, considers cultural differences, is able to be flexible in delivery of communications, encourages autonomy and role models, exudes confidence to solve problems and handle whatever comes up, and has the courage to admit when they are not sure and willingness to find out. They are willing to work side by side versus competitive with followers. They have the ability to communicate clear professional identity and integrity, their values are clear, and so are their expectations. The effective communicator communicates congruence with values and goals, as well as being a role model of ethical and culturally sensitive behavior and values.
2. **Diplomat**: The diplomat considers the impacts on all stakeholders and how to follow up with all those affected, even if it is delegated. There is willingness to consult cultural experts.
3. **Effective Listener**: The effective listener checks that they understand the meaning being portrayed, and goes with an idea even if they disagree until the whole idea is expressed and the originator can think through the complete thoughts with the leader.
4. **Analytical Thinker**: The analytical thinker must be able to see the forest and the trees. The analytical thinking manager/leader must be able to anticipate outcomes and problems, and explore how they might anticipate handling them, walking through possible solutions. They must initiate Professional Development of team members. They think about the how, not just the what-ifs.

2.2 The Agile Manager: Establishing Agile Goals

For the effective agile project/program manager, it is crucial early on to establish goals and objectives that establish the atmosphere for each sprint development team. Understanding how much independence each developer is allowed, how much interdependence each team member and each team should expect, and creating an environment that supports the agile development style will provide your teams with the best chance for success. Below is a list of agile team characteristics and constraints that must be defined in order for the teams to establish a business or development

"rhythm" throughout the agile development cycle for the program/project. Each will be explained in detail in its own section, but general definitions are given below:

1. **Define and Create Independence**: Independence is something many developers crave. In order for agile development to be successful, there must be a large degree of independence and need to feel an atmosphere of empowerment, where the developers are free to create and code the capabilities laid out during the planning phase of each sprint. This requires a level of trust. Trust that the developers and the leader all have stakeholders in mind. Trust that the developer is working toward the end product [23]. Empowerment at the organizational level provides structure and clear expectations [17]. At the individual level it allows for creativity. Independence means having a voice and yet operating under company structure of policies and procedures. Independence is also a sense of knowing that the developer is good and what they do. There is no need to check in too frequently with the leader, but enough to keep the teamwork cohesive.
2. **Define and Create Interdependence**: While independence is a desired and necessary atmosphere for agile teams, the agile manager must also establish the boundaries where individual developers, and development teams, must be interdependent on each other, given that the goal is to create an integrated, whole system, not just independent parts. Interdependence is being able to rely on team members [16]. The end goal will require a level of commitment from each person with a common mission in mind. The trust that all individuals on the team have all stakeholders in mind. This gets the whole team to the common goal and reduces each member motivated solely for their own end goal.
3. **Establish Overall, Individual, and Team Goals and Objectives**: setting the project/program overall goals, team goals, and individual goals and objectives up front and at the beginning of each sprint helps each team and individual team member to work success at all levels of the program/project. This can help to identify strengths of individuals so that the team can use its assets to their highest production. This also allows room for individual development and growth along with a place for passions. This also sets up clear expectations, say, of the overall and team goals. There may be some individual development that is between the leader and the developer that stays between them. This would also build individual trust between members of the team and between the leader and the developers.
4. **Establish Self-Organization Concepts**: self-organizing teams is one of the holy grails of agile development teams. However, self-organization is sometimes a myth, mostly because teams are not trained into how to self-organize. People do not just inherently self-organize well. If not trained, the stronger personalities will always run the teams, whether they are the best candidates or not [24]. Self-organization can be nearly impossible when there are very structured people coupled with not-so-structured people. There may be some work that the leader can do to promote self-organization. Part of that is opening communication, building dyads, calling behavior what it is, and being transparent so that others will follow. It may be helpful for team members to get to know strengths of other members and how each member can be helpful to each individual.

5. **Establish Feedback and Collaboration Timelines and Objectives**: given the loose structure and nature of agile development, feedback early and often is crucial to allowing the teams to adapt to changing requirements or development environments. Also, customer collaboration and feedback at each level in the development allows the teams to adjust and vector their development efforts, requirements, etc., to match customer expectations at all points in the development cycle. Feedback timelines can increase trust and clarify all expectations. It is nice to know when you need to change a direction, when you need to change it, instead of later when you had already put so much work into the project. The more feedback is modeled and practiced, the more natural it becomes and becomes more automatic. This builds on the independence and interdependence of the team and individual stakeholders.
6. **Establish Stable Sprint Team Membership**: choosing the right teams is important for success in an agile development program/project. Creating teams that are not volatile (changing members often) is essential to continued success across multiple sprints. If the teams constantly have to integrate new members, efficiency will suffer greatly. New expectations and explanations will take up much time that could be used for developing. A trusting team can be an efficient team. The more often it changes the more work needs to be done to build the trust. There may be increased commitment from those that work on a cohesive team with high trust levels and knowledge of one another [18].
7. **Establish Team's Ability to Challenge and Question Sprints**: if the teams are going to be allowed individual and team empowerment, then they must be allowed to challenge and question sprint capabilities and content across the development cycle. Forcing solutions on the teams fosters resentment and a lack of commitment to the program/project. If you've built the right team, you should listen to them. It seems more productive to work on something that makes sense to you, instead of handed down by others. The ability to challenge and question will lead to better understanding and more commitment to the end goal.
8. **Establish an Environment of Mentoring, Learning, and Creativity**: invariably, teams are composed of a combination of experience levels. This provides an excellent atmosphere of mentoring and learning, if the agile manager allows this. This must be built into the sprint schedules, understanding that an atmosphere of mentoring, learning, and creativity will increase efficiencies as the team progresses, not just on this project, but on future projects as well, as the team members learn from each other. Keep in mind that experienced developers can learn from junior developer too, as the more junior developer may have learned techniques and skills that were not previously available to more senior developers. The learning environment promotes growth. An environment that fosters learning decreases negative feelings of one's self, and thus other people. An environment that fosters learning isn't run by guilt, or feelings of not being good enough, or doing something wrong. A learning environment allows people to grow and the mentor helps the individuals self-determine the direction they want to develop. The learning environment will foster older members learning from younger members as well. People will want to learn more and more and

reduce competitiveness that can destroy a team. The competitiveness can come out as a good product not team dynamics. Transparency can help individuals feel more comfortable with learning. This can show that is ok to have areas of development and that everyone has room to grow.

9. **Keep Mission Vision always out in Front of Teams**: many believe that an established architecture is not required for agile development. This is absolutely wrong; a solid architecture is even more important during agile development, so each team and team member understands the end goals for the system. However, in order for the architecture and software to stay in sync, the systems engineering must also be agile enough to change as the system is redesigned (or adapted) over time [19]. Agility is not free from structure but the ability to move about within the structure.

2.3 Independence and Interdependence: Locus of Empowerment

Locus of Empowerment has been conceptualized as a function of informed choice and self-determination and has been linked to the concepts of self-efficacy and locus of control as it applies to agile team membership [6]. Self-understanding and empowerment, in relation to development opportunities and factual strength/weakness assessment, represents an important underlying component of feelings of self-empowerment within an agile development team [74]. Locus of empowerment and its counterpart, Locus of Control, help to establish both independence and interdependence for agile team members. Determining those things each team member is "empowered" to make decision on and work independently provides each person with a sense of autonomy, allowing them to work at their peak efficiency without interference or too much oversight control over their work. Establishing the Interdependence, or those things which are outside of the control of the team member, defines communication lines and those things which are necessary to collaborate on, or get inputs from other team members to facilitate integration and validation of "system-wide" capabilities [7]. What follows is a discussion of Locus of Empowerment. Locus of Control will be discussed in Sect. 2.4.

2.3.1 Locus of Empowerment

The notion of Locus of Empowerment is an interactive process that involves an individual team member's interaction with the team and the manager [70], allowing each team member to develop a sense of acceptance into the team, develop a sense of where they belong in the team, self-assessment of skills, and determination of their self-efficacy—their ability to function and participate both on an individual level and as part of an agile development team [49]. These allow each

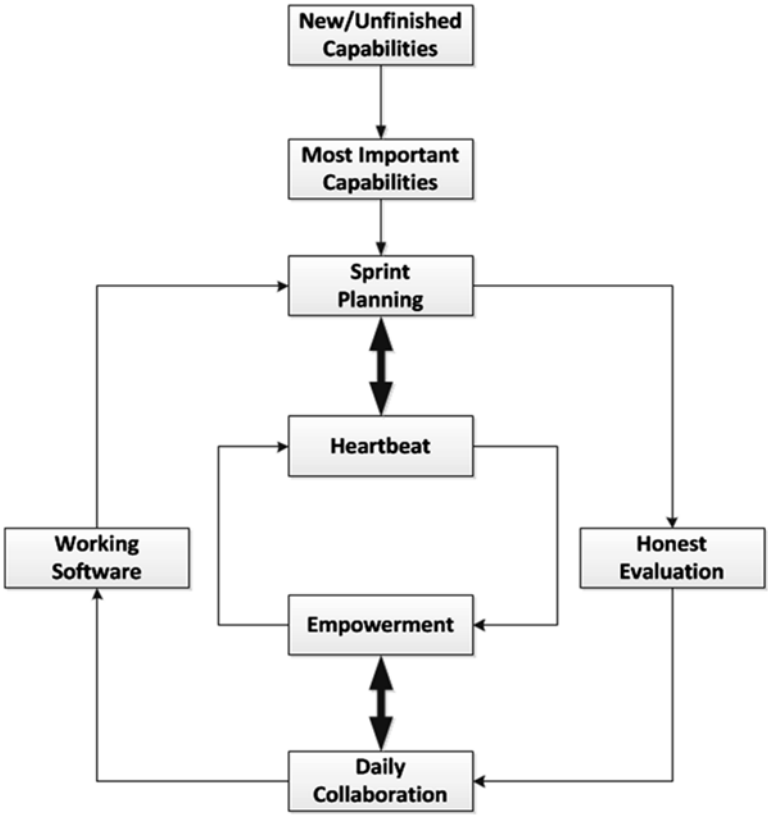

Fig. 2.3 The agile development process with empowerment

individual team member to participate with others, based on their understanding of their independence and interdependence from and to the team, allowing them to deal with the daily, weekly, monthly, etc., rhythms of the agile development cycles throughout the program/project [53].

The process of team and team member empowerment is a continual and active process; the form and efficacy of the empowerment process is determined by past, current, and ongoing circumstances and events [69]. In essence, the empowerment process is an ebb and flow of independence and interdependence relationships that change throughout the agile development process, including each daily Scrum, each Sprint planning session, and each Lessons Learned session, throughout the entire agile development cycle of the program/project. Figure 2.3 illustrates this process.

In Fig. 2.3, empowerment becomes an integral part of the overall agile development process, with evaluation of the team members' abilities, roles, independence, and interdependence, based on the capabilities needed to be developed within a given Sprint, the honest evaluation of skills and abilities; i.e., how to develop the heartbeat, or development rhythm required for each development Sprint. Without an environment of Empowerment, the team has no real focus, since each team member

does not have a sense of what they are individually responsible for, what the other team members are individually responsible for, and what communication is required throughout the Sprint development [68]. There will eventually be a breakdown of the team, a loss of efficiency, and the team will not be successful in their development efforts within cost and schedule constraints. Next we discuss the concepts of goal setting for an agile development project: project/program, team, and individual goals within the context of Locus of Control.

2.4 Overall, Individual, and Team Goals: Locus of Control

As explained above, the very nature of agile software development is to create a loose structure both within each Sprint team and across the Sprint team structure. The purpose of agile development is to allow developers, and subsequently the system being developed to adapt and change as requirements, features, capabilities, and/or development environment change over time (and they will change). However, this does not mean that there are not system-level, team-level, and individual-level goals at each point in time. In fact, it is more important in agile development to have well-defined goals as teams and individual developers write and test software, to ensure the software integrates and, more importantly, creates a set of capabilities and a system the customer wanted and is paying for. Customer and cross-team collaboration and feedback at each level is crucial to allow the teams to adjust, either from customer needs or inter-team needs across the agile Sprint developments. Again, independence and interdependence is essential for overall successful development. Further refinement of the Empowerment concept is to define, for each individual developer, what things are within their own control, and those things are outside of their control, even if they affect the individual.

This notion of internal vs. external control is called "Locus of Control." Locus of control refers to the extent to which individuals believe that they can control events that affect them [63]. Individuals with a high internal locus of control believe that events result primarily from their own behavior and actions. Those with a high external locus of control believe that powerful others, fate, or chance primarily determine events (in this case other team members, other teams, the program/project manager, and/or the customer). Those with a high internal locus of control have better control of their behavior, tend to exhibit better interactive behaviors, and are more likely to attempt to influence other people than those with a high external locus of control; they are more likely to assume that their efforts will be successful [12]. They are more active in seeking information and knowledge concerning their situation.

Locus of control is an individual's belief system regarding the causes of his or her experiences and the factors to which that person attributes success of failure. It can be assessed with the Rotter Internal–External Locus of Control Scale (see Fig. 2.4) [63]. Think about humans, and how each person experiences an event. Each person will see reality differently and uniquely. There is also the notion of how one interprets not just their local reality, but also the world reality [79]. This world reality may be based on fact or impression.

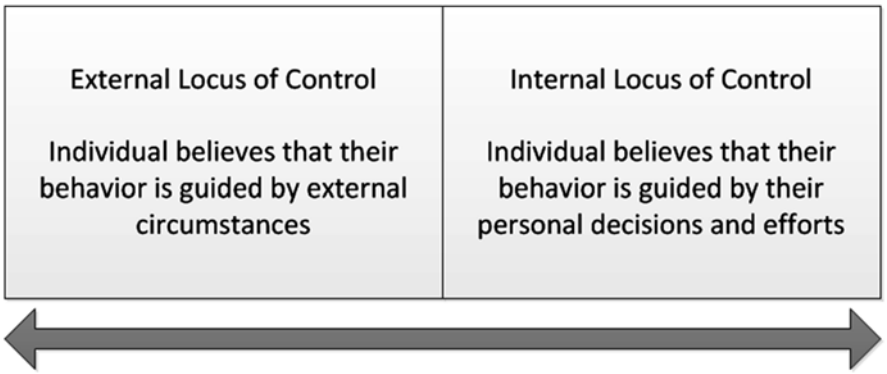

Fig. 2.4 The locus of control scale

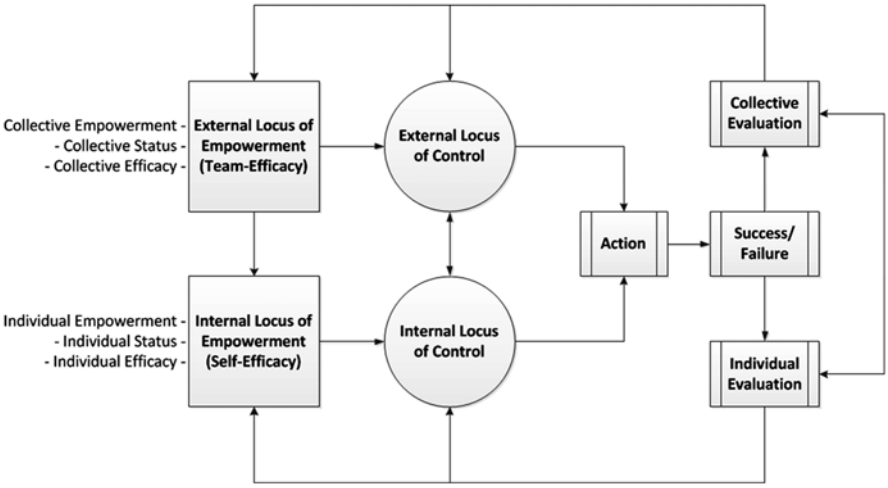

Fig. 2.5 Locus of control within an empowerment cycle

For further thought let's then consider Constructivist Psychology. According to "The internet Encyclopedia of Personal Construct Psychology" the Constructivist philosophy is interested more in the people's construction of the world than they are in evaluating the extent to which such constructions are "true" in representing a presumable external reality. It makes sense to look at this in the form of legitimacies. What is true is factually legitimate and what is people's construction of the external reality is another form of legitimacy. In order to have an efficient, successful agile development team [62], each member must understand and accept their internal and external level of Locus of Control, as well as their Locus of Empowerment level. Figure 2.5 illustrates how this flows throughout the Sprint development cycles.

How an individual sees the external vs. internal empowerment drives their view of internal vs. external Locus of Control. During each development cycle, evaluations are made (whether the individual is aware of it or not) as to their internal and external Empowerment, and subsequent Locus of Control. Actions are determined, based on this self-assessment, and self-efficacy determination. Based on the results of their efforts, individuals, as well as the team, and the entire program/project reevaluate the efficacy of the levels of internal vs. external Empowerment that are allowed, and adjustments are made. These adjustments to Empowerment levels drive changes in Locus of Control perception, which drives further actions. This process is repeated throughout the project/program. The manager must understand this process and make the necessary adjustment so that each individual can operate at their peak self-efficacy, as well as support team efficacy, providing the best atmosphere for successful development.

2.5 Self-Organization: The Myths and the Realities

One of the holy grails of agile development is self-organizing teams. Many software developers dream of having a team with complete autonomy, able to organize however works for them, completely without management involvement or interference. However, what most developers fail to realize is that given to their own devices, without training as to how to organize and what "organizing" actually means, most would fail miserably. Often, agile development efforts fail, even with efforts to educate the team about agile principles [53]. That is because the team doesn't fail because they don't understand agile software development. It's because they don't understand human nature and the difficulties in taking a team of highly motivated, strong personalities, and get them to automatically give up their egos, preconceived notions, and past experiences, and embrace the agile team dynamics required to put together a highly successful agile development effort. We call this "Agile Team Dysfunctionality," and there are many common dysfunctions that plague improperly trained teams and team members. Figure 2.6 illustrates several of the most serious dysfunctions, most of which come both from basic human nature and from people's experience with work on programs/projects in the past. Nothing drives failure of agile development like past failures. Remember Figs. 1.1 and 1.2. Teams that have experienced Fig. 1.2 are hard pressed to throw off their suspicions and embrace agile development processes, team dynamics, and the entire agile agenda fresh. Management must be cognizant of these dysfunctions and work within the teams to dispel them.

Inability to recognize or deal with agile team dysfunctions can destabilize the team(s) and derail the agile development process faster than anything else. Keeping a stable set of Sprints teams is important, as constantly changing out team members radically changes team dynamics, and affects both personal and team Empowerment and Locus of Control [58]. Section 2.6 discusses the concept of stable team membership.

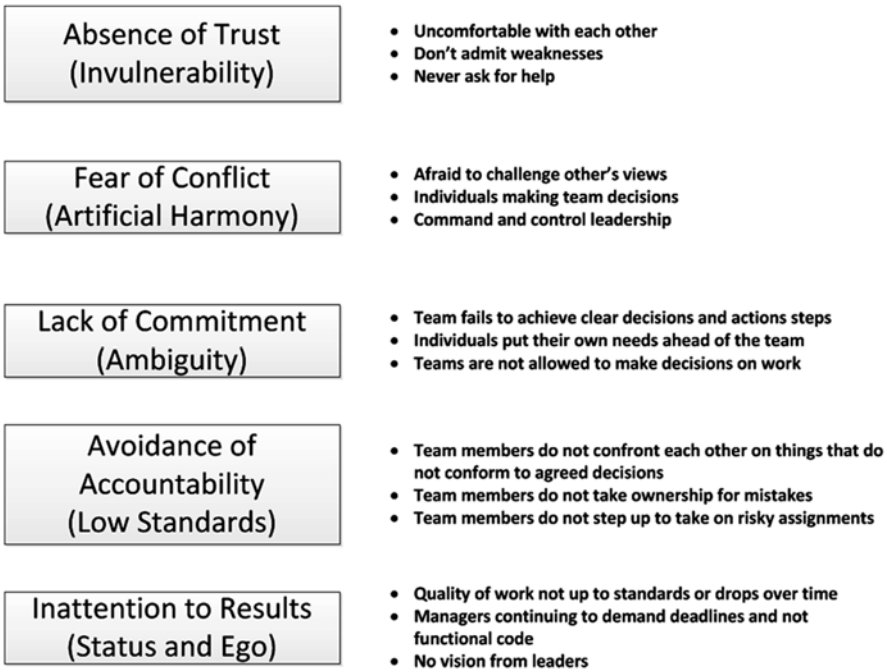

Fig. 2.6 Common agile team dysfunctions

2.6 Creating a Stable Team Membership: Containing Entropy

As previously discussed, it is vital to choose the right teams for any program/project, but it is even more important for agile development. Teams with stable memberships across Sprints are vital, as team members develop trust over time, gain an understanding of each member's strengths and idiosyncrasies, and, with proper training, mentorship, and facilitation by the manager, settle into an agile development "rhythm" throughout the program/project. If the team has to integrate new members, efficiency will always suffer until the new team member is properly integrated into the rhythm. New expectations are created; the new person will most likely have an entirely different notion of Empowerment and Locus of Control than the previous team member, throwing the overall team out of balance. A stable team can be a trusting and efficient team [56]. There is generally an increase in commitment over time with a stable team [52]. In order to facilitate creation of stable agile sprint teams, the Agile Manager must recognize, understand, and know how to deal with the dysfunctionalities discussed in Sect. 2.2. For each dysfunction, the Agile Manager must take on a role, or provide guidance that dispels the dysfunction and allows the team to move toward and independent cohesiveness between the team members [60]. Figure 2.7 illustrates the Agile Manger's role in dealing with classical agile team

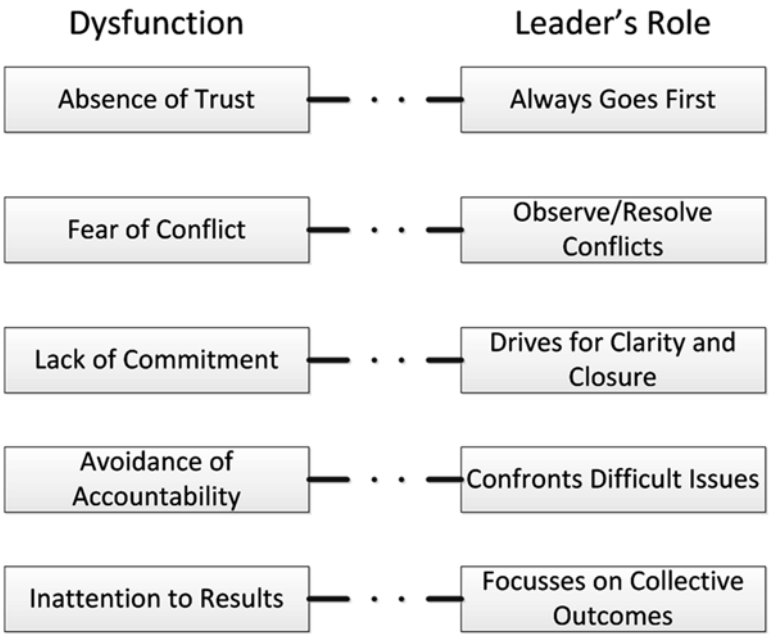

Fig. 2.7 The Agile Manager's response to team dysfunctions

dysfunctions, creating a team that works together, in Empowered independence and dependence, to develop software in an efficient agile environment.

As depicted in Fig. 2.7, for each of the agile team dysfunctions described in Fig. 2.6, Fig. 2.7 illustrates the Agile Manager's response required to eliminate the dysfunction and allow the agile development teams to function effectively and efficiently:

1. **Absence of Trust**: In order to build trust within the teams, the Agile Manager must always be willing to take the lead and prove to the team members that they will "roll up their sleeves" and do whatever is necessary to either get the program/project moving or to keep it moving along.
2. **Fear of Conflict**: Many developers are fearful of bringing up issues, not wanting to start controversy within the team. Many people, particularly strong introverts, may internalize the conflict, never bringing it up, but eventually the conflict will drive controversy between the developers, create a lack of trust, and may drive the team to withdraw from each other, destroying the collaborative nature of agile development teams. In order to diffuse these situations before they begin, the Agile Manager must be observant and cue in on body language and utilize the soft people skills like paying attention to changes in personal habits, language, friendliness, and other clues apparent between team members, and facial expressions to understand when such nonverbal controversies exist and work to resolve the conflict before they begin to negatively impact the development efforts.

3. **Lack of Commitment**: A lack of commitment to either the agile development team or the agile process in general can destroy an agile program/project before it gets started. Observing a low quality of work, absenteeism, lack of willingness to communicate, or constantly seeming to be overwhelmed by the volume of work may be indications of a lack of commitment. The Agile Manager needs to understand the developer's reasons for the lack of commitment, clarifying for the developer what is expected, clearing up any misconceptions the developer may have. In the end, if the Agile Manager does not feel they have dispelled the lack of commitment, the developer must be removed from the team or there is little hope for successful agile development. I know this sounds harsh, but agile only works if all parties have a buy-in to the agile development process.
4. **Avoidance of Accountability**: There may be issues getting developers to step up and take on rolls of responsibility within the agile teams because they are afraid that if they take responsibility for the team's activities during a given Sprint and there are problems, they will be punished. This lack of accountability needs to be dealt with in order for the Sprint development teams to develop a good business rhythm and operate effectively. It is up to the Agile Manager to confront issues, while not assigning blame or punishment, but working through difficult issues, helping each developer learn from the issues in order to solidify the teams and allow the developers to grow and mature as members of an agile development team. This will pay off in the future as each developer becomes more embedded in the agile process and learns to be effective in and excited about agile programs/projects.
5. **Inattention to Results**: Some developers like the agile team process because they feel they can just write code and let other people worry about the details, results, testing, etc. But, it is vitally important that the entire team focus on the results: working, error-free code with capabilities required for each Sprint that can be demonstrated. If any of the developers/team members are not focused on the results, the team will never develop a good agile development rhythm. Also, one member being inattentive to details and results will breed mistrust between the members, reducing the effectiveness of the team(s). Therefore, the Agile Manger must keep the program/project vision in front of all developers and teams, making sure everyone is marching down the same path, ensuring that the collective outcomes of all the Sprint teams, across all of the Sprints, integrate together and are heading toward a common, customer-focused goal.

2.7 Challenging and Questioning Sprints: Individual Responsibility

Creating a team of highly motivated, capable, and experienced developers that are expected to work in an agile development environment and not allowing them the freedom, or Empowerment, to question and/or challenge Sprint capabilities, planning, sequencing, etc., will destabilize the team quickly. The Agile Manager should

not force solutions on the team, for this fosters resentment and breeds an attitude of lack of commitment to the program/project. If you build and train the teams correctly, they should be able to discuss and come to agreement on how capabilities are spread across Sprints, who is the best choice for what role across each Sprint, and to work together when collaboration is needed. The ability to challenge and question leads team members to a better understanding and more commitment to the end goal, not only for each Sprint, but to the entire program/project as well [54].

2.8 Mentoring, Learning, and Creativity: Creating an Environment of Growth

Agile development teams, at least the majority of teams, will be composed of developers at a variety of experience levels. Each member comes with their own strengths and weaknesses and should be provided an atmosphere that not only allows them to succeed, but to grow and learn, both from the experience of developing code for the program/project across the Sprints, but from each other as well. If facilitated correctly by the Agile Manager, the agile development program/project will allow opportunities for mentoring and learning. However, this must be designed into the Sprints, both in schedule and in capability distribution across the team members. Creating an atmosphere of mentoring, learning, and creativity increases efficiencies, as the team progresses through the Sprints, and helps future programs/projects as well. Given the probable diversity of team members, the Agile Manager should make sure everyone has the opportunity and personal attitude of both mentoring and learning from each other. New software techniques brought by junior developers may be necessary for certain capabilities that older more experienced software developers may not be aware of. At the same time, junior developers should also bring an attitude of mentoring and learning, as the experienced developers can aid junior developers from going down disastrous roads already traveled by senior developers. In short, the atmosphere the Agile Manager must *NOT* bring to the agile development teams is illustrated in Fig. 2.8.

2.9 Keeping the Vision in Front of the Team: Ensuring System Integration

I have heard many developers tell me that the one advantage with agile development is that they are free to do what they want, because it isn't necessary to establish a systems and software architecture for agile programs/projects. Such notions lead to serious problems later in the development cycle. Without a systems and software architecture, integration and final testing of the system is problematic at best and normally results in much rework and recoding to create a complete system [26].

Fig. 2.8 The "Rigid" Agile Manager

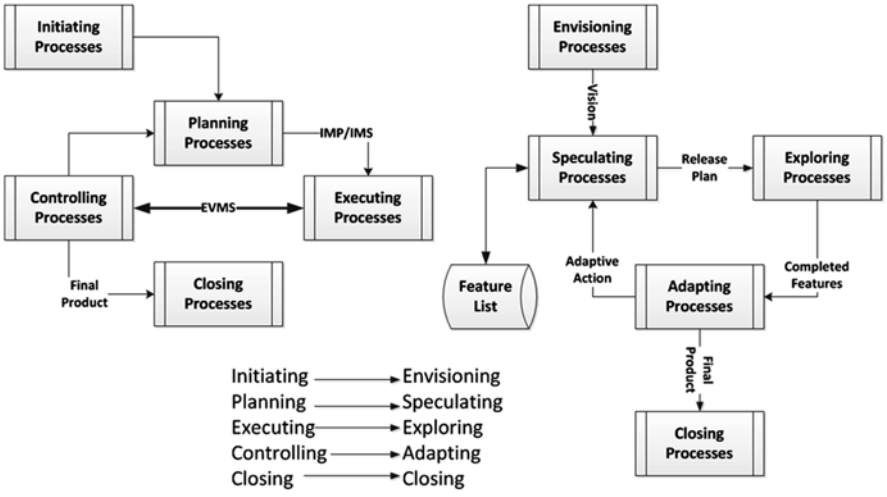

Fig. 2.9 The traditional vs. agile development process

Each team member and each team must understand the end goals for the system. You must remember that agility does not mean there is no structure, but the agile development methodology provides the abilities to move about within a given architecture or structure. Figure 2.9 illustrates the differences between the traditional development process and the agile development process [57].

Every aspect of the classical development cycle has a counterpart for the agile design process, but is designed, or intended, to promote the agile mind-set: one of adaptability to changing requirements or environments. Many are uncomfortable in the agile software development paradigm. Many like the structure of classical software development. So how does one understand who is and is not comfortable with agile development. Can any developer be made to function within the "freedom" of agile? In Chap. 3 we will explore all of the dynamics of agile development teams, how to create, manage, facilitate, and empower agile development teams.

Where the traditional development process involves and is focused on detailed planning, budgeting, controlling, and program/project execution, the agile development process must be adaptive and innovative, deriving solutions to a changing requirements/capabilities baseline, the focus being on working software, not cost and schedule. This is not to say that cost and schedule are not important in agile development, because cost and schedule are always important in any program/project execution. However, the agile development process has much more flexibility to deal with risks or issues that arise than the classical development process, giving the Agile Manager more tools and more opportunities to adjust without major rework in extensive schedules and budgets.