James A. Crowder · Shelli Friess

Agile Project Management: Managing for Success

Springer

Agile Project Management: Managing for Success

James A. Crowder • Shelli Friess

Agile Project Management: Managing for Success

Springer

James A. Crowder
Raytheon
Denver, CO, USA

Shelli Friess
Englewood, CO, USA

ISBN 978-3-319-09017-7 ISBN 978-3-319-09018-4 (eBook)
DOI 10.1007/978-3-319-09018-4
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014945570

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Dr. Crowder has been involved in the research, design, development, implementation, and installation of engineering systems from several thousand dollars up to a few billion dollars. Both Dr. Crowder and Ms. Friess have been involved in raving successes and dismal failures (ok, let's call them learning opportunities) not only in development efforts but in team building and team dynamics as well. Having been involved in agile development projects and team building exercises, both have seen the major pitfalls associated with trying to build teams and, in particular, create successful agile development teams. A general lack of management commitment to the agile development process and a lack of training provided for people working in development teams are two of the major reasons agile teams so often falter or fail. Here we endeavor to discuss some of the major topics associated with team dynamics, individual empowerment, and helping management get comfortable with a new paradigm that is not going away.

Having taught both classical program management methods and agile development and management methods for many years, there are always arguments as to whether the proper term is program management or project management. To settle the matter and not create issues, in the course of this book, we will use the term program/project management. It may seem redundant, but it covers both bases.

There are several case studies throughout the book. These case studies came from a variety of government, aerospace, and commercial companies/groups, and no company should be inferred from a given case study, unless the company name is specifically mentioned. In some instances, the case study may represent a collection of very similar stories from several different companies.

Lastly, we want to emphasize that this is not a book on how to perform agile development, but how to manage the process of agile development and how managers can facilitate successful and efficient agile development programs/project. This book is written to give managers the tools required to be successful as an agile manager.

Denver, CO — James A. Crowder
Englewood, CO — Shelli Friess

Contents

Chapter 1
Introduction: The Agile Manager

Modern productivity teams demand modern leadership, one that understands modern development needs, stresses, teams, and other aspects of agile development team dynamics [22]. The purpose of the book is to introduce managers to the new productivity environments, including geographically, culturally, and generationally diverse teams. The Agile development paradigm embodies a set of principles which at first may seem contrary to classical business practices:

1. Satisfy the customer through early and continuous delivery of software capabilities and services through short software "sprints," lasting from 2 weeks to 2 months, with a preference toward shorter sprints.
2. Embracing the environment of change. The Agile development process harnesses change to gain a competitive advantage in the software development marketplace [28].
3. Business development, management, and developers must cooperate and collaborate throughout the development project.
4. Communication needs to be face-to-face, even if that means teleconferencing over diverse geographical locations. Face-to-face communication is essential for efficiently and effectively conveying necessary information across an agile development team.
5. The Agile development process is designed to allow a sustainable, constant pace development throughout the entire development project.
6. The primary measure of success (Earned Value) is working software and capabilities, *NOT* Equivalent Software Lines of Code (ESLOC).
7. Agile development does *NOT* mean a lack of design. A good design (architecture) enhances agility and allows continuous attention to technical excellence.
8. Simplicity is essential in agile development. The importance of agile software development demands the art of maximizing the work *NOT* done.
9. The best architectures, requirements, and software designs emerge from self-organizing teams, *NOT* from management mandated team structures.

J.A. Crowder, S. Friess, *Agile Project Management: Managing for Success*,
DOI 10.1007/978-3-319-09018-4_1

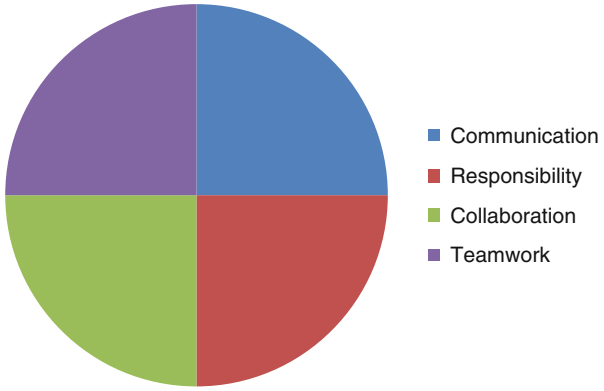

Fig. 1.1 What Agile development projects are supposed to teach you

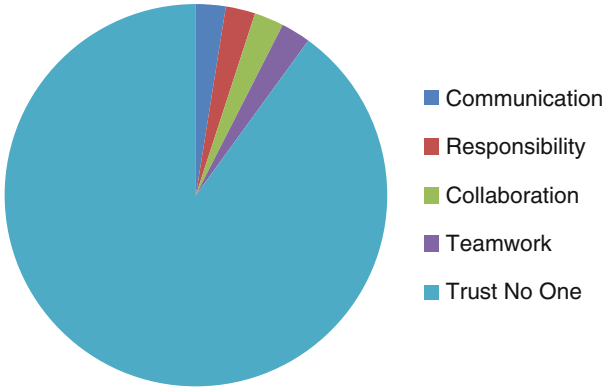

Fig. 1.2 What Agile projects that are managed badly actually teach you

The following illustrates this notion, as well as the results of not following these concepts in managing agile development teams (Figs. 1.1 and 1.2).

1.1 Agile Development Demands Agile Management

Agile software design methods are now commonplace; however, management skills, in general, have not kept pace with the advances in software development practices. There is a major push among companies, both government and commercial, to embrace the concepts of diversity and inclusiveness. Managers need to

be trained in how to manage teams of diverse personnel. Much has been made of managing different personalities, but managers need to be aware of soft people skills, how to manage them, use them effectively, and how they affect people of different backgrounds [29].

It is commonplace to have to work with teams across geographically, ethnically, generationally, and culturally diverse backgrounds within the same team, not to mention a range of skill levels. This book will be helpful in understanding how to manage an agile development team that includes such dynamics [30]. This book will take the project/program manager beyond the concepts of transformational leadership, which provides methodologies to connect to employees' sense of identity, to include human psychological concepts such as "Locus of Control," which will help the manager understand team members' view of how to manage their "world" contributes (enhances or detracts) from their ability to work within team dynamics.

Agile design methods have been utilized since the mid-1990s, and yet program/project managers have been slow to adapt to the changes required for effective agile development [31]. And while there are basic management techniques like Scrum available for the mechanics of managing agile teams, none of these address the dynamics agile development, which include:

- How to choose the right agile development team
- How to facilitate, not control, an agile team
- How to trust your team: trust is an important factor in change [18]

The agile development process demands trust, transparency, accountability, communication, and knowledge sharing [50]. Allowing agile teams to develop these qualities requires understanding and allowing people to evolve and exercise aspects of their internal development processes like Locus of Control. Locus of Control refers to the extent to which individuals believe that they can control events that affect them. Individuals with high 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, or chance, primarily determine events that affect them [62].

The purpose of the book is not to emphasize any particular Agile Development Management style (e.g., Scrum), but instead to investigate and present methods for effective Agile team development and management philosophies. Just because you use Scrum does not mean you are Agile. Scrum is a one of many methods for managing Agile Software Development teams, assuming you have an agile development team. Many management organizations confuse this, with disastrous results. To overcome this and to provide the Agile Manager with the skills required to efficiently manage agile development teams, the book includes discussion agile management techniques [18], the psychology of agile management (Locus of Empowerment [53]), as well as new metrics and methodologies for measuring the efficacies of Agile Development teams (agile EVMS).

1.2 Software and Systems Engineering: Where Did They Come From?

Many think the discipline of Software Engineering is relatively new, and that before the invention of "modern" software techniques, the discipline was nothing more than structured coding. But actually, the first two conferences on Software Engineering were sponsored by the NATO Science Committee in 1968 and 1969 in Garmisch, Germany [58, 74].

1.2.1 Software Engineering History

In 1968 and 1969, the NATO Science Committee sponsored two conferences on software engineering, seen by many as the official start of formal discipline of Software Engineering. The discipline grew, based on what has been deemed the "Software Crisis" of the late 1960s, 1970s, and 1980s, in which very many major software projects ran over budget and over schedule; many even caused loss of life and property. Part of the issues involved in software engineering efforts throughout the 1970s and 1980s is that they emphasized productivity, often at the cost of quality [75]. Fred Brooks, in the Mythical Man Month [14], admits that he made a multi-million dollar mistake by not completing the architecture before beginning software development, a major problem that has been repeated over and over, even today. We will discuss the notion of the importance of having a complete architecture on agile development later in the book. This does not mean that the architecture can't change, as it often does throughout the project, but system's engineering must keep up with changes so that the development teams clearly understand the architecture they are developing to during every sprint [19]. We will discuss this at length in subsequent chapters, as the intent here is just to provide a brief history.

1.2.2 System Engineering History

Systems engineering began its development as a formal discipline much earlier than software engineering, during the 1940s and 1950s at Bell Laboratories. It was further refined and formalized during the 1960s during the NASA Apollo program. Given the aggressive schedule of the Apollo program, NASA recognized that a formal methodology of systems engineering was needed, allowing each subsystem across the Apollo project to be integrated into a whole system, even though it was composed of hundreds of diverse, specialized structures, sub-functions, and components. The discipline of system engineering allows designers to deal with a system that has multiple objectives, and that a balance must be struck between objectives that differ wildly from system to system. System engineering seeks to optimize the

overall system functionality, utilizing weighted objectives and trade-offs in order to achieve overall system compatibility and functionality [19]. During the 1970s and 1980s as engineering systems continued to increase in complexity, it became increasingly difficult to design each new system with a blank page. As system quality attributes like reliability, maintainability, re-usability, availability, etc. became more and more important, the concept of Object-Oriented design techniques was developed. The first Object-Oriented languages began to emerge during the 1970s and 1980s. By the 1990s, the first books on Object-Oriented Analysis and Design (OOAD) were published and available. Unfortunately there were many different OOAD methodologies. There was no consistency among methods. At one point in the 1990s, there were over 50 different OOAD methods. This became increasingly difficult for the Department of Defense (DoD), because contract proposals from competing contractors utilized entirely different OOAD methods to design their systems, making comparison between proposals nearly impossible. Finally, in 1993, the Rational Software Company began the development of a Unified Modeling Language (UML), based on methodologies by Grady Booch [13], James Rumbaugh [64], and Ivar Jacobsen [41], coupled with elements of other methods. Here the Rational Software Company simplified current methods from several authors into a set of OOAD methods that included Class Diagrams, Use Case Diagrams, State Diagrams, Activity Diagrams, Data Flow Diagrams, and many others.

1.3 The Need for New Leadership

While Software and Systems Engineering has matured and evolved over the decades to accommodate a faster-paced, ever-changing development environment, program/ project management still tends to cling to rigid management techniques and principles that critically hamper the agile process [51]. A great example is described below:

Case Study #1: The Non-Agile Manager

Project Length	12 months
Number of Sprints	8 Sprints
Number of Teams	4 Teams
Average Sprint Duration	6 weeks

Description During the planning for Sprint #4, Team #3 discovered that team #4 had a capability scheduled for Sprint #5 that Team #3 needed for their development in Sprint #5 to accommodate their sprint #5 development. Team #3 negotiated with team #4 and they found a set of capabilities that team #4 had scheduled for sprint #4 that were not needed till sprint #6. The capabilities team #3 needed constituted the same number of story-points, and similar complexities, and therefore would not overly tax team #4's sprint work to swap the work between sprint #4 and #5 to accommodate team #3. Team #4 accomplished all of their development for sprint #4. During the progress evaluation with the program manager, the manager was very

upset that the planned work had not been accomplished, but it had been changed, with work moved from Sprint #4 to Sprint #5. Team #4 explained that the work represented the same number of story-points and similar complexity, therefore not perturbing the overall cost and schedule of the program. When the manager pushed back, still upset that the work scheduled had not been accomplished, both Team #3 and Team #4 explained that this is a classical part of the Agile Development Process, being flexible to move capability development around, based on changing needs and requirements. The manager's reply was (and hence the reason for the book), "Then I guess we need a more *rigid* agile process."

1.3.1 Agile Management: Leader, Manager, Facilitator

Many mangers shudder at the thought of agile development projects, feeling like their authority has been eroded. I have heard more than one project manager declare, when the agile methodology is explained, ask, "So what am I going to do?" Many managers are used to being intimately involved in the development process, even though they may, or may not, actually have been software developers [34]. It is true that project/program managers must learn to adapt and take on different roles in the world of agile development, becoming facilitators and leaders and not so much traditional managers. The notion of classical line management, or boss, which many managers still cling to, is no longer relevant in the paradigm of agile software development projects. Figure 1.3 illustrates this, albeit a bit dramatically.

For effective management of agile development projects, the manager (we'll refer to the manager as the Agile Manager throughout the book) needs to have many skills [33], including effective communication, a diplomat, and other skills shown in Fig. 1.4, and will be explained in detail in Chap. 2.

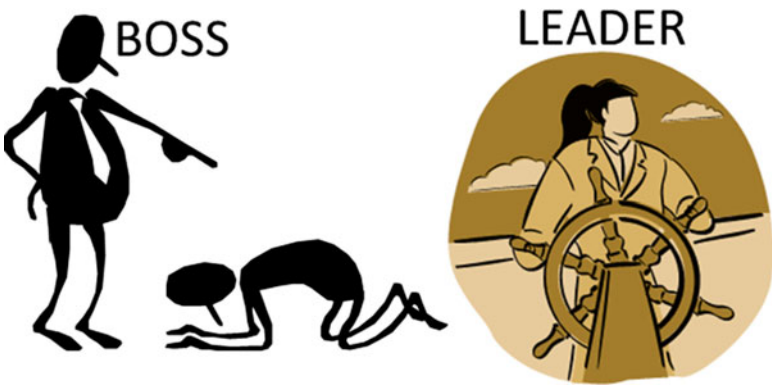

Fig. 1.3 Boss vs. Leader

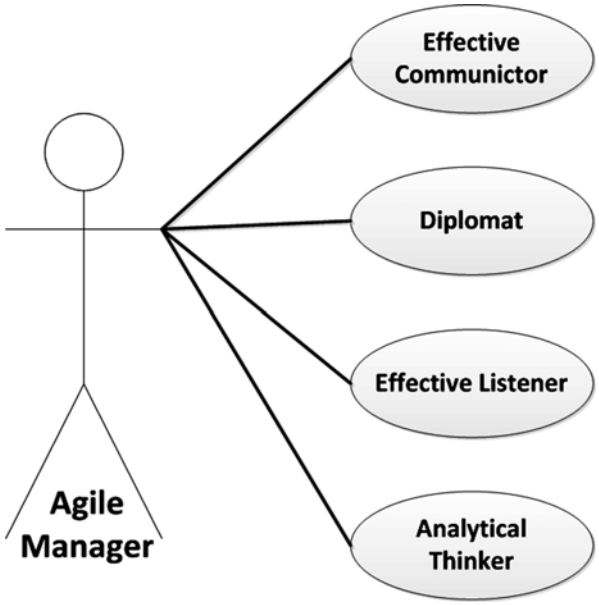

Fig. 1.4 Skills of the Agile Manager/Leader

The rest of the book is dedicated to an in-depth discussion of the new management paradigm required to enable, encourage, and fully embrace Agile Software Development and make them the successes they can be. To accomplish this, the structure of the book is outlined in the next section.

1.4 Layout of the Book

We have arranged the book to build up to new methods for Agile Software Program/Project management.

Chapter 2—The Psychology of Agile Team Leadership This describes 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.

Chapter 3—Understanding the Agile Team Success in the modern development era needs managers and leaders who truly understand what agile development means and how agile teams collaborate, cooperate, and function in various situations, particularly in geographically and culturally diverse environments. Understanding the variety of personalities and soft people skills, coupled with how these manifest themselves across gender, ethnic backgrounds, and cultural and generational diversities, and other considerations, will become essential for modern development leadership

and management. This includes discussion of overall inclusiveness and diversity within the agile development process. Diversity and Inclusiveness are important dynamics that companies are embracing. Building development teams that are not just effective but embrace the concepts of diversity and inclusiveness are important [35], but most leaders and managers have not been trained for the dynamics these bring (both good and bad) to teams.

Chapter 4—Productivity Tools for the Modern Team Providing an agile development team with tools to be productive goes beyond handing each one of them a laptop with compilers. Communication and collaboration tools, whether face-to-face or geographically diverse, are crucial in modern teams. Here we discuss collaboration tools and other tools that will be crucial today and in the future. The proper use of Information Systems can provide management and leadership with effective ways of monitoring and managing teams. Here we discuss the new management information systems environments and tools that are available, will be available in the future, and need to be utilized within the new agile development paradigm.

Chapter 5—Measuring Success in an Agile World: Agile EVMS The Earned Value Measurement System (EVMS) has become a mainstay in Commercial and Government groups to measure progress and success of a project. EVMS is effective (albeit subjective) measure, but does not play well with agile development efforts, due to its requirement of static schedules and work plans. Here we introduce a new paradigm for EVMS that will accommodate and be effective in measuring progress and problems within agile development efforts.

Chapter 6—Conclusion: Modern Design Methodologies One of the things that need to be understood by leadership and management in the future is that just because you deliver a product on time and on budget doesn't mean the project was an overall success. Delivering a product on budget and on schedule but decimating a development team is not, in the long run, a success for the company. Managers and Leaders must understand all aspects of development teams for long-term success.

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

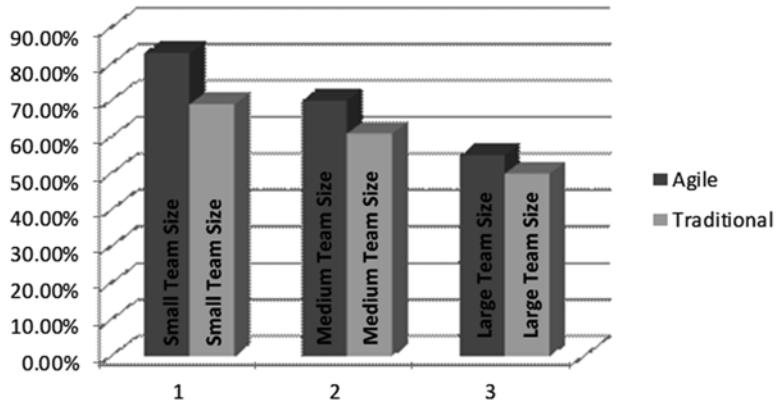

Fig. 2.1 Efficiencies between traditional and agile development

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