Chapter 8
Disciplined Teams vs. Agile Teams:
Differences and Similarities in Software Development

Antonio Alexandre Moura Costa
Federal Institute of Paraiba, Brazil

Felipe Barbosa Araújo Ramos
https://orcid.org/0000-0002-0937-811X
Federal Institute of Paraiba, Brazil

Dalton Cézane Gomes Valadares
Federal University of Campina Grande, Brazil & Federal Institute of Pernambuco, Brazil

Danyllo Wagner Albuquerque
Federal University of Campina Grande, Brazil

Emanuel Dantas Filho
Federal University of Campina Grande, Brazil

Alexandre Braga Gomes
Federal University of Campina Grande, Brazil

Mirko Barbosa Perkusich
VIRTUS, Brazil

Hyggo Oliveira de Almeida
Federal University of Campina Grande, Brazil

ABSTRACT

Software development has been considered a socio-technical activity over the past decades. Particularly, in the case of software engineering, the necessity to communicate effectively with stakeholders and team

DOI: 10.4018/978-1-7998-4165-4.ch008

members has been progressively emphasized. Human resources play a critical role in the success of software projects. Many techniques, methods, tools, models, and methodologies have been proposed, applied, and improved in order to help and ease the management of the software development process. Regardless of the software development methodology adopted, delivering a quality product in a predictable, efficient, and responsive manner is the objective for every team. Disciplined and Agile teams have different characteristics, but also share common aspects when working to accomplish their goals. The main motivation of this chapter is to present the differences and similarities of both teams in the context of software development.

INTRODUCTION

Delivering high-quality products in time and without budget overrun is still a significant struggle for most software organizations. Among the most common reasons are inaccurate estimates of needed resources, unmanaged risks, sloppy development practices, poor project management, commercial pressures, among others (Charette, 2005; Macnab & Doctolero, 2019). Software projects failures have a major negative impact on the organizations and can deeply compromise its future.

Projects continue to proliferate in society today, in both the public and private sectors of the economy. Investments in projects number in the trillions of dollars annually. Just as ubiquitous as these projects, unfortunately, are their significant failure rates. The CHAOS reports have identified the current state of project success rates across organizations, noting that in spite of much higher visibility and importance placed on project performance, failure rates have remained high and relatively stable across over a decade of research. (Serrador & Pinto, 2015)

When a software project fails, it jeopardizes an organization's prospects. If the failure is large enough, it can steal the company's entire future. In one stellar meltdown, a poorly implemented resource planning system led FoxMeyer Drug Co., a $5 billion wholesale drug distribution company in Carrollton, Texas, to plummet into bankruptcy in 1996. (Charette, 2005)

For a long time, Disciplined approaches had been used to increase project success. These approaches, also known as plan-driven or heavyweight methodologies, are conducted in a linear way, whereas process activities of specification, development, validation, and evolution must be performed in sequential order, which means that an activity must be completed before the next one begins. Due to the heavy aspect of the Disciplined approaches, several consultants have developed methodologies and practices with more emphasis on people, collaboration, customer interaction, and working software, rather than on processes, tools, documentation, and plans. These approaches, known as Agile methods, have been gained popularity over the past years, especially after the advent of the Agile Manifesto in 2001 (Fowler & Highsmith, 2001). Unlike Discipline approaches such as the Waterfall model, the Agile ones promote continuous iteration of development and testing throughout the software development lifecycle.