Case Study Before it was called Agile

Dr. Martin Mandischer started developing software in 1979 on the only Apple II Computer in our school. It was the usual stuff you do in school: Basic, Pascal and Assembler programming of sorting algorithm, plotting of math functions and small games. A little later he started programming text editors and developed plotter software for a small local company. He earned a small amount of money, which was used to fund summer hitch hiking through Europe.

He goes on to report: "After I finished high school followed by two years of boring studies at the University I decided to have a break and do some real things. So my first 'real' project went on in the two years between 1986 and 1988. I got a job as programmer at a large university hospital nearby Frankfurt. I was directly located in the purchase department. There were 20 people in this department doing all the purchases for the whole clinic on a single electronic typewriter that was operated by a single overloaded secretary.

"The goal of the head of the department was to improve their daily processes and reduce errors and misunderstanding that frequently occurred at the typewriter. We started with nothing else but this high level vision. Not even a computer was in place at that time.

"During the first week we had several brainstorming sessions with the management (stakeholders), purchasers (key customers) and secretaries (potential users). During this week the vision formed into a high level 'plan.' Each purchaser should be able to quickly process his orders at his own desk. To reduce errors potential order items and buyers addresses should come from a single database.

"With this plan we purchased a computer and asked ourselves, 'What is the most important thing we should start with? Is it the buyers processing the orders themselves or the database that helps reducing errors and saving time?'

"The decision was made while talking to each other. I and a part time assistant programmer were talking about effort and dependencies and the business side about value, which in this case was less costs. Our idea was to have a minimum of functionality in place as soon as possible and grow it over time (iterative).

"We had no detailed requirements nor did we have a fixed architecture. We started with a few framework decisions and paper based sketches of screen designs and a simple workflow I worked out with some key users.

"The first version that could be used was ready within six weeks. It was close to the old typewriter but computer-based with an address and material database in the background to fasten the process of typing an order and to reduce errors. Today one would call this a minimal viable product.

"The first 'production release' ran on a Unix based Siemens MX-2 with 2 MB of main memory and was used by one secretary. It was twice as fast with drastically reduced error rate.

"During that six week I had feedback on an almost daily basis. Whenever I made progress on the screens I showed it to a few key users and asked them for feedback. Every question that popped up during database design or workflow was answered within hours.

"Due to a code generator framework and a simple front end description language – nowadays this is called model-driven software development – it was easy to have running software within days. The development prototype was available at all time to all users. At the end of the day I generated everything, compiled it, updated the database, did some testing and copied the executable into the users prototype workspace. This was continuous integration done manually.

"Although far from being full blown the software, it provided the users with a chance for hands on experience and allowed them to envision new functions and improve on those already there.

"Of course we sometimes slipped the delivery of prototypes for a few days because there was nothing new or a lot of technical things to do. We also had no defined process and no one controlling it. But the feedback was helpful for me and I saw that I could produce better things based users experiences, so I stuck to what was more a good habit than a fixed process.

"During the following 24 month the software was enhanced by master data management, import and export interfaces, convenience function, front end improvements and dozens of reports, serial letter and spread sheet support.

"After six months all of the 20 people in the department were using it on a daily basis and all of them were giving regular direct face to face feedback. Most of the small improvements and bug fixes were directly integrated into the next release; larger ones were prioritized together with the head of the department. We had a regular *jour fixe* once a week for prioritization and regular delivery into production almost every day.

"Working directly with the user was hard sometimes. It was a mutual learning experience. It was a pleasure to see things that were needed by real people come into life in very short time and it actually improved their daily working.

"At that time I had no idea that this would be named Agile or XP or continuous delivery, it was as simple as delivering useful working software frequently.

The software was replaced after 25 years of usage in 2011 by an SAP system."

Exercise 2.4.2

- Which of the four statements found in the Agile Software Development Manifesto are highlighted in this case?
- Which of the twelve principles of Agile software are illustrated in the case?
- The Declaration of interdependence for modern management contains six rules of operations. Which are found in the case?

Exercises 2.4.3

- How to Use this at Work – In groups of two to seven people, have participants list ways that they will use the materials presented back at work. If the group cannot come up with any application, you can share their problem with the whole group.

Section 2.5

Understanding Agile project management

We began this chapter by examining the thrust of Agile philosophy. Next on our journey we will demonstrate how Agile projects differ from the traditional way of managing plan based projects. Traditional and Agile Projects both face with a range of Project Management constraints as illustrated by Fig. 5 below. The Project Management constraints are Scope, Time, Cost, Risk, Quality and Resources.

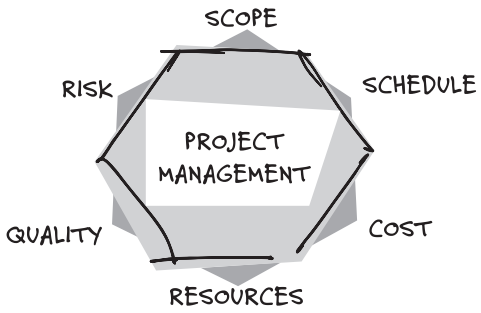

Fig. 5 *Project Management Constraints*