Education Impact of Evolutionary Software Development

Václav Rajlich
Department of Computer Science
Wayne State University
Detroit, MI 48202
rajlich@wayne.edu

Abstract

The paradigm shift from waterfall to evolutionary software development (that includes agile development) has been widespread in industry, but academia is still struggling with it. This position paper reiterates the reasons for the paradigm shift that illustrate the importance of the shift. The position paper then discusses what the shift means for the software engineering education. As an example, it briefly presents how software engineering is taught at Wayne State University.

1. Introduction

Instructors of the first software engineering course (1SEC) select the course topics from the vast array of results that were accumulated in software development history. Today’s selection is done in the presence of the recent paradigm shift from waterfall to evolutionary development [1]. The reasons for this shift, reiterated in this position paper, illustrate its significance.

Programmers following waterfall pass through the stages of requirements elicitation and analysis, design, implementation, delivery to customers, and maintenance; the design serves as a blueprint for the entire implementation. Various variants of waterfall lifecycle more-or-less follow this scheme. However this waterfall development often does not work because of the volatility that is common in software projects. There is a volatility of requirements [2] and volatility of rapidly changing technologies that makes long-term planning inaccurate or even impossible, therefore the design cannot be a good blueprint for entire implementation of large systems that experience big volatility; most of the current software systems are like that.

The problem is further aggravated by the fact that stakeholders who are not involved with the project on daily basis often do not know whether the early stages have accumulated a hidden technical debt. For example, the requirements can be of low quality, the design may be incomplete or suboptimal, and so forth. Because of software invisibility that is one of the essential difficulties of software [3], the stakeholders have few tools that would accurately show them the true state of the unfinished waterfall project. Instead, they rely on the personal reports from the developers, but the reports are often biased. The things then come to their head at the very end of the implementation, when the consequences of accumulated planning and monitoring failures become obvious, but it may be too late for corrective action. The consequences can be overwhelming and they were vividly described in the literature as “death march projects” [4].

2. Paradigm shift

The shift to the evolutionary paradigm is an answer to the problems of waterfall [5]. In evolutionary paradigm, the developers add one feature (functionality) after another to a functioning program, until all requirements are met. The change in requirements caused by
volatility only mean a timely change in the project plans. At any time, there is a running - albeit incomplete – program and any stakeholder can assess the features that are already present, which reduces the likelihood of late surprises. If there is a slippage in the plan, there are early warning signs available to all stakeholders, who can take corrective action. Since the disadvantages of waterfall are so fundamental and the evolutionary development solves them, the shift from waterfall to evolutionary development is in all likelihood a lasting one.

Some authors still incorrectly identify software evolution with the maintenance stage of waterfall, ignoring the fact that evolution deals with the very early stages of code development, not just post-delivery maintenance. Other authors inaccurately use agile, iterative, and evolutionary development as synonyms. In reality, evolutionary processes are all processes that add one feature at a time to a functioning program. Iterative processes are a subset of evolutionary processes where the project is organized around iterations. Each iteration has a plan of what is to be accomplished, and has a delivery date where the stakeholders assess the state of the project and create the plan for the next iteration; the organization of the evolution into iterations makes the project easier to manage. Agile processes are iterative processes with short iterations (in the order of weeks) and the developers themselves assign the tasks based on their consensus. The following formula describes the relationship of the three process categories:

\[ \text{Agile} \subseteq \text{Iterative} \subseteq \text{Evolutionary development}. \]

3. Consequences for the instructors of 1SEC

The industry understands the superiority of evolutionary development and embraced it wholeheartedly; waterfall survives only as an exception and is no longer the mainstream [6]. Academia is lagging behind the industry; for example, the current academic curriculum recommendations contain 14 different software engineering topics, some of them closely tied to waterfall. Software evolution is just one of them, without any indication of its large significance in the software development [7]. The new curriculum that is being prepared, so far has not significantly increased the visibility or weight of software evolution [8]. In the historical context, this is not completely surprising as paradigm shifts always have been protracted social processes accompanied by significant resistance and controversies [9].

The old waterfall paradigm is a comfortable and well-rehearsed topic, with 40-year history of research and teaching. In contrast, the students often have to learn the fundamental practices of evolutionary development on their own. This may be possible for the brightest students, but it is a significant problem for the rest, who learn only peripheral topics of their future profession and of their academic major. Still I believe that the instructors of 1SEC will take into account the interests of their students who cannot learn software evolution on their own; the future of these students is at stake in these controversies.

4. Experience at Wayne State University

When we were designing current 1SEC at Wayne State University, we realized that the key educational issue of the evolutionary development is addition of a new feature to an existing program, called software change (SC). It is a practice that is common to all evolutionary development processes, including iterative and agile. The research in SC has progressed to the point where it is possible to adopt a phased model of SC that is suitable for undergraduate teaching [1]. We explicitly rejected a strategy where students are asked to learn advanced agile practices without knowing the fundamentals, because it again requires them to learn the
fundamentals on their own. We believe that if they move to the next level prematurely, they will learn neither the basic, nor the advanced material.

The phased model of SC consists of phases and a specific SC enactment may contain all or some of them. The first phase SC is concept location that finds the specific code snippet that needs to change. This is the key to SC; while it can be easy in small or familiar programs, it can be a challenge in large and unfamiliar ones [10]. SC often impacts several classes and impact analysis phase determines the full extent of SC [11]. Concept location and impact analysis constitute SC design.

Actualization is the phase that implements the required new functionality by either modifying old code, or creating a new code and then plugging it into the existing code. The portion of actualization that makes secondary changes in related existing code is called change propagation. Refactoring is a phase that changes the structure of software without changing its behavior. It is done either in preparation for actualization (prefactoring) or after actualization as cleaning of the unwanted after effects (postfactoring). There are numerous refactoring transformations and each of them addresses a specific concern, sometimes called anti-pattern or bad smell [12]. Verification assesses and improves the correctness of all code updates, whether they are a part of actualization or refactoring; it is most commonly done by inspections and testing. The projects and the hands-on experience are described at http://www.cs.wayne.edu/~vip/ProjectAndLabs/index.html.

Besides SC, 1SEC at Wayne State University also teaches relevant technologies like version control systems or UML, overviews software evolution processes including agile and directed development, and also very briefly overviews broader topics like professional ethics or software management [1]. It has been taught in this way for four years, and the experience indicates that it has been a successful approach that benefits the students [13].

5. References