Technical Debt: The Ultimate Antipattern

The biggest costs may be hidden, widespread and long term

Lawrence Peters
Project Management Consultant, Software Consultants International Limited, Auburn, Washington, USA
Lecturer, Software Project Management, EMSE Program, Universidad Politecnica de Madrid, Spain
Ljpeters42@gmail.com

***Abstract*—Software projects run the gamut from simple to complex, difficult to impossible and everything in between. Software project managers and their development teams must cope with and adapt to unforeseeable changes in nearly every aspect of the project as originally envisioned, scheduled and planned. In spite of all this turmoil and chaos systems get built, they work and at a later time are seen as having been created via a variety of imprudent development practices now collectively referred to as technical debt. This paper examines the hidden cost of expediency by probing what taking shortcuts does to productivity, morale and turnover on the project identifying debt that goes much deeper than technical.**

Keywords—software project management, software cost, project success factors

I. Introduction

In the movie, "2001 – A Space Odyssey," there is a scene where the computer (HAL 2000) says, "I'm afraid I can't do that, Dave." Software engineers can be put into situations where they would like to say something similar to their manager or colleagues when pressured into engaging in a programming practice which they are likely to regret. Early in the practice of software engineering, the mantra seemed to be, "if it works, do it." Today, getting code to "work" may be the easy part. Creating code that is stable, easy to maintain and secure together with a plethora of other qualities is what stakeholders expect from our software engineers. But there is a problem. Short sighted practices of the past are still with us in the form of code that was created decades ago that is still in use today. The cost of cleaning it up or replacing it altogether is only the proverbial tip of the iceberg. The real cost(s) go much deeper than what we see today. These lie in the effects that cognitive dissonance and other factors have on the development team and its individuals. These effects are short and long term with implications for software project managers.

II. Why People Work

The first step in exploring the hidden effects of engaging in technical debt is to understand its impact on software engineers. This closely relates to people's motivation to work. For several decades, psychologists have been trying to figure out just why people work. The obvious answer is that people work in order to make money so they can acquire the stuff they need to survive. But it goes much deeper than that. In the 1954 movie, "On the Waterfront," Marlon Brando's character is having an argument with his brother about having been forced to throw a boxing match. His brother Charley reminds him he received a lot of money to which Brando's character replies, "it's not about money, Charley, I could have been somebody." Three models about why people work have been accepted by various groups as accurate. These are:

- Herzberg [1] proposed a "Two Factor Theory" to explain why people work. These were "hygiene" defined as being survival related having to do with pay, working conditions, respect, and job stability. The other factor, "Relationship to the job" involves advancement, promotion, fair treatment, and potential for higher rewards.
- Maslow [2] proposed that people have a hierarchy of needs and working satisfies the hierarchy. Starting at the top, these are Psychological needs, Safety and security, Social needs, Esteem needs and Self-actualization.
- McClelland [3] this theory proposes that people work to satisfy three needs – achievement (to do something important), Power (to have control over others and/or their own actions), and affiliation (friendly relationships)

While all three models involve some issue(s) related to self-esteem in one way or another, Herzberg's model may be the most widely accepted model of the three. Software project managers who rely on these factors (i.e. pay rate, working conditions and so forth) to control and motivate software engineers have high turnover rates [4] and turnover increases development costs by as much as 60% [5]. Since no one wants to be associated with a failed project or poor quality, taking shortcuts reduces productivity. In addition, more recent work [4] has shown that technology workers have a common value system which places a high priority on producing work that they can be proud of, that their colleagues will be impressed by and is at or near the state of the art. Producing work that incurs technical debt violates these goals. These goals can be extremely strong in some software engineers. For example, a software engineer who I managed was part of a team on a very aggressive schedule. Some of the work involved fixing bugs in what might be most kindly described as, "spaghetti code. He informed me that since his name was going to be associated with this stuff, he would go beyond just fixing the latest bug and re-construct and re-test the routines assigned to him. Even though he voluntarily worked quite late some nights, he still met the schedule and, due to some requirements changes, actually saved himself and his successor a great deal of time to respond to these changes. His colleagues who simply did the minimum were not so fortunate. An environment in which doing the work less than the "correct" way reduces productivity making it more expensive in several ways than doing it right in the first place. The ways in which these "hidden" expenses are incurred include:

- Loss of Productivity – When a person knows that what they are being asked to do is in conflict with what they know they should be doing, they are working at their lowest productivity level [6]. The phenomenon is called cognitive dissonance. Cognitive dissonance reduces productivity. In the case of software development productivity is often equated to the production of source code. But source code production has only increased by less than one line of source code per programmer per month per year in the period from 1960 to 2000 [7]. This increase has been linear even though dozens of programming, analysis, design, testing and other methods were developed and engaged in over that period [8]. Cognitive dissonance means that even though the coding practices being used are seen as justified because they will foreshorten development flow time, they may, in fact, increase it making this practice something of a self-fulfilling prophecy. Management may be inclined to speculate that even though shortcuts were taken, we still had trouble meeting the schedule. When people are able to fulfill their perception of being productive with work that reflects positively on them, they become highly motivated and are more productive. Therefore, technical debt may actually increase, not decrease, development time.

- Reduced commitment to quality practices – Cutting corners and/or abandoning our standard development process simply sends the wrong messages to the software engineering team. These negative messages all reduce productivity and motivation while undermining the self-confidence of the development team. They include:

    - The development process you so diligently spent time developing and refining is OK when things are going smoothly but must be abandoned when we get behind.

    - We are committed to quality only when it is convenient.

    - Management does not believe we can do it right and finish on time, within budget.

- Reduced use of collective team experience – Most project managers try to compose development teams in such a way that each member possesses skills and experience that complement the skills and experience of the other team members. In this way, the team as a whole possesses the skills and experience needed to be successful. Schedule pressure may be the key driver behind engaging in technical debt [9]. But to overcome possible resistance to engaging in technical debt, a lot of pressure will have to be brought to bear on the team. We now know that if enough pressure is put on a team, they cease to work together as a team and revert back to working as a group of disconnected individuals [10]. Meaning that the collective knowledge and experience within the team is lost.

- Increased cost – We know that in most job markets, software engineers can experience a high degree of mobility. If they do not like working at one firm, there is often another one seeking to hire or, in the case of larger firms, they can transfer to another organization. Experiencing cognitive dissonance, pressure to cut corners and so forth can increase turnover. As stated earlier, we now know that turnover can account for as much as 60% of the cost of a software project [5]. So we have a paradox, the very practice engaged in to reduce flow time and costs (technical debt) may actually increase both due to lower productivity via reduced motivation and increased turnover.

- Putting off correcting/paying technical debt can be really expensive. We have known for decades that the later in a software system's lifecycle we correct a problem the more it costs and that these costs increase exponentially [11]. Taking shortcuts now with the intent of correcting the problems they cause until later practically guarantees increased total system life costs. Besides, software engineers prefer to be creating new code, not cleaning up somebody else's mess.

- Undermining a culture of professionalism by setting a tone of just getting the code out when, in fact, people want to be associated with a culture in which getting quality results is the norm. We know the long term benefits of generating quality code in terms of maintenance [11] but the short term benefits in terms of improved productivity have only more recently been identified [4]. Five factors were identified that effected the motivation of high technology professionals [12] (Table I).

TABLE I. MOTIVATING FACTORS IN SOFTWARE TASKS

Factor	**Description**
Skill Variety	Task requires use of multiple skills
Task Identity	The task is something the software engineer would like doing
Task Significance	The task is seen as important
Autonomy	The software engineer can accomplish the task as they see fit
Feedback	Management provides feedback on how well the task has been done

Looking at Table I we can see that engaging in technical debt is the antithesis of several of the factors that enhance motivation thereby reducing productivity.

III. Antipatterns

Most of the actions taken by software project managers are rooted in a desire for their project to be successful in whatever context success is seen in his/her environment. Due to a lack of training in project management [4] or misinformation, they often take actions that seem to them to be solutions to one or more issues but actually make matters worse. These “good ideas” that were actually “bad ideas” have come to be known as antipatterns [13]. Nearly 100 of these have been identified and labeled and the list is probably growing all the time. They have been organized into groups making classification and referencing easier [14]. In most cases, the negative effect of an antipattern (e.g. adding people to a late project in order to get back on schedule) have a narrow focus and a localized, negative impact. As the previous discussion indicates, technical debt can have a negative multidimensional, broad range impact on the development team, the project [15] and the organization. The obvious as well as hidden negative effects of technical debt makes it the ultimate antipattern. The term ultimate seems appropriate here since the effects of engaging in this practice are largely hidden, may have a long lasting negative impact on the development team and its members and is difficult to quantify.

IV. Closing Comments

As a minimum, the subject of technical debt has brought with it discussions regarding what software development practices are advisable and which should be avoided. Missing from these discussions is the subject of the human toll involved due to cognitive dissonance, personnel turnover and the potential for an attitude in the organization that doing things the “right” way only occurs when the project is on schedule and within budget. As we have seen, the intended effect of engaging in technical debt is illusory. That is, in some cases it may appear to have helped the software project manager’s cause but closer inspection reveals it has likely made matters worse but frequently in hidden ways. Hopefully, this will not become a *de facto* standard practice within software project management.

References

[1] F. Herzberg, Work and the Nature of Man, The World Publishing Company, Cleveland, Ohio, 1966.

[2] A. H. Maslow, The Farther Reaches of Human Nature, The Viking Press, New York, NY, 1971.

[3] D. C. McClelland, The Achieving Society, Van Nostrand Rheinhold, Princeton, NJ, 1971.

[4] R. Katz, “Motivating Technical Professionals Today,” IEEE Engineering Management Review, Volume 41, Number 1, March 2013, pp. 28-38.

[5] E. Cone, Managing that churning sensation,” Information Week, May, 1998, No. 680, pp. 50-67.

[6] G. M. Weinberg, The Psychology of Computer Programming, Van Nostrand Rheinhold, New York, NY, 1971

[7] Jensen, R.W., “Don’t Forget About Good Management,” CrossTalk, p. 30, August, 2000.

[8] Rico, D.F., “Short History of Software Methods,” downloaded from web, August, 2010 – referenced with author’s permission

[9] P. Kruchten, R. L. Nord and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” IEEE Software November/December 2012, pp. 18-21.

[10] H. K. Gardner, “Performance Pressure as a Double Edged Sword: Enhancing Team Motivation While Undermining the Use of Team Knowledge,” Working Paper 09-126, Harvard Business School, January 2012.

[11] B. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[12] L. J. Peters, “Motivating Software Professionals,” The Software Practitioner, May, 2013.

[13] P. A. Laplante and C. J. Neill, *Antipatterns: Identification, Refactoring, and Management*. Boca Raton, FL: Taylor & Francis, 2005.

[14] P. Dias de Silva, Categorization of Software Project Antipatterns, M.S in Software Engineering Thesis, Universidad Politecnica Madrid, June, 2014.

[15] L. J. Peters, Software Project Management on the Edge of Chaos: from Antipatterns to Success, Kindle ebook, 2014, Software Consultants International Limited.