On Design in eXtreme Programming

Extreme Programming (XP) challenges the traditional ways in which software systems are designed though removing the reliance on the up-front design approach favoured by classical engineering models. In Martin Fowler's essay "Is Design Dead?" [1] he ultimately argues against the assertion in the title of the article. He argues that design still happens in XP but the way in which we do design has changed. In XP design removes the need for the formal, rigorous and bureaucratic approach to design present in earlier software development methodologies and towards an emphasis on putting flexibility and simplicity at the forefront of the process.

Among the more interesting points presented in the article, Fowler asserts that the ability to carry out evolutionary design relies on what he calls "enabling practices", namely continuous integration, testing, and refactoring which allow good design to grow organically as the project evolves by diminishing the cost of change. This is echoed by Robert C. Martin in his classic text "Clean Code" [2] in which he advocates the necessity of testing to give programmers confidence in their ability to change the design of the system through refactoring without the fear of breaking it. Fowler also includes continuous integration as the means to give confidence to developers at the level of the team instead of the individual.

A major theme throughout the piece is the necessity of simplicity and the "You aren't going to need it" (YAGNI) principle in the XP paradigm. The main arguments that Fowler attempts to communicate is that XP encourages us to build the design we need instead of the design we want. To clarify that statement, XP done right naturally gravitates towards the simplest appropriate design. It achieves this by not building anything that isn't relevant to the current iteration's user stories. This prevents, in Fowlers words building the wrong thing early when future circumstances change. Additional complexity leads to a design which is harder to efficiently refactor later, while a design that is too simple is easier to build upon.

Perhaps the most interesting sections in the article relate to the applicability of traditional design tools to XP. Here Fowler clearly states that he is a moderate in comparison to contemporaries such as Kent Beck and Robert C. Martin. Hardcore XP advocates tend to reject the traditional use of a formal design language such as UML in favour of techniques such as CRC cards [3] whose primary advantages are their intrinsic simplicity (they take minutes to create), flexibility (multiple designs can be rapidly trialled and discussed), and finiteness (both in terms of physical size and length of existence).

Fowler offers a common sense compromise by suggesting that UML has a time and a place within the XP methodology. UML offers an abstraction that facilities the efficient communication of designs between software engineers and for sketching out a scaffold for a system. The major point of contention with using UML as in XP is when UML becomes less of a napkin sketch and more of a stone tablet. Diagrams should not be set in stone and should exist only as long as they are useful. Their use as a form of documentation is redundant almost as soon as it's written in a methodology that organically evolves over time.

A similar argument is given in regards to the use of design patterns. The contents of the Gang Of Four [4] is a gold mine in the hands of a good developer, regardless of the methodology used. The principle argument spouted by XP proponents is the blind overuse of patterns in every vaguely applicable situation (often by junior developers). Such an approach violates the axiom of simplicity. Patterns have as much of a place in XP as they do in any other methodology. The key concept to grasp is that patterns should be used when it makes sense to use them and not before (which would violate YAGNI) and that developers should gave the courage to remove a pattern during refactoring should it be deemed overly complex. This

issue is humorously summarised by developer Jeff Atwood's rant against the "Head First Design Patterns" book [5].

Another key point that the article makes is the mindset of the developer to design in XP. Fowler asserts that we must move away for the concept of "software architects" who separate themselves from the herd and occupy themselves the the higher realms of abstraction while leaving other engineers to do the dirty work based commandments handed down from above. In XP a different mindset is required. Expert designers should coach the less experienced members of the team thereby bettering their abilities while promoting team cohesion.

There is also the implication that although developers may have varying skill sets, all developers are equal within XP. Younger members should be encouraged to contribute to design. This aids with the sense of project ownership which leads into Fowler's later point about the "The Will to Design". Having team members who are enthused by a project and take an active part in design nurtures the courage and will to improve the design of the system. More eyes on the code base means issues are caught and resolved sooner. It doesn't need to be everyone's primary responsibility, but it helps.

My own opinion on design in XP closely mirrors that which Martin Fowler puts forward in "Is Design Dead". I believe that XP's approach to design is sufficient for building software systems. I would even agree that effective design can be achieved by sticking to the more hard-line versions of XP advocated by the likes of Kent Beck, where there is no place for UML and that all design issues are resolved by refactoring exclusively. However I am not content to throw my weight behind such a hard-line view because it strikes me a overly dogmatic and (ironically) restrictive.

Yes, I would agree that a an effective design in a large project can be achieved without a single UML diagram or basic architectural design but my question is why you would want to limit yourself so harshly? If an idea can best be communicated through the use of a quick UML diagram scrawled on a white board then do it. They only become an issue when they are themselves to tightly adhered to.

Speaking from my own personal experience on industrial placement I can happily testify first hand that many of the techniques outlined in the article are working in practice. Continuous integration and detailed test coverage give the developer the courage to refactor and redesign as necessary and bypass the red tape of planned design. Likewise I've seen full UML designs that have been created only to become inaccurate within days of implementation. The problems XP attempts to tackle are very real and the solutions it offers have been conclusively proved to work in industry.

Failure comes from the teams that do not commit to the principles driving XP and evolutionary growth. That is not to say I believe XP should be followed to the absolute extreme, but rather that a team needs to use take the core ideas it preaches and define a process that works for them. Common sense should be the compass guiding how a process works. Having faith in the power of refactoring in changes, adhering the the YAGNI and DRY principles, and keeping minimising complexity in all its forms is essential for XP to be successful, however a loose architectural design at the start of project or a UML diagram on the back of a napkin isn't the end of the world providing the design can still evolve as needed.

References

[1] M. Fowler. Is design dead?, 2004. URL `http://martinfowler.com/articles/designDead.html`.

[2] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2008.

[3] Don Wells. Crc cards, 1999. URL `http://www.extremeprogramming.org/rules/crccards.html`.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Jeff Atwood. Head first design patterns, 2005. URL `http://blog.codinghorror.com/head-first-design-patterns/`.