Extreme Programming

ExtremeProgramming is one of many AgileProcesses. It is also known as XP. The names AthlonXP and WindowsXP would appear not to be related to ExtremeProgramming, but if interested, you could see the WindowsXpNameDiscussion.

See ExtremeProgrammingRoadmap for an index of information about extreme programming on this site.

The basic advantage of XP is that the whole process is visible and accountable. The developers will make concrete commitments about what they will accomplish, show concrete progress in the form of deployable software, and when a milestone is reached they will describe exactly what they did and how and why that differed from the plan. This allows business-oriented people to make their own business commitments with confidence, to take advantage of opportunities as they arise, and eliminate dead-ends quickly and cheaply. -- KentBeck

"XP is the practice and pursuit of effective simplicity, as applied to software development." -- Victor (from the XpMailingList)

ExtremeProgramming was best described by KentBeck in his landmark book *ExtremeProgrammingExplainedEmbraceChange* (take a quick look at that page for a 'back-cover' summary of XP).

This OneTrickPony ain't no SilverBullet. And vice versa. -- *PhilJp*

Interesting to note that ExtremeProgramming has emerged as a methodology for programming; it appears to be based in trial and error programming! Without tests and refactoring, it is not workable at all. So why not design first, write tests to that design, then program, test, perform refactoring and iterate?

That's exactly what ExtremeProgramming is, friend. Just do it in tiny increments.

The differences between XP and trial-and-error are basically two-fold: first, you solve each error as it comes up, so that it doesn't compound with later errors. And second, you meant to do it that way in the first place, rather than being forced to come back and deal with unforeseen complications. *(This paragraph would need to be written out more verbosely. I can't understand the point the paragraph is trying to make. PleaseComment)* this looks like a development of StepwiseRefinement where the project is broken down into smaller and smaller tasks until each task is small enough to code, the idea being that one task is one module to handle an operation ie input:process:output if a task is too big to code, its split into modules. these modules are iterated to the same criteria

---

The ExtremeProgrammingCorePractices constitute an ExtremeProgrammingProject.

Why "Extreme"?

"Extreme" means these practices get "turned up" to a much higher "volume" than on traditional projects. XP ignores any other practice (like BigDesignUpFront) that does not appear on the list. The result is stable, productive, and very rapid because the practices support each other the more they are used together without interference. An Extreme project is typically so stable and sedate it can lead to FortyHourWeeks without any schedule slips.

What really matters?

Software is too damned hard to spend time on things that don't matter. So, starting over from scratch, what are we absolutely certain matters?

1. Coding. At the end of the day, if the program doesn't run and make money for the client, you haven't done anything.
2. Testing. You have to know when you're done. The tests tell you this. If you're smart, you'll write them first so you'll know the instant you're done. Otherwise, you're stuck thinking you maybe might be done, but knowing you're probably not, but you're not sure how close you are.
3. Listening. You have to learn what the problem is in the first place, then you have to learn what numbers to put in the tests. You probably won't know this yourself, so you have to get good at listening to clients - users, managers, and business people.
4. Designing. You have to take what your program tells you about how it wants to be structured and feed it back into the program. Otherwise, you'll sink under the weight of your own guesses.

Listening, Testing, Coding, Designing. That's all there is to software. Anyone who tells you different is selling something.

-- KentBeck, author of *ExtremeProgrammingExplained*

The ExtremeEnvironment

XP is designed to meet the CustomerBillOfRights and DeveloperBillOfRights. Participants work in one of the ExtremeRoles (TheCoach, TrackerRole, etc.). XP is done during a FortyHourWeek, with AllEngineersInOneRoom, having a daily StandUpMeeting. As ThereMustBeFood, there are SnacksForPositiveReinforcement in the room where EngineeringTasks are carried out with a high ProjectVelocity. If you don't have this kind of ideal environment, you could still attempt ExtremeProgrammingInEnemyTerritory.

ExtremeListening

The OnsiteCustomer participates in the PlanningGame. The development team interviews the customer to determine UserStories about how the system will work. User stories are combined or split to come up with a story that can be described (WriteItOnaCard) on one of these CrcCards, and completed by a pair of programmers during one release cycle (typically about three weeks). The customer prioritizes the cards according to the business value of the UserStories, which puts stories into the release schedule based on the development resources available. This lets customers have an accurate ReleasePlan.

After each release, the customer has a system that works according to the stories completed so far; they don't have to wait for the whole thing to be done to start using the initial functionality. The development group identifies which stories are risky to complete on time (mainly because of a lack of experience with that type of coding) and does WorstThingsFirst based on a SpikeSolution. Read more about ExtremeRequirementsGathering.

Special note on Listening: we do listening in two ways:

1. UserStories. (Similar to "use cases".) On cards, our customers write stories describing how something is supposed to work. A story might say "An employee making $10 an hour works four hours of overtime on Friday and two on Sunday. She should receive $60 for the Friday overtime and $40 for Sunday." We have hundreds of cards describing the product.
2. AcceptanceTests. These are typically single use cases with expected answers provided by the customer.

-- RonJeffries

ExtremeDesign

XP relies on TestDrivenDevelopment for its inner loop. This pumps the outer loop, and makes the political end of XP possible. Promises are automatically

kept. ProgrammerTests are written which the successful software will be able to pass. At first, the tests fail because the software hasn't been written yet. A simple design is developed by doing the SimplestThing which might pass the tests. BigDesignUpFront is avoided because YouArentGonnaNeedIt. However, once the SimplestThing is done the programmers RefactorMercilessly, because in the end things should be expressed OnceAndOnlyOnce. Read more about SubsystemsInXp.

ExtremeCoding

XP involves pairs of programmers (who PairPromiscuously) working together on code, so that the entire development team achieves CodeStewardship. They use TestDrivenDevelopment to CodeUnitTestFirst.

This PairProgramming leads to CollectiveCodeOwnership. Programmers follow the CodingConventions whenever they write new code, and while they RefactorMercilessly.

ExtremeTesting

After code passes ProgrammerTests (cf RelentlessTesting), practice ContinuousIntegration (at TheIntegrationStation) of FrequentReleases and run AcceptanceTests to verify that the software meets the customer's requirements. NoBugDatabase is needed.

Several wikis describe/implement Web-based systems that support XP practices:

- http://www.program-transformation.org/twiki/bin/view/Transform/GenerativeProgrammingWiki
- http://xplanner.sourceforge.net
- FitNesse

Noted PerlLanguage luminary, and mad scientist, DamianConway believes that ExtremeProgramming is actually a misnomer. Since it embodies many of the good programming practises that programmers are taught but almost certainly ignore, he believes that it should really have been called UltraConservativeProgramming. -- SimonLaw

How is XP "best practices" different from other best practices lists? If you have EmpiricalEvidence beyond anecdotes, I would like to see the data!

I do not have data to publish, but I've done several projects this way and it sure makes projects go faster, way faster, although programming goes slower. Think about that, it makes lots of sense, since you are programming only a 10% of the time.

And since there are so many practices, if you skip one, you will think you are going faster, then when you fail, you will realize you were not going as fast as you thought. Another way of saying it is that you were going fast, but in the wrong direction, or you were going fast but you hit a wall (it depends on which practice did you skip).

The last project I worked on, was achieved with half the estimated people and half the time. The quality was not so good because we skipped functional testing during the whole project. When everything was coded and seemeingly functioning with a lot of defects, we wrote functional tests for all user stories (full use cases in our case). It just took 2 months to completely stabilize the system of around 300 use cases.

XP is spreading like wildfire in the industry, which is good("change is good"). There is one big danger, people always misunderstand XP as jumping right into the code and getting the code out, and figure out what the code needs to do later. This is not leaving us in any better position than we are today. I wish XP gurus will advocate more on listening and understanding of the problem before trying to solve it. If you are advocating to understand the problem while (well) after writing the solution, then that is wrong and ought to be illegal.

I think some coding should occur while you are understanding the problem. How many times when talking to an end user, have you thought you understood what they wanted and they thought something very different? How many times were details glossed over by the client that made a huge difference on the project? The client just assumed those features would be included but "forgot" to mention them! How many times was the customer incapable of visualizing the benefits of the system, you describe to them? You verbally describe it because it is too expensive to create it and then have them not like it. If you can get agreement on what a customer wants by them looking at a very quickly made example, you avoid 1) misunderstanding what is needed 2) the customer doesn't have to visualize something they have no expertise at visualizing. The trick is to show something useful enough to get across a concept that can be created very quickly. If you can create something for them to see while they looking over your shoulder, you are on the right track, although all problems aren't solved that easily. Breaking the problem into very small deliverable pieces, getting user buy-in all the time, having the system work at all times while you are creating it, are all features that contribute to XP programming success. Having a system that can be changed quickly and often, and a set of configurable tools where you don't have to start from scratch, makes XP programming practical. Most languages and development systems can't implement XP programming for these reasons. -- DavidClarkd

---

I'm unconvinced of ExtremeProgramming's utility when it comes to the design and construction of platforms for other software (e.g. programming languages, frameworks, message protocols, library APIs, operating systems, security models, concurrency models, etc.) - too often, the need to maintain backwards compatibility, especially to support the 'quirks' and 'bugs' of previous modes, quickly becomes an enormous burden against any attempt to RefactorMercilessly. However, I think it works fine for situations where the programmers DO possess full authority to muck around with all code and repair any dependencies broken as a consequence of refactoring. BigDesignUpFront vs YagNi, ProgramInTheFutureTense vs DoTheSimplestThingThatCouldPossiblyWork - I believe that each have their place in design and construction of software.

Discussion moved to ExtremeProgrammingForPlatformSoftware.

I vote to drop the *Extreme* and *Ultra* things. I understand that is *swiftly* abbreviates to something like XP, but this is the future of programming. It is. So why name it like the next non-evolution of washing agents. To me, all I have read in CategoryExtremeProgramming pages, *is* about programming. No more no less. Anything else than that doesn't *earn* the name programming with me, at most *coding*. -- StijnSanders

See WhyExtremeWasUsed

SimplifiedProcess seems to have been an ExtremeProgramming-like practice from GermanyCountry.

ExtremeProgramming NotQuickQuestions

Q: For "simple" programming language (e.g. CICS-COBOL/DB2) on BigIron (e.g., ZOS) development projects, would XP practitioners agree that *XP approach is not really suitable* because

1. nobody benefits from PairProgramming with these environments (no reuse / pattern applicability discussions needed)
2. it's not realistic to have users participate in the team because enterprise tasks (e.g., ERP integration) have organization-wide impact and are not suited for small but FrequentReleases
3. PairProgramming does not improve productivity as complexity is in the business and not in the technical environment
4. of inertia, and no one wants to be the first shop that does MainFrame development in XP. See also XpAndHierarchies.

-- DavidLiu

A: It may be arguable that XP would be difficult simply because it would be such a radical change from the existing practice and most people doing it have so much inertia/experience with their current approach. But two people could certainly pair and benefit from it without having to change the whole culture. The benefits would come precisely because the existing app is already mature and complex that it greatly helps to have two pair of eyes watching every change.

I have two junior programmers, fresh out of school, breaking out the UI from the business logic of a legacy app. They are buying into the value of pairing and writing tests. The old guard are stunned at how fast they are tearing through. Working together works. 8:-) -- BradWhite

Language used for that LegacyApplication?

Just considering the Extreme*Programming* aspect, this is exactly how programmers want to work, without necessarily knowing it. Coders like coding, not documenting, and coders like seeing code they've written work. I love it - no more huge DesignDocumentation, and rapid feedback from ProgrammerTests. It makes programming fun again, like when you were a kid (or when you were hacking some OpenSource piece where no one demands a 90-page design doc complete with full UnifiedModelingLanguage models).

About documentation: I recently started to put my documentation (UML snippets, method signatures and other remarks) inside my unit tests. I must off course admit that that is easy to do in a web-oriented language (PHP in my case). I use a lot of shared components and these components usually have some idea behind them that can be expressed in UML nicely. I thought that the unit tests were the best place for the documentation, as you will see it a few times a day anyway. And you *really want* to see it when that particular unit tests breaks. For a demo, see the tests at http://www.w-p.dds.nl/storyboard/admin/rununittests.php -- WillemBogaerts

---

Maybe not design docs but how about help docs? You have to document at some time, because it's how customers actually use those features that you thought up. It'd be selfish just to code and never document or comment on how the software works. But I'm sure you mean just reduction of design docs, right? Any documentation is tough for a programmer since he can never explain the software in less than ten words. More like ten thousand words. (I think you'll see VideoDocumentation in the future as more popular.. and it's easier to 'write up' for a programmer) *(See UserDocsInXp)*

I've been an admirer of XP ever since learning of it at MIT in '01. However, it seems wedded to a development model *outside* of my "tribe" which is more the world of OpenSourceCulture. That is, my tribe codes without customers, without a budget, for personal use and for fun. Much of the Internet has been built from it. XP concepts are still amenable, but I would like to help re-work the idea for a **culture of coders** rather than a programming team answering to the Industrial Economy.

For OpenSourceCulture, new coders need to be able to come and go freely. This means tools like TestDrivenDevelopment become primary, over UserStories, for example. The **test** becomes the UserStory because the customer is the coder-community! In Python with DocTests this is perfect because DocTests encourage the story to be filled out with prose. In other, write DocTests code for all the functionality you hope for and wait for the community to make the tests pass.

As a quick attempt, the takeaway concepts of XP I see are these: PairProgramming, TestDrivenDevelopment, and ReFactoring. Thanks!

-- Mark Janssen

See ExtremeProgrammingRoadmap, ExtremeProgrammingPrinciples, ExtremeProgrammingTopics, ExtremeProgrammingLinksOutsideWiki, ExtremeProgrammingForDummies, ExtremeProgrammingProjects, WhoIsUsingExtremeProgramming, ExtremeProgrammingArticles

---

AprilZeroEight

CategoryExtremeProgramming CategoryExtreme
CategoryExtremeOpenBusiness

BiLinks:

<->

- ExtremeOpenBusiness *(is a generalization of ExtremeProgramming, applying experiences and principles to the Business Process. It is especially interesting for the CreativeCommons.)*

Last edit November 9, 2014, See github about remodeling.