Refactoring Iteration

XP requires you to EmbraceChange. Sometimes you have to rip the crap out of your design to meet some changing requirements. When you need to make a fundamental or global design change and some external force beyond anyone's control is driving you to get it done now. You need a RefactoringIteration where refactoring is the theme of the entire iteration.

This has nothing to do with how to refactor iterations, I'm afraid; see CartesianProduct.

When did RefactoringIteration come to be an XP practice and part of the PlanningGame? Did I miss a meeting? I think of it as a recovery technique more than as something ordinary. -- RonJeffries

I guess I didn't express my point clearly, as I'm not attacking RefactoringIteration in any way. I'm all for having RefactoringIteration as an auxiliary XP practice. We might even put it in the book, since two important projects (at least) have needed to use it. My concern is ONLY that it popped up in the middle of PlanningGame just as if it had been written by Kent. (He didn't sign the section, so editing it inline was certainly consistent with WikiNature. It just confused me, because it's not in the book.)

When a project does full XP, i.e. customer and developer, including the planning practices, a RefactoringIteration may be necessary for the reasons you've so well described. However, it will by the nature of its causes, be unplanned. As such, it has to be communicated as a major plan change. And in my opinion, while some RIs may be normal course of events, such as Vcaps', many are likely to be the result of some kind of slacking from ongoing refactoring, as was the C3 RI back in late 96. That being the case, the student must be advised to explore the cause of the RI to be sure it isn't evidence of something correctable.

You're doing a fantastic job with VCaps, and you're totally right to adjust the practices continually, just as we do on C3. My only concern, and it's probably an overreaction, is to keep clear what's really in XP-the-documented-process, and what's in the apocrypha that your project, and the C3 project, create. Keep it coming! -- RonJeffries

Ron, I apologize and thank you. No one here at the VcapsProject created that link on PlanningGame nor would we have wanted one. Our intent in creating this page was to give the practice a definition, justify its use at our site and explore why it is you consider it a failure while we do not. RefactoringIteration happens. -- DonWells

---

Let us contemplate AreYouDoingXp.

AreYouDoingXp says: To be extreme you must **communicate**. A RefactoringIteration communicates to management and customers exactly what impact their new requirements have made. Communication within the team goes without saying.

AreYouDoingXp says: To be extreme, you must be **aggressive**. A RefactoringIteration is a very aggressive approach to dynamic requirements. Is there such a thing as too aggressive?

AreYouDoingXp says: **Optimizing process you are aware of your software development process, you are aware of when it is working and when it isn't, you are experimenting to fix the parts that aren't working**. We know our process is working. We try new things and keep them if they work, RefactoringIteration is one example.

Why demoralize a team with the stigma of failure? Wouldn't it be better to buoy the team up, just when they need it the most, by calling them to the challenge?

We do not see how this potentially erodes XP. We see each XP practice as having two levels of application: Pair level and Team level. You can get too focused on details and not see the bigger picture of how each pair level practice has an associated team level practice. Without team level practices, the synergy of the team cannot be applied when you need it. Some problems need to be hit with a team. -- DonWells

---

By this he means, each pair can be working on different EngineeringTask-s and still see how their piece of the puzzle fits into the big picture. -- JeanineDeGuzman

---

This situation could also arise after working on an application in individual pairs for multiple iterations. Because each pair is focused on their specific EngineeringTask, it is only natural that they may not see opportunities to refactor, redesign, etc on a larger scale. These opportunities often arise when ExtremeProgrammingCodeReviews are done. -- TomKubit

Tom, your comment confuses me. As I understand it, many UserStories are done per iteration, and each UserStory is split via CRC (?) into EngineeringTasks. So each individual EngineeringTask is assigned to a pair, but a pair will work on multiple EngineeringTask(s) derived from multiple UserStories in an iteration, no? If that's the case, then how could your pairs get so uncoordinated?

Also, the discussion on ExtremeProgrammingCodeReviews suggests that ExtremeProgrammingCodeReviews are never done - reviews aren't part of the XP methodology. Me, I'm not entirely comfortable with this; at the very least, I think reviews should be used to evolve ProjectStandards. But is that what you mean?

Hmm. I guess I'm seeing that XP agrees with SpecializationIsForInsects - x-men please comment. -- PeterMerel

---

The master refactorer can make huge changes in tiny steps- water over stone. This takes

- vision
- persistence
- courage
- and skill.

If any of the four is lacking, a pause might reasonably be called to gain more. However, it is a break in the flow of feedback, and a break in the flow of customer decision points, both of which put the project at risk.

This last is key:

If any of the four is lacking, a pause might reasonably be called to gain more. However, it is a break in the flow of feedback, and a break in the flow of customer decision points, both of which put the project at risk.

Exactly. A RefactoringIteration that wasn't in the original schedule is evidence that you've gone off process. Project risk is increased and must be reported and if possible, reduced.

When any project goes off plan, these things need to be done:

- Determine whether whatever caused the deviation is likely to happen again. If unlikely, forget it. Else if unavoidable, adjust the process to avoid future occurrences. Else [unavoidable] stretch the plan to accommodate future occurrences.

- Produce a new plan showing the impact of the deviation and any future impact predicted. Associate any important process adjustments aimed at avoiding recurrences. Publish ASAP. Slips taken late give management less

time to adjust.

Peter, when I mentioned the ExtremeProgrammingCodeReviews earlier I probably should have mentioned it was in context of how we use them in the VcapsProject. DonWells recently added a nice summary at the bottom of the code review page discussing just that. I agree that completely throwing away a review process may not be in the best interest of a development effort.

Your assertions on how UserStories and EngineeringTask-s are doled out are correct. Our contention is that the stories and tasks are a finer grained level than what we're talking about here for what a RefactoringIteration is. As such, the point of view of the pairs is at the level of the story/task they are currently working on and the collaboration of directly related classes. And while these pairs work together, there are always refactoring considerations taking place for that iteration. In other words, all XP processes are being followed, and the project is moving along nicely. What we have found, though, is that there comes a time when much larger, fundamental design change opportunities appear that hadn't been anticipated earlier in the project. There can be a number of driving factors for this to happen, so let me see if I can explain it a little better with an example of something that happened on our project and I'll let the community here decide if we actually fell off track.

We had evolved a design that we felt was quite stable and met all the needs of the current requirements. This design remained fundamentally unchanged for a period of probably 6 months or more. The refactoring that was being done was at the level of abstracting out new classes within hierarchies and consolidating others as need be on a story/task basis. A new requirement came along that appeared to be a simple addition. However, while researching the necessary behavior that was to be provided, a major design change was discovered that could both accommodate the requirement and at the same time significantly simplify a large portion of the current system. I don't know about your views, but when something like this is discovered, I don't want one pair going out and making a global decision and undertaking the task without communicating it with the rest of the team. So we had ExtremeProgrammingDesignReviews with the whole team and realized that this change was going to save us a lot of work later on, so we revised our plan and scheduled a RefactoringIteration and assigned it to a couple pairs to hammer out while other pairs concentrated on tasks unaffected by the design change.

It was because we were using XP practices (especially having a solid unit test harness) that we were able to undertake this in a fairly short period of time. We feel it's better to adjust the schedule early (as soon as you notice) instead of later which will be harder because there will be more things dependent on the original design. Every project I have been associated with has never had all requirements up front, and even the ones that were fairly complete, there were always occasions mid-way through the project where significant revelations occurred. We decided that to have the kind of system we wanted, we needed to incorporate RefactoringIteration into our process. Getting management to buy in to adjusting the schedule was very difficult at first, but when we showed the tangible benefits after the first one, and improved the long term schedule in the process, they trusted our judgement more the next time something like this came up again. A short term schedule slip will often equal an earlier delivery.

Sorry for the long winded explanation, but I felt it necessary to get across the points we were trying to make by adding this page. -- TomKubit

Thanks for the explanation Tom, that makes things a lot clearer. So what you're saying is that a RefactoringIteration is a little like C3's recent RevivalMeeting in that it refocuses the entire team. It sounds to me, though, that what happened to you was that you actually changed your SystemMetaphor somewhat midstream, and that this is what required the RefactoringIteration - yes? I can certainly see how that would require a whole-team effort. -- PeterMerel

Yes! It brought us all together, but this is part of our normal practice. Instead of thinking we had fallen off our practices and needed to regroup, we came to it from the point of view as we found something major that wasn't *handled* specifically within what we knew as XP and that it would affect how the team thought about the basic system model. When anything like this is discovered, any team member can call a group design meeting to discuss it. So this is our solution of incorporating it into the XP process. Team wide communication is essential. -- tk

---

It is true any refactoring can be accomplished in tiny steps. I strive for this in my day to day work, but I myself am clearly not a master refactorer. Sometimes customers just change their minds. I am faced with the options of laboring under a system that is just not meeting our needs or go with a RefactoringIteration and EmbraceChange. I think that standing in front of management and customers and saying that in the long run taking one step backward will get us to the finish line faster requires

- vision
- persistence
- courage
- and skill.

Recognizing that a big change is required and a RefactoringIteration is the only solution is how the risk is reduced. If the problem was caused by customers surprising you with new requirements then yes, you bet it will happen again and you had better be ready to EmbraceChange again. Taking the RefactoringIteration as soon as you need it is exactly how you minimize slips and deadline impact. In ExtremeProgramming everyone works at their full potential, you can't have people working at half speed while someone is off polishing stones with a water gun. -- DonWells

Yeah, that's the choice. Which would you rather do? Polish stones with a water gun, or sculpt ice with a blowtorch?

Actually I have seen ice sculpted with a blowtorch at our nationally recognized ice sculpting convention. The designs are intricate, elegant, and beautiful.

Well, actually, I don't do software with a water gun or a blowtorch, so I can't help with that. It seems to me, though, that an iteration involving the entire team to change the architecture is a very unlikely occurrence unless you have a tiny team. It's much more likely, in a well-factored system, that even a serious architecture change can be done incrementally and by a small team.

The largest architecture change I can recall on C3 was done by two people during an ordinary iteration. The only special thing we did was allow some tests to fail for a few days during the cycle.

If the entire team needs to be refactoring, isn't it much more likely that they're working on lots of things? If the whole system needs refactoring, isn't it likely that something was actually discernibly wrong even before the new requirement arrived?

A RefactoringIteration is a form of process recovery. It is in any case unplanned and needs to be treated as an exception. That's not to say it shouldn't be done, just that one needs to look underneath it to see what can be done to prevent recurrence. More often than not, IMO, something can in fact be done. -- RonJeffries

---

I did not say the entire team was involved in implementing the refactoring, just that the entire team was brought together so the new design could be discussed and communicated so that everyone was on the same page. Once it was decided to move forward, only as many pairs as necessary were assigned to the refactoring task. In the case mentioned above, I cited 2 pairs (FYI, in another instance during the project, a single pair was assigned the task). In fact, one pair was driving the core refactoring effort while the other was making changes to supporting architecture because of slight changes to the interface of the classes being refactored. We were able to separate out the two EngineeringTask (s) because the application was already well factored. Because of the way we approached this, all unit tests continued to work at 100% for every incremental code release, and coordinating the effort was

fairly straightforward.

Secondly, neither did I say the entire system needed to be refactored. After reading my comments above again, I thought I made it quite clear that the portion of the system impacted just happened to be a fundamental piece of our base design. If I can try to make it even clearer, take your FourBlankCards and change the collaborations between 2 of them. This won't render your original design a failure, but it will change the way you "see" the system.

This was NOT process recovery, but the natural discovery of a better design alternative given some new input. Because we did follow XP practices of DoTheSimplestThingThatCouldPossiblyWork and YouArentGonnaNeedIt in the original design, we were not carrying around any extra baggage and were able to adapt to the new requirement. I don't know what the dynamics are in your environment, but the business domain rules of our customers are quite dynamic and we must have a flexible enough system and process to be able to handle these situations. -- TomKubit

---

I'm sorry, Tom. When you said refactoring was the theme of the entire iteration I took you to mean that the entire iteration was dedicated to refactoring.

Nonetheless, if you're working to an existing CommitmentSchedule, isn't it likely that dedicating N engineers to refactoring is going to slow you down (even though development would of course bog down even more if you DIDN'T refactor)? If the project's schedule is changed negatively, I'd call that a deviation from plan and report it accordingly. Your mileage may vary.

Now, of course, one always refactors anyway, sometimes after adding new function that violates OnceAndOnlyOnce, sometimes before adding something new, so it will go in more readily. But it sounds in the case you describe as if the refactoring was profound enough that it needed to be called out as special. Thanks for the clarification. -- RonJeffries

---

While we feel RefactoringIteration should be in everyone's tool kit, we do agree with Ron that you should use it with care. It should not take the place of good pair level practices. But anyone who is in an environment like ours (VcapsProject) where the requirements do change every 3 months you will find it useful.

There was a stretch on VCAPS where we changed the deliverable functionality on our entire CommitmentSchedule every week. Roll with the punches.

Let me be perfectly clear about my position, since my above attempt at subtlety didn't work. RefactoringIteration is a bad idea. It comes out of the fear that progress will slow or stop without it, but because it interrupts the flow of feedback it raises the risk that the progress that is regained will be in the wrong direction.

My first week on LifeTech we identified a parallel class hierarchy that was getting in our way. After a year and a half the duplication was still there, but much reduced. In the meantime we kept delivering business functionality almost daily. Whenever we could, we made progress towards our ultimate goal. But we didn't let our fear overcome our desire to serve our customers. -- KentBeck

---

Kent, you already know subtlety is lost on me. RolledUpNewspaper will do just fine. You are talking over my head here. Can you tell me more about interrupting the flow of feedback? I am not seeing that here, perhaps I am looking in the wrong place. While we could do as you say and make steady progress to a goal, waiting that long looked counter productive. We are often surprised with a new requirement that is not negotiable due to outside forces. The implementation of one such requirement was going to be a boat anchor. As XP professes we (the developers) took control of one of the four variables: Time. -- DonWells

---

IMHO, the whole thing lies in the antagonism between Refactoring and BusinessValueFirst. maybe EmpowerTheTeamFirst could fix it. i mean, Refactoring, BetterDesign are TeamValue. and there is no room for them, if you keep setting the highest priority to BusinessValue. -- SlimAmamou

---

Last edit September 13, 2007, See github about remodeling.