Worst Things First

From discussions on the XP mailing list I get the impression Kent no longer uses WorstThingsFirst. Is that correct? -- MartijnMeijering

It seemed to me that it's more that he realized that if you're doing XP, WorstThingsFirst is automatically done. It's the same as asking WhatIsTheWaterStrategyOfaFish -- JasonYip

Not at all true. But maybe Kent has a new way of saying what he means.

The notion of doing the "worst things first" is fundamentally a method of avoiding the purely human tendency to put off the hard stuff until later. If you do that consistently then you are in big trouble towards the end of the program.

When I worked for the Navy, large programs were managed out of the Naval Air Systems Command up near the Pentagon. The programs were managed by a Program Manager who was a Captain or Admiral in the Navy. The typical tour was three years. There was a marked tendency for the first Program Manager to push problems that were too hard into the next Program Manager's tour, so the second tour became known as the Dead Man's Tour - because the program was invariably delayed when all the problems piled up. The first guy looked good, and the second guy got a bad fitness report and didn't make Admiral.

If you want to have any chance of finishing a risky program on schedule do the "Worst-Things-First" then you clearly identify the big problems up front. -- RaySchneider

The main reason I like to do WorstThingsFirst, is that the system is easier to change and evolve when it is small. WorstThingsFirst guarantees that you will do the hard stuff before it becomes even harder because of all the baggage you're carrying around. You're right that it also mitigates risk: doing hard/risky things early means if you get into trouble you still have time to do something about it.

Alistair asks: "how do we select what to attack first, up front?"

In the PlanningGame, users assign priority to stories, among other reasons, because of risks they perceive. "System must be able to pay 60,000 people in two hours. High priority." Developers assign technical risk similarly. "System must be able to marshal up to 60,000 networked PCs simultaneously for 1 minute each. High risk." We sit together and build consensus on what is risky, what needs a SpikeSolution, what order to do things. Is there a better way to identify worst things than to get together and work on it? -- RonJeffries

---

I sometimes find the only way I can do the hard pieces is by separating out all the easy pieces and doing them first. After repeated iterations of this, the hard part is stark, clean and pure, and it usually turns out not to be so hard after all. (Sometimes it evaporates altogether.) -- DaveHarris

I'll second this. I can't count anymore the number of times I have have a seemingly hard Smalltalk problem go away after I put in all the obvious objects with their obvious interfaces. On a big project, the issue is different. If you take the worst thing first, the team has not had time to jell yet, they don't have any successes behind them, they don't have the communications paths smooth, yet. So I like to do EasiestThingFirstHardestSecond. The first gets you moving, the second is the test of the system design. Yes, I know we delayed the possibly really bad news by one month, but if we did the worst thing first, and failed, we wouldn't know what was at fault - the problem, the thinking, the communication, etc. I like to debug our development system first before taking on a nasty. -- AlistairCockburn

---

In ExtremeProgramming, WorstThingsFirst is the rule. But so is DoTheSimplestThingThatCouldPossiblyWork. How do they go together? Quite nicely!

Identify a high-risk area. Do some research. Do a prototype. Do a SpikeSolution. You'll learn something - usually enough to know that the risk is no longer high. Write a TechnicalMemo on the solution if you can't put it in place immediately. Repeat until risk is low.

In BigBallOfMud, BrianFoote adds a nice paragraph to this discussion, about the use of CreativeStupidity to select what and when. I also like his annotation to the idea of RiskReduction, "kicking the nearest, nastiest wolf from your door". This is, to me, still slightly different from WorstThingsFirst. I am finding that my strategy on my current projects is sometimes kicking the nearest, nastiest wolf from our door, sometimes evaluating the worst based on near but not immediate futures, and sometimes opportunistically proactively reaching into the future to change the overall direction. -- AlistairCockburn

---

What I mean by WorstThingsFirst is entirely consistent with this. First, though, you have to broaden what you think of as the problems you need to solve. If you think in exclusively technical terms, then WorstThingsFirst is a terrible strategy, because only by blind luck will you get a confident, gelled team. If you immediately attack a bunch of technical problems that you are unlikely to solve, you're dead already.

Getting the team pointed in roughly the same direction is the worst problem. Then, getting the trust of the client is the worst problem. Then, getting the team and the client aligned is the worst problem. Then, get a good handle on the technology is the worst problem. Doo-dah, Doo-dah... -- KentBeck

---

I've applied WorstThingsFirst for years, just for the very simple reason that it helps me sleep better at night. -- MichaelFeathers

---

Converting between Risk and Opportunity (BigBallOfMud, etc.) can be accomplished with a dash of rhetoric. Opportunity loss is a risk, and the ability to sleep better is an opportunity. The problem with focusing solely on the most imminent peril, though, is that it can lead to a chronic siege mentality, where the time to make things better always comes right after the last wolf has been vanquished. I'll grab the $10,000,000 sitting in the basket on my porch right after I unclog this dang sink. So headache reduction is an opportunity, and so are better factored components. Letting a project erode into a BigBallOfMud is a risk.

Also, kicking a few pups away from the door first can build confidence and experience, where going after the big mean one first can make you a meal.

-- BrianFoote (little bucks)

So perhaps getting stuck in a SimplifiedParadigm for action is a case of FalseSurrogateEndpoint. When faced with a project and different options for where to proceed, I like to remember my ImportantFreeCellLessons.

I'm with Brian. My first reaction to a big technical problem is "Oh, my God, I'm too stupid to tackle this one, I've bitten off more than I can chew!" I address this by picking some small easy problem and solving it. After I've done that, I think "Gee, that wasn't so bad", and I'm ready to tackle one of the hard problems.

Isn't there an EarlySuccesses pattern somewhere in JimCoplien's Organization Patterns?

-- BetsyHanesPerry (who does)

Question is, seems to me, which way do you lean? If you lean to always doing the least risky thing, your last step will be a big one. If you lean to

doing the least risky thing, your last step will be a big one. If you fail to always doing the most risky thing, things get easier as you get closer to the sharp end of the project. And that's when you need things to get easier. So, if you need one easy one to gell the team, OK ... but then? -- RonJeffries

---

A manager of years ago gave sage career advice: when you land at a new company, pick a gnarly problem - one that people have been avoiding - and solve it. It gets you up the learning curve, *and* it gains you credibility and respect, both of which you'll need to be an effective developer and influencer. -- DaveSmith

Maybe there's some kind of a pattern in there: ListenToYourCoworkersFears, or something like that. Points to a CodeSmell, or maybe an OrganizationSmell?

A suggestion lifted from the EvoFusion literature: EnlighteningThingsFirst.

I am troubled. I get the vague feeling that WorstThingsFirst contradicts the EconomicsOfXp, even though they both seem to stand well enough on their own. What am I missing?

-- BillTrost

WorstThingsFirst is about risk. Think of risk in terms of probability - the chance the bad thing will happen, and impact - how bad it is if it does. Highest risk is kind of the product of these two.

Worst things first says address the high-probability high-impact risks, reducing them. This is economically sound, because it preserves the investment (when addressing the risk works), and reduces the loss if you can't in fact get rid of the risk.

EconomicsOfXp says, "... you want to do everything in your power to increase the probability of a project surviving ...", which goes to the same point.

DoTheSimplestThingThatCouldPossiblyWork and YAGNI are also economics rules, both going to minimizing the investment, preserving the project for the longest time, and increasing ROI. Arguments against these rules are essentially economic in nature, suggesting that it will be more costly or impossible to put in the needed function later. In XP we use WorstThingsFirst to select important (risky) issues to resolve, then DoTheSimplestThingThatCouldPossiblyWork to reduce the risk, and use YouArentGonnaNeedIt to avoid "opportunistic" diversions from the overall plan we're working.

-- RonJeffries

WorstThingsFirst is in danger of becoming a tautology. For any criticism or alternate suggestion, the reply will come back, "well, if that is the worst thing, then you should of course do / have done that first". That makes for great project management in hindsight, which is not what any of us need. The question is, how do we select what to attack first, up front? -- AlistairCockburn

---

One view that I like to take is that it isn't strictly WorstThingsFirst, but rather LeastUnderstoodThingsFirst. If there is something I do not understand, I want to do it first since it is one of the things that I can not estimate (and therefore a risk). When I start a project, I don't have any idea what the worst thing will be, but I have some idea of what I do not understand. Forgive me if I just restated SpikeSolution in a different way. -- MichaelFeathers

Not at all, but a good statement of one of the best reasons to DO a spike! -- rj

---

I hope we agree that assessing the impact side of worst things is generally easy. We know whether the world will come to an end if X, we just don't know the probability of X. (If this isn't agreed, let's come back to it.)

The main HeuristicRule we use on C3 to decide on the probability side of worst things, is how big the estimate is and how sure we are of it. If we have no idea how long it will take to do some important thing, it is probably high risk. If we are sure it will take a week, it's probably quite low probability for a risk. If we are sure it will take three weeks, it might be a moderate probability because three weeks is a big estimate for us. If we think maybe it will take three weeks, maybe two maybe four, it's a higher probability. -- RonJeffries

---

On EasiestThingFirstHardestSecond, I suggested doing the easiest thing first so that the team has a chance to meld, to get a success with each other. The unsigned reply (but I lean toward suspecting you, Ron) was: *WorstThingsFirst never said anything about "most difficult technical problem first". It says to work on your worst, most pressing, most likely to cause trouble, most restricting problem first. Early in a project, getting people talking is far and away the worst problem. Doing something easy is a great way to address it.* That strikes me as turning good advice into a tautology. If the easiest thing can be the worst thing, then WorstThingsFirst has no meaning. We might as well say Anything First, and define the choice of Anything as whatever you later discover would have been the best thing to have done first.

I don't think I wrote that, it doesn't sound like me. Cleaned it up anyway, check it now. I think we're in agreement if we only knew it.

The reply also says, *RequirementsAnalysis really is your worst problem first.* I'm sorry, I can't get from WorstThingsFirst, defined above as *a method of avoiding the purely human tendency to put off the hard stuff until later*, and *In XP we use WorstThingsFirst to select important (risky) issues to resolve* to WorstThingsFirst meaning either do requirements first or else do the easiest thing first.

'See if it's better now."

To say *WorstThingsFirst requires that you must be aware, in a very broad sense, of what your worst problem is. Second, you must be aware of when it is no longer your worst problem and have the courage to stop working on it on start working on something else* removes the meaningful part from the advice. -- AlistairCockburn

This objection I don't understand. What part of that is meaningless? Know (by navel inspection or any standard technique) what's bad. Reduce risk with as small an investment as possible. Reassess what's bad. Now it's probably something new, unless risk wasn't reduced enough. Select the worst. Repeat.

Ron, I moved the paragraph I feel is most informative up to the top of the page. That is the one I have seen that gives me an indication of what you consider "worst". If your readership can't figure out the rankings for worse or better, then they can't apply WorstThingsFirst. That policy should be, I believe, distinct from EasiestThingFirstHardestSecond, otherwise WorstThingsFirst becomes the tautology I have been threatening you with.

I really don't want to do the worst thing first, for two reasons: first is, I don't want to risk a failure early in the project. Second is, I really do want a success early in the project. As DaveSmith wrote, early success helps the team meld. "Making the team strong" is - to me - a different idea than "detecting a failure".

I disagree with EarlyFailure being a bad thing. Things do not always work out the way you want it to. If the team is constantly in fear of failing then they will be less likely to try risky solutions that JustMightWork. You as a project manager needs to reasure your team that while the eventual success of the project is paramount, small failures along the project development is expected and they won't be disicplined for trying and failing. -- AndrewRicketts

---

One more thing. All this is mere rhetoric and attempt to stick something into a formulaic phrase. I have a suspicion that if we were on the same project, we

formidable phrase. I have a suspicion that if we were on the same project, we would actually agree quite quickly on the what would be the first thing to build. You might subsequently describe that choice as WorstThingsFirst or BusinessCaseFirst, I might subsequently describe it as EasiestThingFirstHardestSecond, or else we might change our formulations as to what those phrases mean. I detect a hazard here on wiki of getting stuck in what appears to be defending an insufficient rendering of our ideas in a medium not suited to the task. Do you get that sense, also? -- AlistairCockburn

Indeed. It's not unlike the frequent occasions where someone rails at XP because of their interpretation of a catch-phrase whose subject page they've not grokked. Here we know what we're talking about and would surely agree in any real case. -- r

---

I have a suspicion that if we were on the same project, we would actually agree quite quickly on the what would be the first thing to build.

Getting that first thing step right ... the smallest, most valuable step in business or risk resolution terms. You believe you would find that easy (with Ron)? I need to retire! I have never looked back at any evolutionary project without being able to improve that first choice with hindsight. The only choice I've made right since 1986 is to make the first and all subsequent steps short.

-- RichardDrake

If you have a really small team, as I do at my company, you are constantly plagued by interrupts for doing something on a crisis basis. Examples are Marketing wants some small feature added to a product to sell a large order and they want it **now**. Sometimes we have problems that occur because a part has changed and we have to modify our calibration procedures which are computer driven to accommodate the new characteristics of the part. There are a large number of other examples. These all create interrupts.

What we do is a combination of RuleDrivenTasking: Rule 1 is 1) Interrupts Affecting a Shipping Product First, 2) Projects Nearing Completion 2nd, and 3) Standard Priority Development 3rd. A related rule is ClearTheDecks, which is a take-off on a Navy term. Clear The Decks means, don't let a lot of little things pile up. When you have several together, even though they may not really fit into the rules, do them anyway and get them out of the way. If you don't do that they nag on you and hurt your productivity on your main tasks.

-- Ray Schneider

I think we can safely rename WorstThingsFirst to BestThingsFirst and not lose any information content on this page. Do that first which is best to do first. Simply stare at your navel until you decide what is the Best thing to do first. Do it. Repeat. -- AlistairCockburn

I haven't read the XP book, but I do the WorstThingsFirst. WorstThingsFirst is indispensable, at least for the following reasons which perhaps have not already been discussed:

- The worst things can be worst because they ultimately require a paradigm shift in thinking for a solution, and paradigm shifts only happen after your ideas have had sufficient time to germinate and evolve and time has a way of running out.
- Worst things can be worst because they involve completely foreign material, and new material needs time to be digested.
- Worst things can be worst because they touch on or determine the behavior of many other parts, are therefore powerful determinants of everything which follows and need to be done first for stability's sake.
- Nothing needs clear thought like worst things and nothing ruins clear thought like panic, and nothing is as panicking as a looming deadline

-- Chad Pratt

When I don't know where to start, I practice "Choose as if you were to die tomorrow". If I were to produce just one story/one task/one "thing", which one would be the most useful, to the customer, to the team, to the project success, ... This helps me find the starting blocks.

-- Alain Ravet

You guys have all rediscovered TheoryOfConstraints, I think. Goldratt just used different terminology, but he worked from the same viewpoint and encountered the same difficulty described by Alistair above. The problem is how to find the constraint (the worst problem/best thing to do), not what to call it. -- PeterHansen

---

Usually the worst thing has smaller building blocks. Create an initial success by completing these building blocks - then the 'worst thing' is no longer the worst thing, but an easy thing, by virtue of the building blocks. -- Konrad Bloor

---

One way to find the constraint is by locating the task which most strongly activates a tendency towards DisplacementActivity. This isn't necessarily a good strategy, however, if the DisplacementActivity is then used to soak up the time that should be used to tackle WorstThingsFirst.

-- SteveHolden

Is it ever thought that the logical 'first' or 'next' thing to implement is the first or next thing the user will see/use? This makes a lot of sense in certain domains, I think(e.g., programming games), than in others, of course. In a game I'm working on, I started by implementing the splash screen, with title music and everything, because it's easy and fun to do. Throw up a pretty graphic, fade in some text. Then, I added the 'Continue Old Game'/'Start New Game' screen... oh, hey! Look, I can refactor this around, and a design is starting to emerge. Next, I implemented the stuff necessary to begin a new game, putting together a party... and so on. In each case, I wrote some UnitTests, made enough dummy stub classes to make it compile, made the UnitTests pass one at a time, and then... I was done! Then I refactored. (I'm using ObjectiveCee, if anyone cares.) A lovely design is emerging, and I always know what to do next because it's what the player will see next. After the player arranges a party of adventurers, he'll start out at a given point on the world map. Then he'll probably go into the town. Then he'll talk to someone; then he'll buy equipment; then he'll equip said equipment; &c &c &c.

I'd argue that this could even work in, say, a LanguageBridge that allowed code written in an arbitrary scripting language to execute ObjectiveCee code. First implement the adding of classes dynamically via the ObjectiveCeeRuntime; then implement calling of methods that already exist; then implement addition of new methods; then implement a parser to read in files of a specified format(previously, everything was verified to be working by using UnitTests). For a database application, implement reading existing records, then adding new records. And so on. The approach one takes to a problem(, in this case, the order in which one solves certain problems in a big MetaProblem), should be determined by the order of operations the user must execute, at least to some extent.

So, not the WorstThingsFirst - the FirstThingsFirst!!

-- JoeOsborn

I don't think I've ever consciously done WorstThingsFirst except by accident, when the thing I'm doing at the moment happens to be the worst thing.

On a mature project that I have been involved with for a while, I just do UsefulThingsFirst, because useful things hunt me down and bother me twice a day until they get done, while non-useful things are not very social animals

a day until they get done, while non-useful things are not very social animals.

If I have too many things to do, I'll do QuickestThingsFirst. This reduces the overhead of re-sorting my incoming priority queue - a process which consists of dragging the owners of every item in my queue into a room and having them mud-wrestle to determine who I work for next. Often I can do one or two Quick things while the wrestling occurs, so I bring a laptop to these, ummm, meetings.

On a new project that I'm unfamiliar with, if I have no idea what's worst and what's not, and there's no external pressure to do things in a certain order, I just do RandomThingsFirst, because it's no better or worse than any other order. Without specific knowledge of the project and its problem domain, RandomThingsFirst is equivalent to the results of any other sorting function applied to the available data.

Aside: As a worst-case example, I once selected what I thought would be an easy task when I joined an existing project: change the copyright notice on a piece of software to include the next year (I started near the end of a year). My time estimate for the task was 15 minutes (grep the source code for the string that appears in the copyright message, find the current year, change it to next year, build, test, commit, done). Actual time was 4.5 hours. It turned out that the text of the copyright message was in one file, the year range was in a second file, and the message itself was assembled and displayed in three different places in two different languages (plus a "fall-back" mechanism that was employed if one of these mechanisms failed, which featured a notice that was two years out of date). One of these was *intentionally obfuscated* in the source code as some kind of copyright violation detection mechanism, and another was unintentionally obfuscated by being incorrectly spelled ("Copyrihgt"). The year happened to be the same as a magic scaling constant value that was used heavily in the program, so attempting to grep for the year matched 60% of the code modules. After the first two hours of work proved unsuccessful at changing the one instance of the message that I was initially aware of, I degenerated into a pattern of visually scanning all of the code in the project to find things that looked like they manipulated copyright messages. After I discovered the intentionally obfuscated copyright message code, I also reviewed any code that didn't have an immediately obvious function in case it *might* display a copyright message. After all that effort, a few weeks later I discovered a few instances that I had missed. Did I mention that the numeric year value was displayed as the string "19" followed by a text string in some places, and 1900+year in others?

On a new project that I'm starting (in those cases where I'm also the GoalDonor and/or GoldOwner), I will attack UnknownThingsFirst, because I need to know how hard these things will be in order to determine if the project can be done at all.

So it would seem that I actually do WorstThingsFirst (according to the widest definition of "Worst" above), even though I always intend to do something else.

-- ZygoBlaxell

One of the confluences of WorstThingsFirst and ExtremeProgramming comes, I think, in ExtremeProgramming's recognition that *programming is hard*, so we will do as much as possible to make programming as easy as possible. As such, XP will concentrate on making the "worst thing" *easy*, as a fundamental part of the XP paradigm. "Worst things" become straightforward as XP is applied. -- BrentNewhall

---

This conversation reminds me of a motto we used at one of my former employers (I coined it, so I'm biased.)

```
Big alligators, little alligators, then the lamps, then your wife.
```

Reading the discussion above, it seems that there is some intermingling of the concepts of efficiency and effectiveness, swamps and alligators respectively. I have been a big advocate of "clearing the decks" from time to time, also sometimes put as "clearing the underbrush." Yet, a good case can be made sometimes that the slew of minor issues aren't the most important thing. They're important mainly because they introduce noise and overhead in getting at the current, big problem. They're mainly an efficiency drain.

Alligators and swamps are a bit like clearing the decks, I think. It's easier to tackle some big alligator absent a swamp. But when the big guy is gnashing his teeth mere inches from one's vital anatomy, the problem changes. Stay alive first, or the eventual ability to deal with the alligator on your terms not his will matter little - you won't be around to do so.

In practice, I found that we ended up characterizing alligators by here, and incoming as well as big and little. Draining the swamp type tasks increased our agility, and reduced the volume and velocity of incoming reptiles with teeth. It turned out to be an effective metaphor for me in several confused, dynamic situations where we were juggling multiple risks, while building both systems (alligators, if the system is mission critical) or own capabilities (swamp improvements.)

I think that priorities in dealing with risks are at odds between management and staff sometimes because of different appreciation of efficiency and effectiveness. It doesn't matter that this emergency fix of the moment is silly, and ultimately could have been avoided if the current alligator will render better development processes moot if not tended to - whatever it takes. So whacking the big guy with a stick while waist deep in muck isn't terribly efficient. It is effective in terms of living to fight another day, and maybe, drain the swamp eventually. Complementing the risk of pursuing efficiency at the cost of effectiveness, is getting stuck in alligator-whacking mode. You never think about stopping the incoming alligators.

I think the ideas about ordering risks can be sorted into effectiveness strategies and efficiency strategies. You need both, of course. The other important strategy, I think is resources, especially calendar time. Some problems require specific resources and some resources are hard to get. Integration problems, for example, tend to serialize, and tend to take a while to solve, and sometimes take quite a while to solve. So integration problems consume lots of time. The XP policy of continuous integration is powerful in part because it causes a team to "integrate early, integrate often" - it buys calendar time for a class of problems that are often obscure, tend to serialize and take a long time to solve.

-- JamesBullock

As I see it, WorstThingsFirst is deduced from the value of Courage. As many have said here, people tend to avoid scary problems and postpone them. It is important to get these worked out, so people can sleep lighter, knowing that one ogre is already taken care of. You have done a day's work and tomorrow it's only easier.

One must not forget the PlanningGame. It is the answer for those who argue about making first the things that the customer will be seeing first. Also, it means that the hardest thing you do first, is not necessarily the hardest thing in the whole project, but instead in the release cycle.

It is also not vital that the task you start with, is THE hardest. Lazy evaluation and cathegorizing the tasks into easy ones and hard ones and then starting with a random task among the hard ones, usually suffices, as long as the classification is reasonable. This applies especially, when a tasks difficulty is close to impossible to evaluate in advance, and also, when there are several pairs working on several tasks simultaneously -- it'd be silly to ask for the "beta team" to wait until the "alpha team" has got their seats adjusted and worked with THE hardest tasks for five seconds first.

Personally I prefer the EasiestThingFirstHardestSecond, which I'd rather rephrase as: aperitive, main course, dessert. You should have something light first to get you(r stomach) prepared for the main issue. Then you start actually working, starting from the heaviest issue first and proceeding onwards, untill

at the end you only have the little dessert (or choosing fancy colours and stuff) left. You may need to go to toilet in the middle (the crisis based ordering, mentioned by Ray Schneider) and then come back and continue (here I mean that you may even have to make a hack into the running release that you have to fix later, as a task, in the proper XP order), but you should keep your cellphone closed, and not let the marketing people interrupt your business dinner. If you are not working for a hospital, people would need a reality check, but then again, I can appreciate the position of a small firm, whose livelihood depends on customer reachability, although this usually breaks even the 40 hours a week principle, as such customers tend to even call you home on weekends for advice on installing the computer they just bought their kid for birthday present...

I don't agree with the idea of decomposing hardest tasks so that they become easy. Everything can be decomposed into smaller and smaller, until there's nothing but atoms left. Somewhere between the whole project as a single task and the atoms, there is the level of dissecting a task into program code. What often confuses me, is how big epoch should a single UserStory be?

I also don't know how good practice it would be for another pair to start working on a Hard Task, left unfinished by the first team the previous day. It might be useful for recomposing the pairs and have one of the previous day's pairs to work with another set of brains with fresh view to the problem. It is here common with my disliked decomposing of harder stories, where I think that tasks should most always be continuously worked on, until they are finished. It is important for the team to accomplish things. And as long as a Hard Task is open, it may still be unsure, if it's 90% finished, or actually only 10%, when a pandora's box opens.

I guess you can think of the aperitive part (the first one or two easy tasks) as a spike, if you wish. However, I consider it as important as the release parties and ThereMustBeFood as a motivator.

-- TomiBgtMantyla

I feel like the term WorstThingsFirst is a distraction when talking about stories and tasks. When I sit down and plan out a release or an iteration, I want to be doing the thing which adds the most value to the business next. I would hate to spend weeks on the most difficult/worst thing just because it may be sticky, at the expense of other tasks which are far more valuable. -- ErikTennant

The reason to include the difficulty of a task in planning has to do with risk management. By pushing the harder bits to the front, you find out sooner if they can be done (or at least of they can be done by your team), and, if they are necessary to the long-term success of the project, failure means less spent on the bits that aren't going to be used. Note, that both the value and risk should be used in planning.

CategoryExtremeProgramming

Last edit November 16, 2014, See github about remodeling.