A Method for Product Development

Premise

Concentric is a methodology to deliver finished products in the digital age.

Whether you are developing a piece of software, writing a term paper, working on a presentation or video, or anything else which is *digitally revisable*, you have trouble knowing when to stop editing it. "When is it done? Shouldn't I keep working on it? I still have time…."

If you are cooking, building a dog house, making something physical, it is different, because there are physical limitations that dictate when things are "done." You can only cut wooden boards shorter, you can't make them longer. There is no Undo in cooking. The processes themselves have an inherent end point.

As human beings, our brains and habits have not caught up with this fundamentally new ability to *keep changing things* with no obvious downside.

Except that you probably have made something worse, by changing it at the last minute, haven't you? Changing that slide in your presentation 3 minutes before giving a talk. "Fixing a bug" in a piece of software that actually introduced a worse problem.

What follows is a radical proposal for a better way to build products, interspersed with rationale and counter-arguments for all the reasons why at first you're going to think it's a bad idea.

Raison d'être

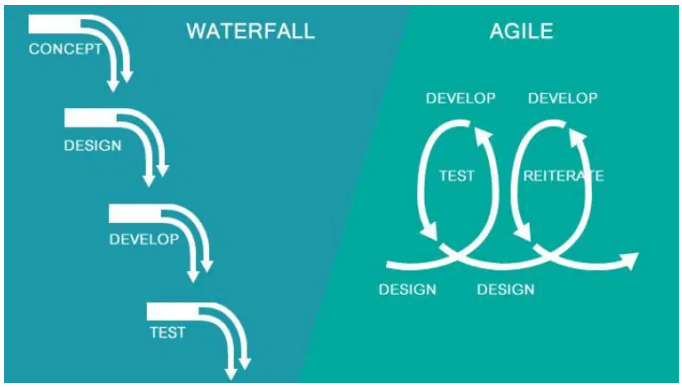
The Concentric method grew out of the observation that products are almost always late and have too many bugs. The *Agile* methodology was proposed around year 2000 as an antidote to the *Waterfall* method, which was perceived to have failed. My belief is that Agile is no better, and perhaps worse, because there are built-in excuses for missing the schedule, disguised as Sprints.

Waterfall

The so-called Waterfall method is basically: "write a spec, then go do it." This isn't a terrible approach, but the scope of the project tended to be 1-2 years of development without re-assessing if the product was on the right track. It is like open-water swimming where you forget to lift your head up and look around to see if you're still on track.

Agile

The Agile methodology is intended for teams who don't know exactly what the end result will be — an iteration model that allows for continuous development. It's right there in the Agile manifesto, in fact: "*Customer collaboration over contract negotiation; Responding to change over following a plan*." The Agile Manifesto is authored by 17 people. I've never heard of any of them. Have you?

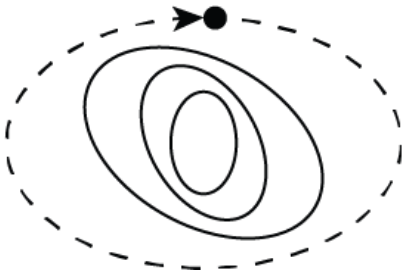


If you actually know what you're trying to build (and you should), Agile is not the right methodology. Following it blindly, chiseling away at mountains of backlog ideas, sprinting from one week to the next … nothing is ever done.

Concentric

The Concentric methodology essentially combines the best of these two models:

1. Actually write a spec and think through what needs to be done.
2. Build what was spec'ed, but the simplest version of it.
3. Finish it — *completely* — before you start adding more stuff.



Mantra

The Concentric method answers these two questions:

4. Is it finished?
5. Is it good enough?

Concept

Approach

The Concentric approach is to iteratively build a product or deliverable *in circles* (or ovals, or single-cell organisms with a nucleus), each of which is complete, functional, polished, and "finished" each time you complete one loop.

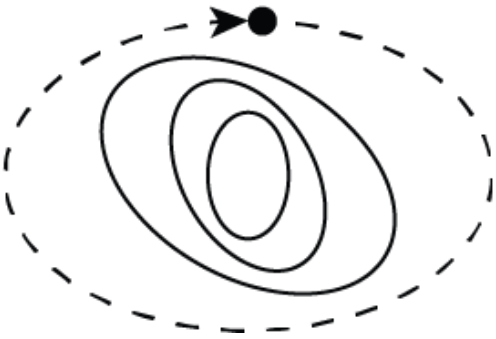
Concentric Loops

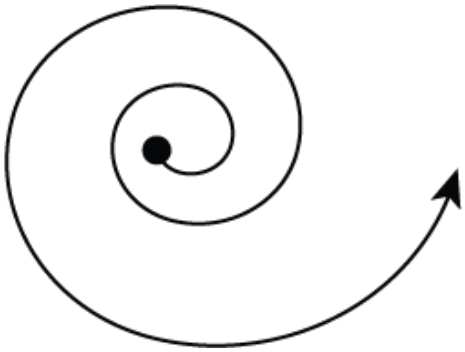
Using the circle as a metaphor, one combines this shape with a process: you *go around the circle*, as it were, and come back to close it. The size and ambition of the circular path is the *loop*. Loop in this case is more a verb than a noun: loop around, orbit, go around the whole product once.

What you don't want is a Spiral, with a rush-to-finish at the very end.

The difference between concentric loops and a spiral is that at any given point in time using the Concentric model, there is the just-previous inner loop that is finished and closed. A spiral is never finished at any given point in time.

You build on the core product. It's very much like shipping version 4.0 of an existing product: the three previous versions are inside it. Except that you don't want a single loop that is the entire version 4.0 — that is the secret: do lots of shippable versions along the way. That is the Concentric method.





Smooth Exterior

Like an egg, all finished products have a smooth exterior, with no visible cracks or discontinuities.

A fundamental tenet of Concentric is to preserve a smooth exterior *throughout the process* — not just at the end.

The basic concept of Concentric as a methodology has three steps:

1. **Specify -** Plan a specific change to the product.
2. **Change -** Edit it, change it (and test it).
3. **Finish -** Bring it back to a smooth exterior and save it.

Another way to think of a Smooth Exterior is that it is "shippable," "finished," "ready to give to someone."

You may not think it's good enough (does one ever?). That's where Concentric gets interesting. You just do it again:

1. **Specify -** Plan another specific change to the product.
2. **Change -** Edit it, change it (and test it).
3. **Finish -** Bring it back to a smooth exterior and save it.

This sounds simple, and it is. What's difficult is actually having the discipline to do it this way, rather than leaving it in an unfinished state during the project, and "getting back to it" the next day, or week, or month.

Inside Out, not Outside In

Most products are built "outside in": the user experience is mocked up first, it looks like a finished product, but there is no interior, and nothing is functional. A physical product is mocked up in foam. A term paper has a strong beginning, the first 4 pages are great, but it has a great big ellipsis on page 5 ….

Concentric is the opposite: you start with a core, functioning product, and improve it until it's good enough — but you improve it in concentric circles, where each time around you bring it back to a fully functional product with a Smooth Exterior. A Smooth Exterior is necessary, but not sufficient — the product needs to actually *do something* as well (assuming it is a functional product) or *convey something* (if it is a work of communication).

No Loose Ends

Above all, Concentric is a mindset. If you know that you have to finish whatever you're doing by the end of the [day, week, month] then you start wrapping up at some point, tying up loose ends, finishing or deleting things that are still underway.

This takes some getting used to, as a methodology, but is not unlike how construction crews work on roadways in high traffic areas:

1. **Specify –** Let's pave the left lane for a stretch of 2 miles.
2. **Change –** The roadway is blocked off, probably in the off-peak hours of night; paving ensues, and the 2 miles are completed.
3. **Finish –** Trucks are moved out of the way, cones are deployed, we create a pathway for the cars, perhaps in the middle lane, and reopen the highway.

No Stubs or Placeholders

Don't put in "stubs" or user interface for features that are not yet implemented or "final content here", or "to be continued."

Here's a simple test to see whether your product is being built in a Concentric way: you should never hear yourself saying:

"Oh, that feature isn't working yet."

…or…

“That section is coming soon…”

You will see later in the Methodology section that there are ways to have placeholder features and sections that aren’t done yet … without compromising the Concentric notion of closed loops with smooth exteriors.

Philosophy

Why?

Now that you have the Concentric concept in mind, it's a good time to bring in some of the philosophy behind this method.

Specification vs Schedule

One of the core tenets of the Concentric Method is that you can choose a feature set for your product (the Specification), or you can choose a ship date (the Schedule). You cannot choose both.

Pick one.

Choosing a feature set and setting a schedule simultaneously has been proven over and over not to work.

People have tried to solve this conundrum for 40+ years. It is not any better than it ever was. Schedules are "padded" to try to account for this, but just as information expands to fill all available disk space, so also do tasks expand to fill all available time. If you give yourself a month, it will take a month. And a half.

Finishing

There are two definitions of Finished:

1. Your Specification has been met.
2. The allotted time has been exhausted.

Running out of time (if you are on a Schedule basis) is like "pencils down" in a standardized test. You are done because you are out of time. The work itself may or may not be in a good state.

To correctly manage a Schedule-based finish, you must be prepared to reduce the feature set or content such that the product can actually be released on the date you set. This is harder than it seems, but it can be done if the Concentric model is fully embraced.

If you are working on a Specification-based finish, then you just work on it until it's done — you don't try to also predict when that will be. If you truly need to hit a specification, then it's done when it's done.

"But I Disagree"

You may be dissenting in the back of your mind: "But in our case, we need to hit a specification *and* be done by January."

To repeat: Pick one.

If you still can't resolve this and you think that you really can pick both a specification and a schedule, ask yourself this question:

"What will I do if the product is not finished on the specified date?"

If you seriously ask yourself that question, and take your own answer seriously, you will realize that there are only three possible answers:

1. We will have to move the scheduled date out and finish it
2. We will have to release what we have, in whatever state it is in.
3. We have failed.

If you answered (1) to that question, then you have chosen the Specification model.

If you answered (2), then you are on the Schedule path.

If you can't budge on either one, then you either ship what you have, or you give up. They evaluate to the same thing: you failed. This is, surprisingly, the most commonly chosen option.

Moving the Goal Posts

A classic thing that happens with most/all product development projects is that in setting both the schedule and the feature set, you run out of time before the feature set is realized, so you move one (or both!) of the goal posts.

1. Revise the feature set / Specification
2. Move the ship date into the future

If you are reading this document, you have been there. It is, of course, okay to move the goal posts, but it does mean that your methodology has failed you. It is likely because you have chosen both schedule and feature set. This cannot be overstated: if you really have a hard ship date, then your feature set HAS TO CHANGE. And vice-versa.

If you truly are able to move the goal posts, then your "hard finish" date wasn't really that hard a date, was it? It was just an arbitrary date that you picked to *motivate the team*. I will argue that arbitrary dates do not motivate teams. Fear of failure motivates teams, but arbitrary dates do not.

If you can't move the goal posts — you are really and truly out of time and money, or you give up — then yes, you are finished, but perhaps not in the way you envisioned.

Methodology

If you're still reading, then you may have accepted some of the premise of Concentric, and wondering just how to go about it. The good news is that it's straightforward and actually easier and less cluttered than other methods, but does require, above all, the correct mindset.

Round Trips

Loops, cycles, concentric circles, you get the metaphor, right?

The Loop

The fundamental concept of the Concentric method is simple: execute a complete change to a product in one cycle, start to finish — with the emphasis on finish. Close the loop before doing the next thing. Each Loop adds to the product, as it grows outward in a holistic way.

The scope of any given Loop doesn't matter: it could be a little thing, like a change to the user interface, or an entirely new feature set, or anything in between.

It is a round trip or a Loop because you end where you started: with a fully functional product.

Inner Core

But what's the innermost Loop? Where do you start?

To bootstrap a Concentric project, you have to start somewhere — think of the simplest possible instance of your project that will have a Smooth Exterior. A minimalist, but complete product. "Not quite good enough," perhaps, but complete, with a Smooth Exterior. Some examples follow, as this is a critical thing to get right:

Software Examples

Many of the people reading this will be software engineers, so we will start there, but it is not the only meaningful application of Concentric.

Web App

Your app has a correct URL, a real server in place, a working home screen, is connected to the DB as designed, and does one thing correctly. That is, it has a front end, a back end, and is functional.

Consider this carefully. This does *not* mean that it "runs on your machine", or has hardwired parameters, or has a wireframe, non-functional user interface. It *does* mean that the product is finished, works with the final architecture in place, etc. So if you're planning to deploy the web app on an AWS server with a Mongo DB and an Angular server … do that *first*, not last.

It does need to do something, to be a valid Inner Core. Maybe create a user login (email address, password, validation, etc). It doesn't matter if the functionality is

extensive enough that someone would actually want to use it. It just matters that there is a *final-form* feature, and it works correctly all the way through to the DB.

Movie-Editing app for MacOS

The product has a basic feature set, can capture video from a camera, edit it, and make a finished product out of it. Copy/Paste and Delete are working, so a movie can be reduced in time (the main editing paradigm) and get rid of sections that suck.

iMovie 1.0 was developed and shipped in less than a year, on a Schedule basis: it was going to be done for a January MacWorld release, no matter what. There were several features in progress in November that were removed (hidden) so that the product could ship on time. They were unhidden and made available in later releases of the product. Note that for this team, a Loop was typically one day, and the app would be released to testers every day, even early in development where it didn't have the full feature set that was anticipated.

Other Digital Examples

The idea of an Inner Core for a product that is not software is a little different. Here are some examples.

PowerPoint Presentation

A presentation is a digital deliverable, and can be developed with a Concentric approach. Start with the introduction and the conclusion, and maybe one or two content slides. Like this:

1. I'm Glenn Reid, and I'm here to tell you about the Concentric method
2. The concept is simple: you create a whole, minimal product first.
3. Then you add to it, iteratively, and make it whole again each loop.
4. If you have the discipline to actually finish it each loop, that's all there is to it.
5. Thanks for listening.

A Term Paper

A written piece like a term paper is just like any other digital work, and can be constructed with the Concentric method. You write the beginning a [probably short] middle, and the conclusion. It reads through coherently, and could be considered done.

Then you go back and add to the middle.

A Book

A long, structured written piece like a book is a bit of a challenge, but it can be done too. The key is that the Inner Core is probably pretty meaty.

Construct the beginning, middle, and end, like your term paper, but probably there are chapters. What's the minimum number of chapters you could get away with? 3 or 4, maybe? How short could each chapter be, but still feel like it's an actual book?

Let's say you write 4 chapters of 10 pages each, plus intro/conclusion, and you have a 50-page book. That's your Inner Core. It's probably too short, but maybe not Maybe you're done, and it's just a short book.

If you're not done (see above), then Loop in another chapter.

Closing the Loop

A critical (and sometimes difficult) discipline in Concentric is actually closing a Loop and bringing the product back to a Smooth Exterior. It's tempting to leave some loose ends hanging, leave some scaffolding in place. You can do that, actually, but it just means your Loop cycle is taking longer than expected, because it's not closed.

For each product/project, it is important to define what "done" is — what are your criteria? The easiest way to do this is to create some *tests*, often in the form of hypothetical questions. Here are some examples, mixing different kinds of deliverables. You have to be able to honestly answer "Yes" to your tests in order to declare a Loop to be done.

1. Can I send this to my boss?
2. Can I ship this to my customer?
3. Can I email this to a reporter who might want to write an article?

A cursory glance at a list of questions like this makes you think, "hey, those are the criteria for when it's done-done."

Exactly.

The definition of "done" for each Loop, the criteria for closing the Loop, is that you would actually be willing to stop and send it off into the world.

Remember at the beginning of this treatise, the two fundamental questions:

1. Is it finished?
2. Is it good enough?

It's critical that at the end of each Loop, it's "finished". Whether or not it is good enough is a separate issue, which often is decided by a third question:

3. Do we have time to do another Loop and make the product better?

This is the core idea in Concentric. Don't just be "finished" at the end (or "almost finished, as it usually turns out"). Be "finished" several times during the life cycle of the release, and pick the best finished product as your deliverable.

It really does work, and it works better than trying to fit ten pounds of shit in a five pound bag, which is the typical product release exercise.

"But, but, but … wait!"

You are perhaps thinking that this is an impractical method, that surely you can't do all that work to finish the product multiple times during the release cycle. Finishing a product is a lot of detail work, fixing bugs, finishing sentences, proofreading … can't I leave that till the end, and work efficiently during the project?

No, you can't. Sooner or later you have to do the detail work, fix the bugs, proofread, and if you leave it to the end, you'll realize you spent too much time on the middle part (the features, the content) and you won't have enough time to polish it up. If you're honest with yourself, it has happened every single time, with every product you've worked on, with every team.

But there are ways to proceed where you can have the best of Concentric while not quite finishing bits and pieces along the way that really do take a long time to do.

Feature Hiding

Feature Hiding is a way to develop a feature in such a way that nobody knows it's there until it works correctly, and lets you maintain a Smooth Exterior while actually continuing to work on something that takes a while. This is the opposite of how most features are developed, but it is a much better methodology.

A feature can be thought of as some Functionality, a way to Invoke it (controls) and some Feedback to the user that it is working.

- Invoking mechanism
- Function
- Feedback that it is working

For a software feature, the invoking mechanism might be a button or a menu item. Often there are parameters that are involved as well that need to be input by the user (check boxes, text fields, etc, that provide parameters to the operation). Collectively these are the Invoking Mechanism.

To see that a feature is working, there is typically some kind of feedback. For software this could be a confirmation dialog, a progress bar, a visual display, a small message in the corner that says "done," or some combination of the above.

In a presentation, you can "hide slides" for unfinished parts of your work. If you are writing a book, you can have a separate file for a chapter that is in progress but not yet complete.

For mechanical products, there are similar concepts. It is a little harder to hide a mechanical feature than a software feature, but it is possible. There are many products already on the market that have "missing" features, where perhaps several models of the product exist, some with the extra controls/hardware, some with a panel that covers where the feature might go. You've no doubt driven a car with a plastic insert covering a hole into which the switch would go, had you ordered the car with that feature. Feature Hiding.

This is another key concept to the Concentric Method: develop *all features* essentially as hidden features, then "un-hide" them at the end, when they work and have been tested. During development you put the Invoking Mechanism and Feedback mechanism into the product, work on it, then *remove* them (hide them) when you commit the changes.

If a feature is invisible and can't be invoked, it can't "break," and the product will be releasable without the feature. This is a very important concept.

If a feature can be implemented and tested in a single day, then you don't need to hide it. If it cannot, then you need to add/remove the feature every day when you commit it.

Pitons

There are many parallels between rock climbing and product development. Technical climbers plan a route to the top, but they also allow themselves to make adjustments in the route depending on what they encounter.

A "piton" is a piece of metal that you hammer into a rock face that is strong enough to support many times your weight. You secure a rope to it, then climb above it. If you miss a handhold and fall, you will [hopefully] fall no further than 2x the amount of rope you climbed above the piton. If you feel good about the progress you've made up the wall, you hammer in another piton.

In product development, a Piton is a metaphor for a safe state to which you can return if things don't go well. You almost never need them (or perhaps you just don't want to go backwards) but they are still a good idea, partly to add structure to your progress.

Pitons: bang them into the wall frequently.

If you are building software, a piton is basically a code check-in to your source code control system. You commit the changes you've made, usually at a known, working state. It is important to do this no less frequently than once a day, and more frequently as you reach micro-milestones of safety.

Example:

> **Concentric:** Change an API call — and *all* instances of the code that calls that API — test it, and check it in.
>
> **Other:** Change an API call, then write the code that you need that calls that API, then go back later and change all the other code that calls it. If more than an hour (or at worst a day) goes by before you've changed all the instances ... you have created a bad situation.

All of the Above

Like any methodology, some of these concepts are optional. You can pick/choose what you want to use, but you may be subverting the value of the methodology. These are battle-tested concepts, not academics. Concentric is not a difficult methodology, but the more of it you follow, the greater the benefits.

It works. It makes sense. Close the Loops, every day, and finish every round trip.

Are you still here?

This paragraph is a breakpoint to see if you're actually reading this. If someone asks you what is Point One in the Concentric Method, you can say "Pitons".

Concentric in Action

In this section we look at some real-world examples and build upon the ideas established so far.

Inner Core

Remember that to bootstrap a Concentric project you have to build an Inner Core. This is, in a way, the "MP" in the popular phrase "MVP". Consider our two key questions from the beginning of this treatise (yes, again):

1. Is it finished?
2. Is it good enough?

The answer to the first question, "Is it finished?", is an MP, or Minimum Product. Yes, it's finished. It works, it is a product. "Is it good enough?" The word *viable*, in the over-used Minimum Viable Product acronym MVP, is the answer to this question. If it's good enough, it's viable.

Example: Web App

Let's consider a web app as an example. It typically has these components:

- A site that hosts the web app
- A page that loads from the server containing a Javascript user interface
- A server back-end API to add/edit/delete pieces of data
- A database in which to store the data

Right Way

A correct, Concentric-based approach would start with the Inner Core and would implement all of those components end-to-end, for just *one small feature*. The database might just contain FirstName, LastName, EmailAddress, and Password. The Javascript front end might be a form that allows you to enter your name and address (create an account), and probably should provide a way to reset your password, delete your account, and whatever else you want in your first working product release. But you'd have a complete system: a form to fill out, a server-side API to call for CreateAccount, DeleteAccount, and EditAccount, and a database to store them. Finish it. Test it. Release it. And go back around the loop again if need be.

Wrong Way

The wrong way to develop a web app is to lay out all of the UI for all the features you want to implement, wireframe them, and have someone work for days/months/weeks to refine the user interface — meanwhile in another room, a programmer is sketching out the whole app, with all the features laid out, but none of them working yet ("stubbed out features"). In a third room, a database engineer is designing a complicated Schema for all the bits of data that the app might ever want to store, and yet a fourth room, a back-end developer is writing an API for all the features to use.

In other words, the entire app is assembled from the outside in, and it looks like it works, because all of the UI is there, but it doesn't work, or worse, you aren't sure

what is supposed to work and what is not, so you can't even test it. "Oh, yeah, you can create an account, but it's not saved in the DB yet," said almost every development team ever.

To truly build an Inner Core for a web app, consider these steps that are typically left to the end:

- Create an AWS instance and give it a domain name (yourcoolapp.com).
- Allow users to create user accounts, edit them, delete their info, log in, etc.
- Create and install an SSL certificate.
- Get payment system working and be able to take credit cards.

Consider this list carefully, and consider the core premise of Concentric, that you have to choose between Schedule and Features. If these features (logging in, changing email address, setting up payments) are mandatory, that you literally cannot ship your product without them, then you should do them first, as part of your Inner Core.

I guarantee that you're not agreeing with this idea, but here is why you are thinking that:

- You hate installing SSL certificates.
- The payment systems are a pain in the ass to set up.
- Editing and deleting user accounts isn't critical functionality: it's just another pain in the ass.
- We don't know what we're going to call it yet, so we can't register a domain name.
- AWS is a pain in the ass, and it's faster and easier if I just use an internal server.

While all those things are true, and understandable, it doesn't change the fact that if they are core, important features, and you can't be finished with your product without them, then they need to be in the first truly "finished" Loop, which is the Inner Core. Think about that for a while.

I also guarantee that if you follow this methodology, you will send me a personal note of thanks later, because all that grungy stuff was done early on, and the end of the project was totally fun and you were done on time.

Note also that in the above list there is a step to register the domain name and point it at your AWS instance (and get the SSL certificate set up). If you finish this, and it works correctly, you can still invoke the "Is it good enough?" clause, and do another Loop later to change it to some other name. It will be a lot easier, because it's already set up — you just have to register a different domain name and get a new SSL cert. At least you know where to put the cert files on the server, and which config files to change, so that Loop will go quickly.

If you leave these small issues till the end, like creating real user accounts, or more commonly, having the ability to edit and delete them, you realize it's a lot of work, and it's not usually budgeted in the schedule. If you get the main features to work but don't bother with user login up front, you can give a great demo to your boss and make it look like you're making great progress, but you will pay for it later, and your project will not be done on time. Right?

Work Backwards from Deliverable

What are we Building?

A Specification is a written description of a product that clearly specifies how to build it, but more importantly, what it will be when it's done. It could be a very simple statement, or a fully-detailed document with every scenario spelled out. If done well, it will be clear when it has been achieved, and it can be used as an argument for including or excluding a feature: "it's in the spec!" or, conversely, "it's not in the spec!"

Example:

> **Steve Jobs Spec:** Build me a movie editing app that can import video from a camera, trim/edit the clips, add a few simple titles and transitions, and save it in a few useful formats. And make it not suck.
>
> **Actual iMovie 1.0 Spec:** A 22-page document with screen shots of sample features, specifics as to formats, cameras, and data models. Many details thought through in advance, features envisioned, boundaries created.

Agreeing on what a product should do, and how it does it, is critical to creating (and finishing!) any product. You and your team and your customers can agree at any stage along the way, but eventually, you must reach agreement. If you don't specify anything up front and just go organically build a product, and people use it and like it, clearly you've succeeded. If you tightly specify a product, build it, and no one wants it, clearly you've failed. There are a lot of in-between scenarios, but if you never nail anything down, you won't know if you're done or not. Maybe you don't care, but then again, you are reading a document on product development methodology, so probably you do care.

When is it Done?

If you start by defining what "done" means, you are working backwards from a deliverable. This is crucial. Do it. Decide if your schedule is important, or your feature set. Choose one. Because that is how you will know when it is done; remember this?

There are two definitions of Finished:

1. Specification has been met. Features are implemented.
2. Allotted time has been exhausted.

Work Backwards

If you know what Done looks like, then you try to figure out how to get there.

Context

Comparisons

The "Waterfall" method of yore is simply one very large Loop, with a long timeline. It has been shown to be deficient through years of trying, though it's not inherently bad. The problem with Waterfall is that with a long timeline (a year or more, typically) by the time you figure out that the time/schedule you allotted was not enough to finish the feature set you spec'ed, you've dug yourself into a hole that's hard to get out of.

The "Agile/Scrum" method of today is no different than Waterfall, in that short "sprints" are arranged (Schedules) but if the product is not releasable/shippable at the end of each sprint, all you're doing is creating more (highly time-padded) milestones along the path to your Waterfall. But worse, you are probably also changing the feature set after each sprint. You are, aren't you? That's what a "backlog" is – all the features you still want to do.

Agile/Scrum isn't inherently bad, either — it is one way to organize a project, and it can be made to work, like any methodology, but it has some weaknesses:

- You are setting specific feature sets *and* the timeframe in each sprint.
- Focusing on short-term goals can cause you to lose the big picture.
- If the feature set is changed as you move tasks in and out of the backlog, you are moving the goal posts a little bit every day.
- Focusing on short term goals provides a feel-good framework that creates an illusion of progress but may be masking divergence from major goals — like actually finishing the product, and having it be cohesive and great.

There is an adage among writers in expository writing that goes something like this:

- **Introduction:** Say what you're going to say
- **Body:** Say it
- **Conclusion:** Say you've said it

Product development is kind of like that:

- **Scope:** Say what you're going to do (a Specification or a Schedule)
- **Body:** Go do it
- **Conclusion:** Fix the bugs, tie up the loose ends, and release it

Waterfall is a year-long version of this; Agile is a two-week version of the same thing

Concentric is an Overlay

The Concentric Method is, to a large extent, compatible with both Agile and Waterfall: it sits above the tactics, like a watchdog, making sure it gets done. It is a way to enforce milestones such that they are sewn up and releasable, that's all. It can be time-based, or it can be feature-based.

If you use Concentric but have just one very large Loop rather than iterating, then you are actually using the Waterfall method. When you resolve the "is it finished?" and "is it good enough?" questions, you will find out if you were on a Schedule basis or a Specification basis.

If you use Concentric but have many short Loops, and you *actually finish* the product at the end of every loop, rather than leaving loose ends, then you are *also* following the Agile/Scrum method. But if you leave the product in an ambiguous state at the end of Loops, and it does not have a Smooth Exterior, then you are following neither method, and you have the worst of both worlds. Doesn't that sound familiar? That's probably what you're doing now: loosely following Agile/Scrum, but ignoring the fundamental questions "is it finished?" and "is it good enough?"

Don't Let the Tools Dictate your Process

If you use an Agile platform like Jira, you are conforming your product development process to match their way of thinking. It is better to decide on your methodology, then choose the right tools to support it.

Every tool can be configured, and every field in a database can be tweaked to do what you want it to do, or not be used at all. It is the process that matters, not the tools.

Example:

> **Concentric:** A milestone or to-do item or "story" is crafted that matches the *pitons* you set mapping your path.
>
> **Other:** A milestone or to-do-item is created that seems like something that should take two weeks to accomplish.

Why is Concentric Better?

"It just sounds like more work, but not better than Agile/Scrum"

Concentric is a proven way to deliver a product on time. It forces you to maintain a product in "ready to go" state at all times, or at least with predictable cycles between states of ready-to-go.

The reason it works is derived from human nature and countless examples of things that didn't work, processes that left products not ready to ship, and lots of blame to go around.

Concentric is about *accountability*. If the product is brought to a fully finished state with a Smooth Exterior on a predictable cycle, you don't have to rely on the "best guess" of individual contributors.

Human Nature

Scheduling

If you ask a team of people to estimate how long it's going to take to do something, you get a combination of optimism ("I can totally do that in a week") and sandbagging ("but I'd better add a week of padding"). Neither is a good way to design and schedule a project.

Fun vs. Tedium

It's always fun to create something new, think big thoughts, or tackle interesting problems.

It's not much fun to fix bugs, make design compromises, make file formats compatible with previous versions, get Undo to work, settle for a cost limitation, or do proofreading.

When a deadline approaches, which do you think is going to get neglected?

Take-away Concepts

- **Iterative:** The process is inherently iterative, but not [necessarily] on a fixed time basis.
- **Mini Releases:** At the end of every loop, product is "released," or at least it could be, because it is finished.
- **Schedule vs Features:** Is the feature set critical (MVP)? or is the schedule? Pick one.
- **Closed-Loop:** Product is fully functional and "finished" at the end of every loop.
- **Feature Hiding:** Unfinished features are hidden/disabled until fully functional.
- **Pitons:** Analogous to rock climbing, pitons are placed for backing out changes.

Project Management

Teams

In most product development environments, there are the primary "doers", and a cloud of stakeholders around them who worry about whether it will be good enough, or done on time. Project management is what that cloud of people does to try to ensure a good product.

Usually, the stakeholders make the typical mistake of setting forth the requirements as well as the deadline for delivery, and usually, the product meets neither. The crux of any methodology needs to be to provide enough transparency that the correct decisions can be made to cut features or slip the schedule, depending on the goals.

Evaluating the Status

Once a Loop is established (either time-base or spec-based), project management boils down to convergence on the end point. Evaluating the status of the project during the Loop is the worrying process, but also provides an opportunity to adjust if things aren't going well.

Closing the Loop

The beauty and simplicity of the Concentric Method is the *convergence* on closing the Loop. Earlier we mentioned working backwards from the defined end state. This is where it comes into play.

Keep Asking the Question: "Is It Done?"

The key to working backwards is to stay focused on the end state. Repeatedly asking the question: "Is it Done?" is how you do that. It's straightforward, easy to remember, and it actually works. The magic is in how that question is answered.

Good Answers: Actionable, Transparent

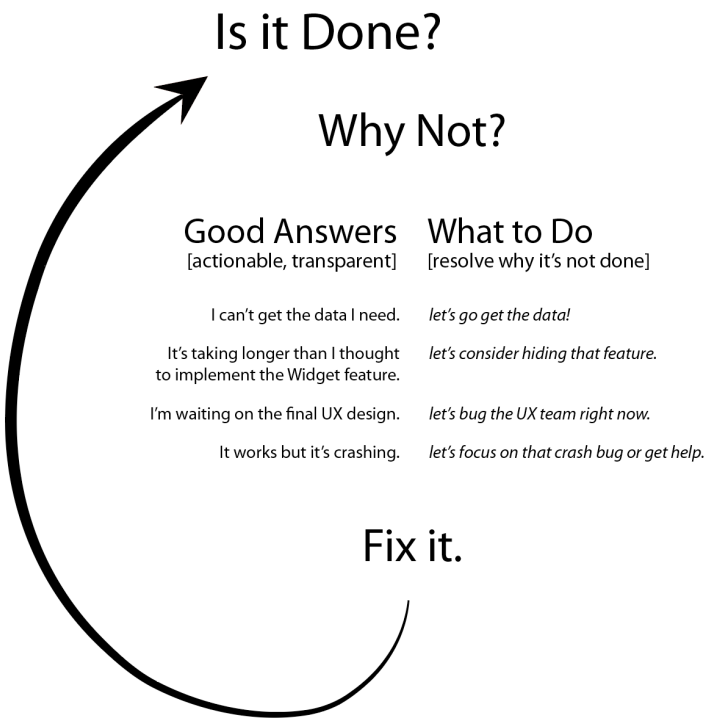
- I can't get the data I need.
- It's taking longer than I thought to implement the Widget feature.
- I'm waiting on the final UX design.
- It works but it's crashing.

Bad Answers: Trust-based, Opaque

- I'm working on it...
- I still have time before the deadline.
- The OS team is behind schedule.
- I am making good progress and will hit the deadline.

Good answers to "Is it Done?" are both *actionable* and *transparent* (a euphemism for honesty). If the reason something isn't done is because you can't get the data — it's clear how to help, and what needs to be done. Let's go get the data!

A bad answer to "Is it Done?" boils down to "Trust me, it'll get done. Go away."



Team-Based Approach

The approach outlined above works well with teams and stakeholders, who implicitly are always asking "Is it Done?" The key is to insist on Good Answers to that question: actionable, transparent reasons why it's not done.

It is also the best way for an individual (engineer, writer, pitch-deck-preparer) to converge on being done: to continually ask yourself "Is it Done?" It keeps you focused on the end result, rather than staying on something that's fun, or adding features, or diverging in your thinking.

The Question Forces the Design

An interesting and hugely important aspect of this approach is that if you *can't answer the question* "Is it Done?" then you have not done a good job of setting the parameters of your project, and you are not applying the Concentric Method.

Methodology

Designing Loops

A Concentric loop is a start-to-finish plan for a single iteration of the product. Think of it as a *version* of the product.

The Concentric approach is to iteratively build a product or deliverable *in circles* (or ovals, or single-cell organisms with a nucleus), where it is complete, functional, polished, and "finished" each time you complete one loop.

The basic concept of Concentric as a methodology has three steps:

1. **Specify -** Plan a specific change to the product.
2. **Change -** Edit it, change it (and test it).
3. **Finish -** Bring it back to a smooth exterior and save it.

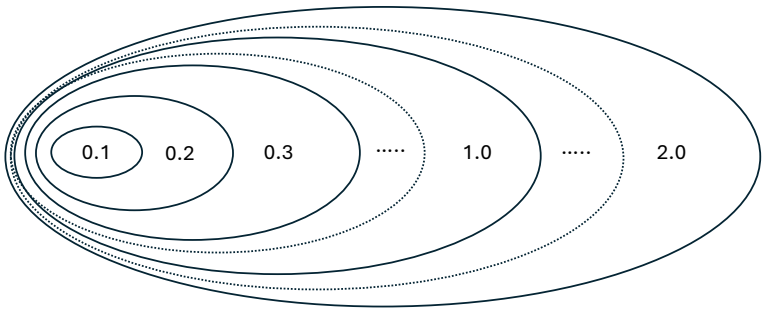
So to design a Loop, you decide (Specify) what you're going to do. This is similar to a Sprint in Agile methodology – a set of things you intend to do in the most immediate time frame.

Releases

When you Finish a Loop, you *release* it. The audience to whom you release it may vary depending on how close you are to finishing the whole project, but the idea is that you make it available to *other people* in one form or another. You commit it to GitHub; you release a build to the dev server; you release it to QA; you ship it to your customer.

Versioning

By custom, using a **1.0** rubric for numbering releases, any release that is *internal*, not intended for public/customer release, is less than 1.0 (e.g. 0.1, 0.2, 0.3…).



There really is no difference between, say, Release 0.96 and Release 1.0 – in terms of intent, methodology, or smooth exterior – because you have crafted every release according to the Concentric criteria, and every release is *Done*. Whether or not it is *Good Enough* is what you have to decide.

It is common practice (and even recommended) when you decide to release a product *to your customer*, that you just take a release that you have decided is Done and release it one more time with no changes – just a new version number. So you take 0.96 and release it as 1.0.

Some groups manage this by having both a version number and a build number, so it might be "v1.0 (build 793)".

The most important thing about versioning is that you *bump the version number* whenever you change anything. Anything at all. The color of something. A version number lets people understand that something has changed. That's all it does. If you're concerned about burning up a version number for some insignificant change, remember this: *integers are cheap*.

Release Process

In large organizations, there are entires teams dedicated to Release Engineering. They take daily source code changes, roll them up and release them – somewhere. The *where* depends on who the audience is, but the process is the same other than destination.

3. **Known Location**: release it where people know where to get it.
4. **Versioning**: Make it clear that it's different/new.
5. **Release Notes**: let people know what changed in this release.

This Document

You will notice that the footer of this document contains a version number, and a release date. You will also notice that you obtained it from a known location (probably the **concentric.works** web site). If you're a careful observer, you may also notice that I don't include Release Notes. You may just have to re-read the whole thing to see what changed. I make no apologies for that. But I will say this: if you're reading 0.831, this last section on Methodology is what is new/added.

Thanks for Reading

I often ask people I work with to read this, in order to understand me better and to understand why I don't want to use Jira. I've found that only a few people actually do read it, so I designed in a couple of checkpoints so I can ask a seemingly innocent question to see if they read it or not. So if someone (Glenn) asks you "What is the Main Point of the Concentric Method," you can say "Smooth Exterior."