# Tracer Bullets vs Prototypes

Asked 14 years, 1 month ago Modified 8 years, 2 months ago

Viewed 13k times

38

I've started reading The Pragmatic Programmer, which I am enjoying and learning heaps form but I'm having difficulty understanding the difference between tracer bullets and prototypes. Are tracer bullets like having all of the views of an application there but not yet adding the entire functionality?

Thanks

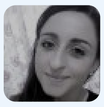
prototype

Share

Improve this question

Follow

asked Oct 28, 2010 at 21:31



Lilz

4,081 ● 13 ● 63 ● 98

5 Answers

Sorted by: Highest score (default)

40

I feel slightly silly responding to a three-year-old question (and one that might be better asked on a different stackexchange), but I think there is still room for improved clarity in describing the difference between tracer bullets and prototypes.

To begin, *they are different things, with different purposes*. But they also have similarities. I will describe their similarities as a means of highlighting their differences.

Much of this is my own take on the concepts, so please read critically.

## How They Are Similar

Both tracer bullets and prototypes are used to make headway on a new undertaking. You are creating something new; an exciting but complicated process. The unknowns are unknown. The features are not well-defined. You aren't sure what components you will need, or how the code must be structured. Tracer bullets and prototypes are similar in that they are *both used* in this situation. Another similarity is that they are both focused on quick, efficient results.

## How They Are Different

The two methodologies are different in two general ways: which unknowns they attempt to explore, and which principles guide them.

## Difference One: What They Explore

Prototypes explore *implementation details*. Will you use an SQL or NoSQL backend? Quickly mock them both up to make a comparison. Can you even supply 10,000 whoozits to the main server on a daily basis? Whip up a little script and give it a shot.

Tracer bullets explore *feasibility of the solution*. Once you *know* the implementation details, pick one self-contained component, and give it a spin. Building a project management app? Single out one piece, like todo lists, and shoot at that.

In product development, you want to iterate as fast as possible, so you should start shooting quickly to see if you're hitting anything. The "tracer bullets" are shareholder feedback!

## Difference Two: Principles

The main principle guiding prototypes is "git 'er done". It's quick, it's dirty, it produces a result (usually information) and gets thrown away. Don't add anything that gets in the way: get the information and get out.

The main principles guiding tracer bullets are completeness and simplicity. The reason for simplicity is straightforward: you want to see results quickly.

Completeness is slightly trickier. It is an acknowledgement of reality: once you have a product that "works", you must immediately put your efforts elsewhere. If the product is shoddy, or lacks documentation, or took shortcuts, you will be living with those flaws for a long time. Make it right the first time!

You may note there's a lot of synergy between the idea of a complete tracer bullet and the other Pragmatic Programmer's idea of "No Broken Windows".

## Benefits

Both prototypes and tracer bullets provide information that will guide the development of a new undertaking. Both emphasize *focus* to get results quickly. Prototypes allow you to explore the unknowns of implementation, while tracer bullets allow you to explore the unknowns of feasibility. In any normal project, you will probably switch back and forth between the two methodologies as you progress. Just keep in mind what they are good for, and use them appropriately!

Share Improve this answer Follow

answered Oct 16, 2013 at 15:37

chreekat
1,024 ● 8 ● 17

The trace bullet approach is to get something working right away. In the book they state:

21

> Tracer code is not disposable: you write it for keeps. It contains all the error checking that any piece of production code has. It simply is not fully functional.

Where a prototype might be throw-away that's not my read of what they're advocating in this particular essay.

The essence to me seems to be, if there is some difficult part of the system try to show that it can be done, before you spend a bunch of time supporting a solution that's never going to be shippable.

Share Improve this answer Follow

edited Oct 29, 2010 at 12:35

answered Oct 28, 2010 at 21:44

Paul Rubel

27.2k ● 7 ● 62 ● 83

---

Your quote seems to contradict itself. If you write it for keeps and contains all the error checking and what not, shouldn't it be fully functional? – Richard Marskell - Drackir Oct 28, 2010 at 21:51

---

Does this mean that it has all the views and hard coded data where functionality is not yet available? – Lilz Oct 28, 2010 at 22:39

---

2 @Drackir, Ugh, I clipped too much and didn't proof enough. I don't have the book with me any longer but will update it tomorrow. The jist is "all the error checking for the

functionality that is implemented". Which in this case isn't much. – Paul Rubel Oct 29, 2010 at 0:24

"If you write it for keeps and contains all the error checking and what not, shouldn't it be fully functional?" Nope. A system can have the necessary error checking, and enough logic to handle "happy path" scenarios representing,say, 80% of traffic, leaving out "rainy day" logic to be implemented later. – luis.espinal Apr 14, 2021 at 16:01

8

We'll start with the easiest one: Prototypes: Prototypes are essentially a way to test the boundaries of what you can and cannot do for your user stories. Essentially it's just a dummy application that's only use is to test whether or not a solution IS possible for given problem. For instance, you would write a prototype if you wanted to see if your server can handle over 1,000 requests at a time. So you write a script that sends a request to your server 1,000 times. The functionality isn't what's important, it's making sure that the task is possible and that you have a clear idea of how to do it.

Tracer Bullets: Tracer bullets are used for "Skeleton Applications" which are just shells of applications that don't contain much (if any) functionality, but walk through the extent of the program's life. i.e. connects to the client, connects to the database, queries the database (but you don't really care about the data). The skeleton application is basically the framework for your application. After you develop the skeleton application, you use tracer bullets to identify the core components of your application. To me, this goes a little beyond simply looking at what functionality you want to implement. As a good practice, and actually what I do at work as a Software Engineer, I think the tracer bullets are more based off of your unit tests (which you should have for any application you design). If you accurately define unit tests that capture ALL of the functionality of a given user story, that works as a very very important tracer bullet for two reasons: 1) When you start modifying the skeleton of your code, if you have accurate unit tests set up for each of the functionalities of a given user story, when you go back to modify the code (which is what your tracer bullet really is for), than you KNOW you aren't breaking any of the current functionalities or any other functionalities already created (because your tests will break if you broke your code somehow). 2) If you are building a new function, just like in the tracer metaphor of the gun used in this article, your test cases are going to tell you how accurate your "tracer bullets" are for a given function. So if you intend for some function to perform a specific action and you have modeled that well in your unit tests, if your code isn't giving that desired outcome, you can easily go back and modify the code to get a more accurate output. So essentially the tracer bullets (encapsulated in your test cases) are going to tell you how accurate to your software solution you actually are.

So, in summation, Prototypes are for testing whether a solution IS POSSIBLE, Tracer Bullets are for testing how accurate your current solution is to the ideal solution you want to create.

For more information on unit testing, I would recommend looking into Test Driven Development (TDD).

I hope all of this makes sense and is helpful. Let me know if you have any questions! Josh

Share Improve this answer Follow

answered Sep 12, 2013 at 16:07

Josh Herr
81 ● 1 ● 2

---

4

See this. I think it's fair to say that tracer bullets are just a metaphor to help explain the value of prototypes. I don't think there's a difference.

Share Improve this answer Follow

answered Oct 28, 2010 at 21:36

Carl Manaster
40.4k ● 17 ● 106 ● 157

---

1

Tracers are about building an end to end, minimal value-able thin but real implementation of a use case, that is used for a quick feedback loop, i.e. whether or not you are building the right thing (hit the target) and based on feedback you can adjust your aim and fire a new tracer bullet.

Prototypes are not necessarily end to end, they are mostly used as a learning tool and are focused on the area you are unsure about and want to explore before implementing all supporting logic, these can be used to do things like evaluate a new library, build a quick UI/wire frame, evaluate how a query would perform if you inject 1 million records etc. Since they are not designed as fully functional, end to end, and are mostly hacked, just to get the quick answers, so you can't easily put them in production.

Share Improve this answer Follow

answered Oct 9, 2016 at 3:55

Usman

**2,345** ● 1 ● 23 ● 18