

What is User Journey Testing?

Testing has been important to me as a professional software engineer for as long as I can remember.

Now, I'm not suggesting that I've been writing tests since the very beginning, or that I always write tests, or even that the tests I write are particularly good. But my first professional development gigs as a software consultant were in Java in the late 1990s, and that was the time that JUnit — a unit testing framework written by Erich Gamma and Kent Beck — was really taking off.

What intrigued me about JUnit at the time was that it was not just a simple testing library — it was a key part of a philosophy called Test-Driven Development (TDD), which in turn was a key part of a larger system of Agile practices called Extreme Programming (XP). Not surprisingly, Kent Beck (along with co-author Martin Fowler) wrote a book about all of these practices called *Planning Extreme Programming*.

Despite the "extreme" qualifier in XP, the practices recommended by XP seemed quite sensible and practical:

- Customers pick the features to be added
- Programmers add the features so that they are completely ready to be used
- Programmers and customers write and maintain automated tests to demonstrate the presence of these features

This felt like such common sense to me at the time that I couldn't fully grasp why all software developers didn't use this approach. If I drop my car off at the repair shop and say, "When I drive above 55 miles per hour, I hear a loud clanking", I fully expect the mechanic to:

- Drive my car above 55 miles per hour so that they can hear (and verify) the clanking sound
- Fix the clanking
- Demonstrate to me, when I pick up my car after the repair, that the clanking is gone by driving above 55 miles per hour with me in the car

Now, if you've been programming for a while, you might be thinking, "That clanking is a bug, not a feature!" And while you're technically correct, what different behavior would you expect if I dropped my car off and said instead, "I'd like you to upgrade my sound system" or "I'd like you to install a new sun roof"? I'd expect the same sequence of events. Wouldn't you?

So then, what is User Journey Testing?

Suppose my client says to me, "I need a website for a software conference I'm running. I'd like to have a page that lists all of the speakers. When you click on a speaker, I'd like that to lead to a page with their biography and a list of their talks." What they just described to me is a User Journey.

I now understand the feature they're asking for. I can add that feature with relatively little effort. But how can I demonstrate the new feature I just added?

As the developer of the feature, I probably manually go through the process of "Go to the Speakers Page; Click on a Speaker; Verify that I end up on a page with the Speaker's biography and list of talks" tens, if not hundreds, of times during the development process. After all, I want to be fully convinced that the process works before I demonstrate it to my client.

But manual testing can be time consuming and prone to error if not done consistently. What if I could automate the User Journey? What if I could write a little bit of code that tests the User Journey in a consistent, repeatable manner? Something like this:

A User Journey test written in Taiko

```
openBrowser()
goto('https://thirstyhead.com/conferenceworks/speakers/')
click('Dr. Rebecca Parsons')
highlight('About')
highlight('Talks')
screenshot({path: 'speakerListTest-screenshot.png'})
```

[Screenshot from the User Journey Test] | *what-is/speakerListTest-screenshot.png*

Figure 1. The resulting screenshot from the User Journey Test

Taiko is an open source Node.js library for testing modern web applications. It is a purpose-built DSL (Domain Specific Language) for writing User Journey Tests.