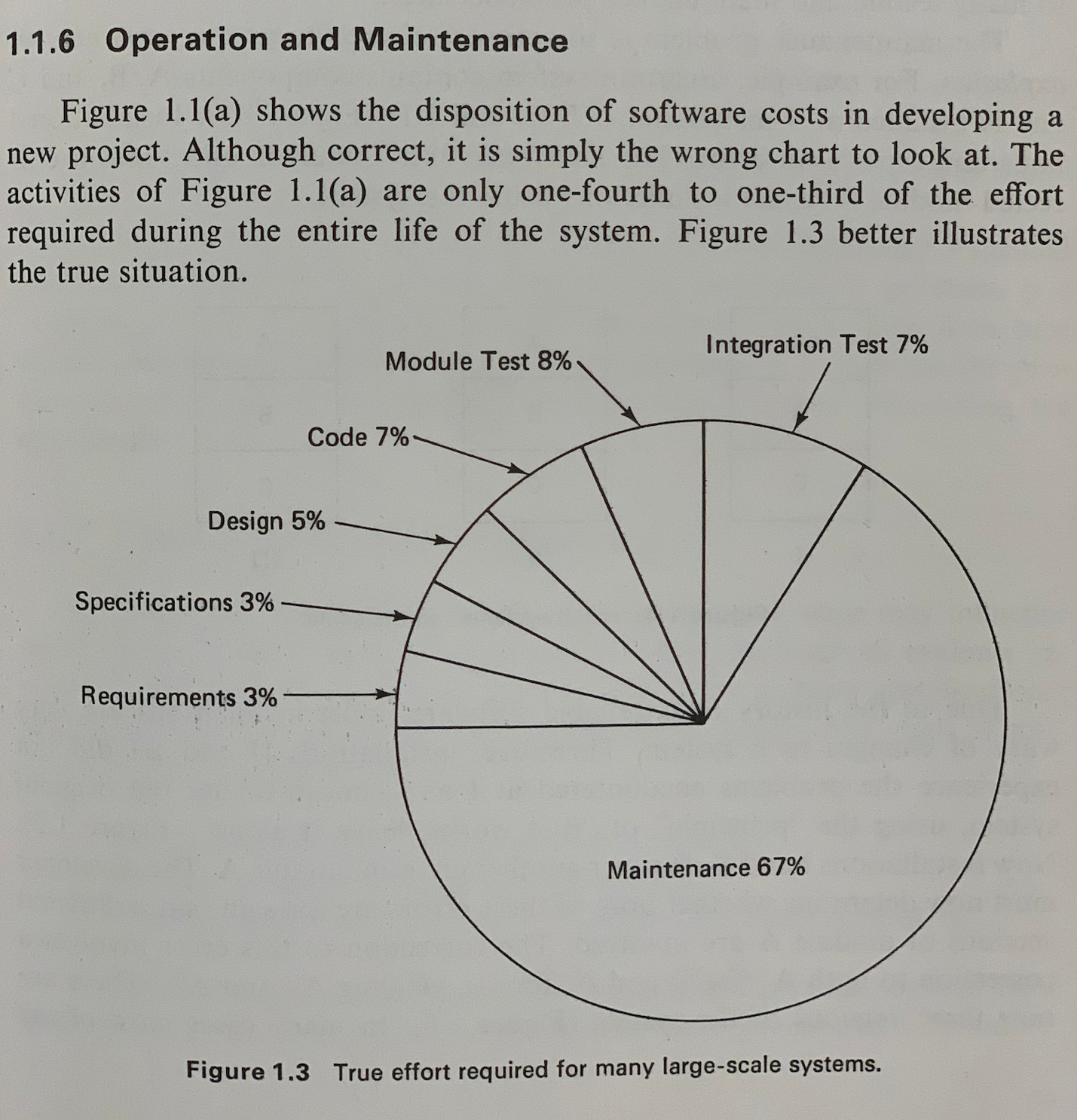
**<https://lepiter.io/feenk/developers-spend-most-of-their-time-figuri-9q25taswlbzjc5rsufndeu0py/>**

**Developers spend most of their time figuring the system out**

I am often asked what I mean when I say that developers spend most of their time figuring the system out. Let’s unpack the statement.

The oldest reference on the topic I know of dates back to 1979 in a book by Zelkowitz, Shaw, and Gannon entitled [Principles of software engineering and design](https://www.amazon.com/Principles-software-engineering-design-Prentice-Hall/dp/013710202X). It said that most of the development time was spent on maintenance (67%).

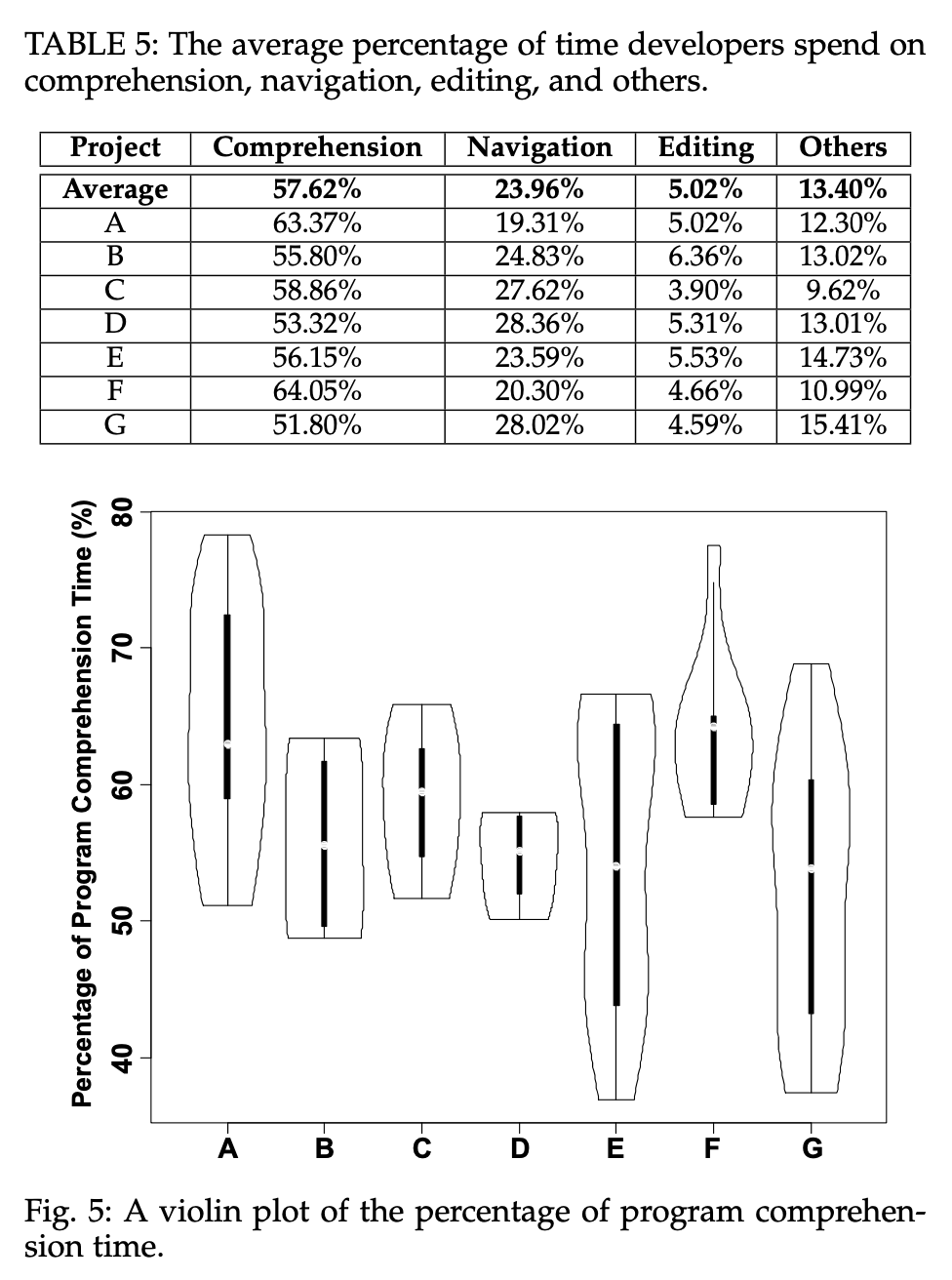
Software development costs (1979)

Granted, the book does not specify how the figure was obtained. Still, it was deemed an important enough problem to attract significant research attention since then.

So, how far are we now, more than 4 decades later?

Let’s look at a recent paper by Xia, Bao, Lo, Xing, Hassan, & Li entitled [Measuring Program Comprehension: A Large-Scale Field Study with Professionals](https://www.researchgate.net/publication/318811113_Measuring_Program_Comprehension_A_Large-Scale_Field_Study_with_Professionals) and published in IEEE Transactions on Software Engineering, 44, 951-976, 2018. This paper is quite interesting in that it describes in great details how the figures are obtained. And it says that Comprehension took on average ~58%.

Now, take a closer look at the results table.

Software development costs (2018)

In particular, take a look at the third column in the table: it says Navigation accounts for 24% of the effort, and that is considered separately from the Comprehension effort!

So, after 4 decades, we can observe that not much has changed beside learning how to measure the “figuring out” time.

So what?

Well, that is the single largest expense we have. If we want to optimize anything in our discipline we should look at this part first. We talk often about how we build systems, but how often do you talk about how you spend the “figuring out" time? If we do not talk about it, it’s not explicit. If it’s not explicit, it does not get optimized.

If we do talk about how the “figuring out the system” time is spent, we notice that people spend it reading. In fact, as the above paper also shows, comprehension is essentially measured as reading! The two are considered as mostly synonymous.

So, how should we talk about how we figure the system out?

Given that not much happened for 4 decades, we should entertain the idea that maybe we should frame the problem differently.

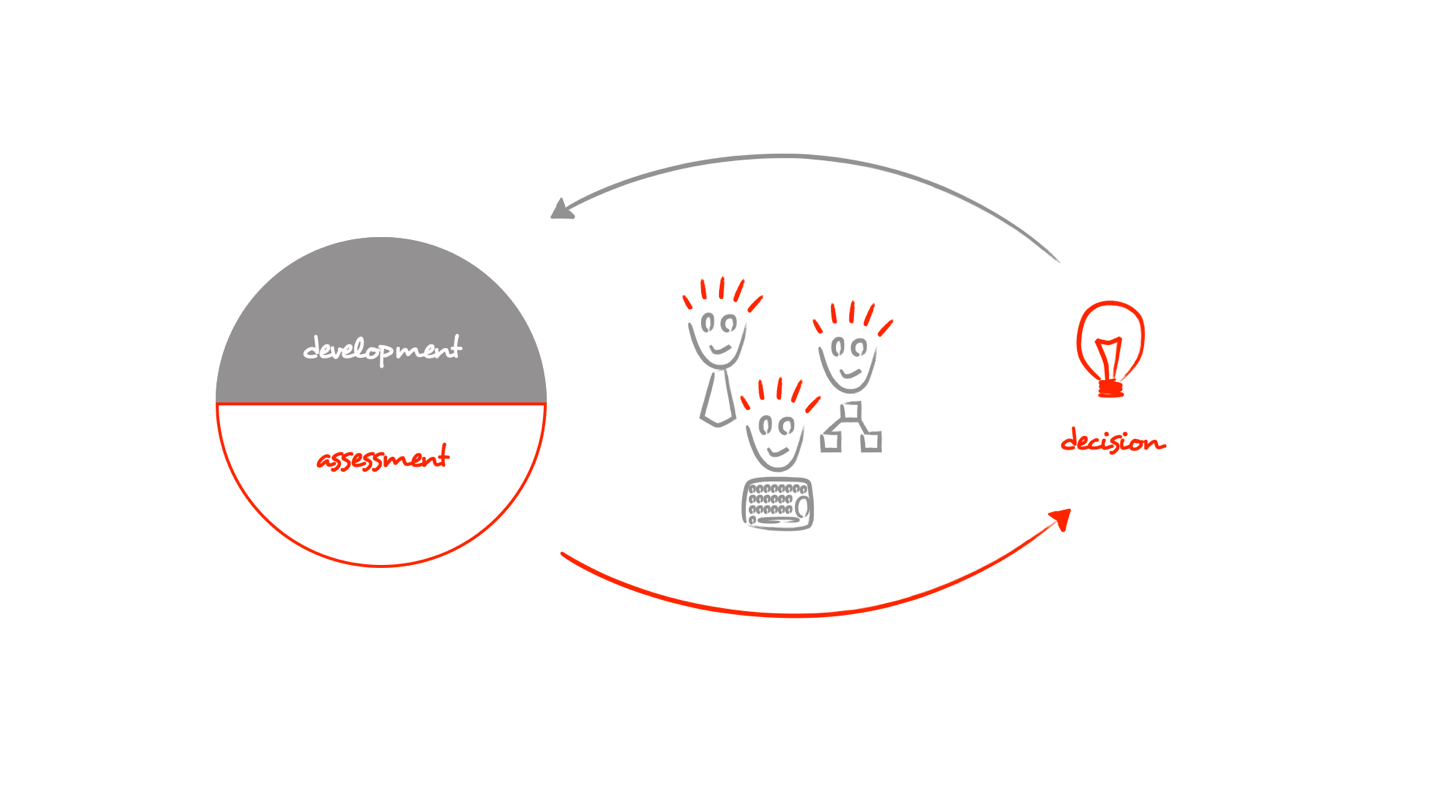
Please, bear with me. This is where it gets interesting. So, why do developers read code? Because they want to figure the situation enough to know what to do next. The intent is important. This is decision making.

A diagram of a diagram of a development process

Description automatically generatedThe figuring out time is a decision making time

From this perspective, reading is just the means through which information is gathered from data. It also happens to be the most manual possible way to do that, so this lends itself to an important opportunity for optimization.

Before you can do something significant about anything, you have to name it. Otherwise it's like with Voldemort. [Many winters ago](http://humane-assessment.com/), I called the effort of “figuring the system out to know what to do next" assessment. And I claimed we should optimize development around it.

Assessment is the process of understanding a situation around a system enough to make a decision

For a whole decade my colleagues and I explored this idea. And it led us to what we now call moldable development.

What’s that?

Reading is the most manual way to extract information out of data. It does not scale and leads to incomplete information & uncertainty.

Software is hard enough. Not knowing what the current system is like should not be an acceptable variable in the equation. A hand drawn picture about the current system is a belief. Decisions should never be based on beliefs. Not in engineering.

Once we accept that systems are data, it becomes obvious we should approach it like data, too. Data science tells us that you first start from the problem and then reason through a tool that matches the context.

A diagram of molding process

Description automatically generatedTools should match the context

As software is highly contextual we cannot predict specific problems. We can only predict classes of problems. To handle this, the key idea of moldable development is that the tool should be moldable after knowing the problem. In this way it can deal with what's important in context, and because of it, it can take care of the boring part of reading. Of course, for this to be practical, the cost of creating custom tools must be small.

I see the flow of constructing custom tools during development, and, ideally, for every single development problem, as the next major leap in software development.

In a decade from now, we should not be measuring the "figuring the system out” in terms of reading. We should occupy our energy with solving actual problems. To get there we should start by talking about how not to read code. We cannot afford not to.

We created [Glamorous Toolkit](https://gtoolkit.com/) to provide a concrete start for the "how not to read code" conversation. Glamorous Toolkit is a moldable development environment that makes it possible to create custom tools about software systems inexpensively.

So, go to [gtoolkit.com](https://gtoolkit.com/). Play with it. Feel #MoldableDevelopment.

And do engage with us: [discord.gg/FTJr9gP](https://discord.gg/FTJr9gP).

Let’s make that conversation happen.