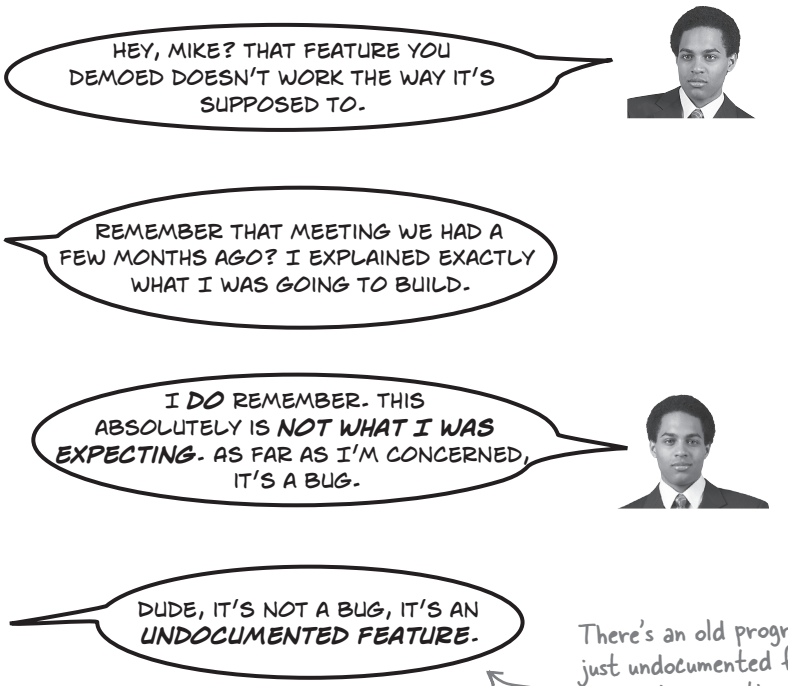
Working software over comprehensive documentation

What does "working" software mean? How do you know if your software works? That's actually a harder question to answer than you might think. A traditional waterfall team starts a project by building comprehensive requirements documents to determine what the team will build, reviews that documentation with the users and stakeholders, and then passes it on to the developers to build.

Most professional software developers have had that terrible meeting where the team proudly demonstrates the software they've been working on, only to have a user complain that it's missing an important feature, or that it doesn't work correctly. It often ends in an argument, like the one that Ben had with Mike after he gave a demo of a feature his team had been working on for a few months:



There's an old programmer joke that bugs are just undocumented features. A lot of bugs happen because the programmer thought the software should work one way, but the users expected it to work differently.

A lot of people try to fix this problem with comprehensive documentation, but that can actually make the situation worse. The problem with documentation is that two people can read the same page and come away with two very different interpretations.

That's why agile teams value **working software** over comprehensive documentation—because it turns out that the most effective way for a user to gauge how well the software works is to actually use it.

BRAIN BARBELL

This little puzzle should be familiar to anyone who read Head First PMP!

Lisa is testing the firmware component for the *Black Box 3000™*. The product is only "working" in one of these scenarios. Can you help Lisa figure out which version of the product has "working" firmware?

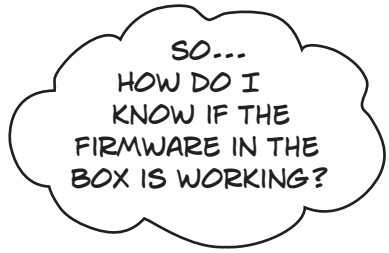
The Black Box 3000™

Here's a hint: there's an important piece of information that we haven't given you. Without it, this puzzle is really hard to solve.

Some might even say impossible!

Scenario 1

Lisa presses the button, but nothing happens.



Scenario 2

Lisa presses the button and a voice comes out of the box that says, "You pressed the button incorrectly."

Lisa, our tester, is testing the Black Box 3000™ firmware, but she isn't sure what she's supposed to be testing for.

Scenario 3

Lisa presses the button and the box heats up to 628°F. Lisa drops the box and it shatters into hundreds of pieces.

Just in case you don't happen to know the term, **firmware** is software that's programmed into the read-only memory of a piece of hardware.

BRAIN BARBELL

EXPLANATION

Here's that crucial missing piece of information that we didn't give you on the previous page: the *Black Box 3000™* is the heating element from an industrial oven. So Scenario 3 is the scenario that demonstrates "working" software:

The best—and sometimes only!—way to tell if software is working is to put it in the hands of the people who need to use it. If they can use the software to do what they need it to do, it's working. But it's not always easy to know exactly what "working" means, which is why agile teams *also* value comprehensive documentation—they just value working software more.

In this case, literally in Lisa's hands. Let's hope that she was wearing heat-proof gloves!

Here's an example of comprehensive documentation that the team actually found useful: the specification for the Black Box 3000™.

BLACK BOX 3000™
Specification Manual

The BB3K™ is a heating element for an industrial oven.

BB3K™ must heat up to exactly 628°F in 0.8 seconds.

BB3K™ must have a large, easy-to-press button.

Sometimes documentation can be useful—like when it's not clear exactly what "working" software is supposed to do.

LOOKS LIKE SCENARIO #3 IS THE ONE THAT SHOWS WORKING SOFTWARE. GOOD THING I HAD SOME DOCUMENTATION TO TELL ME THAT... BUT IT'S EVEN MORE IMPORTANT TO ME THAT I HAVE THE ***ACTUAL PRODUCT*** IN MY HANDS.

Now that she knows what "working" means for this software, Lisa can actually test it.