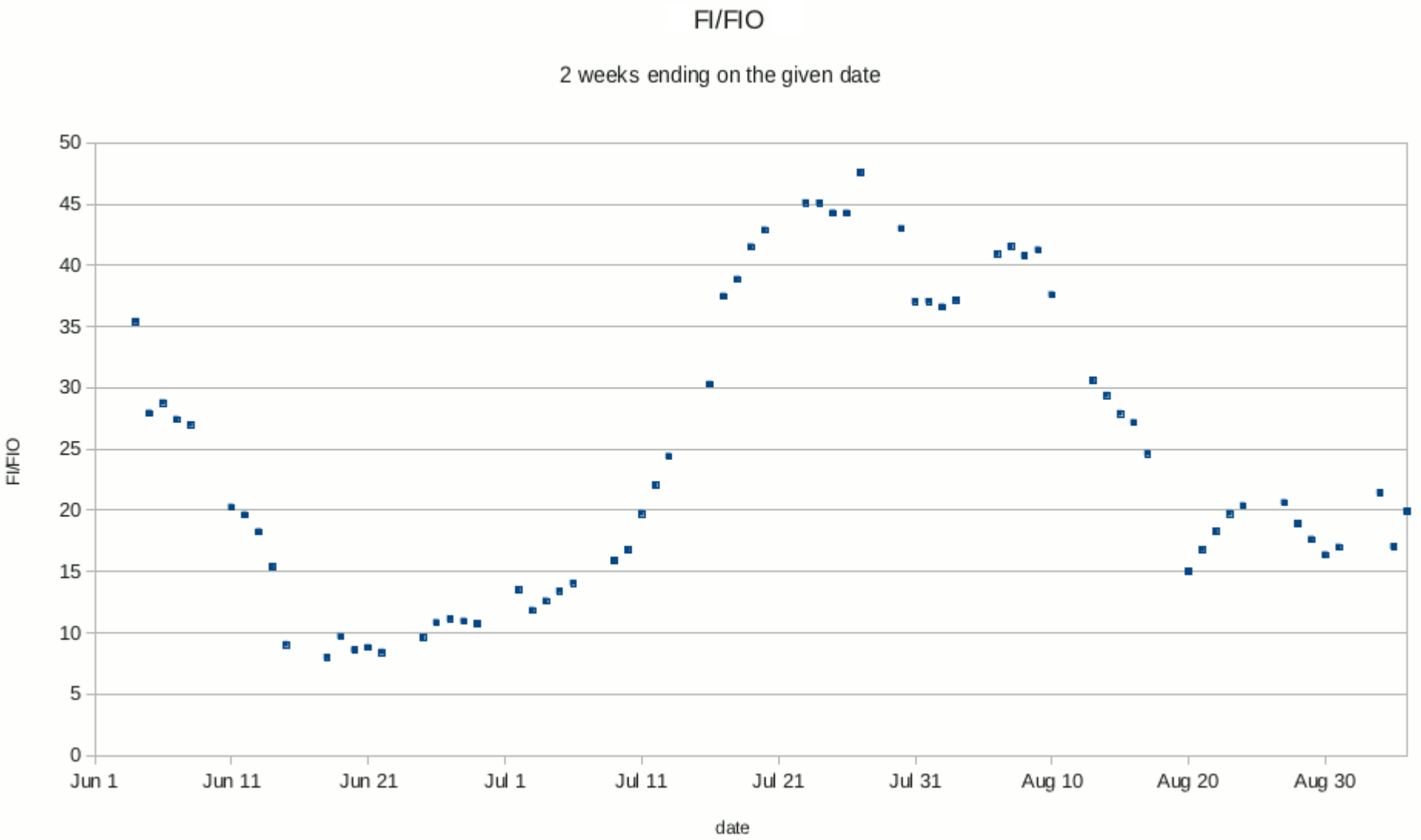
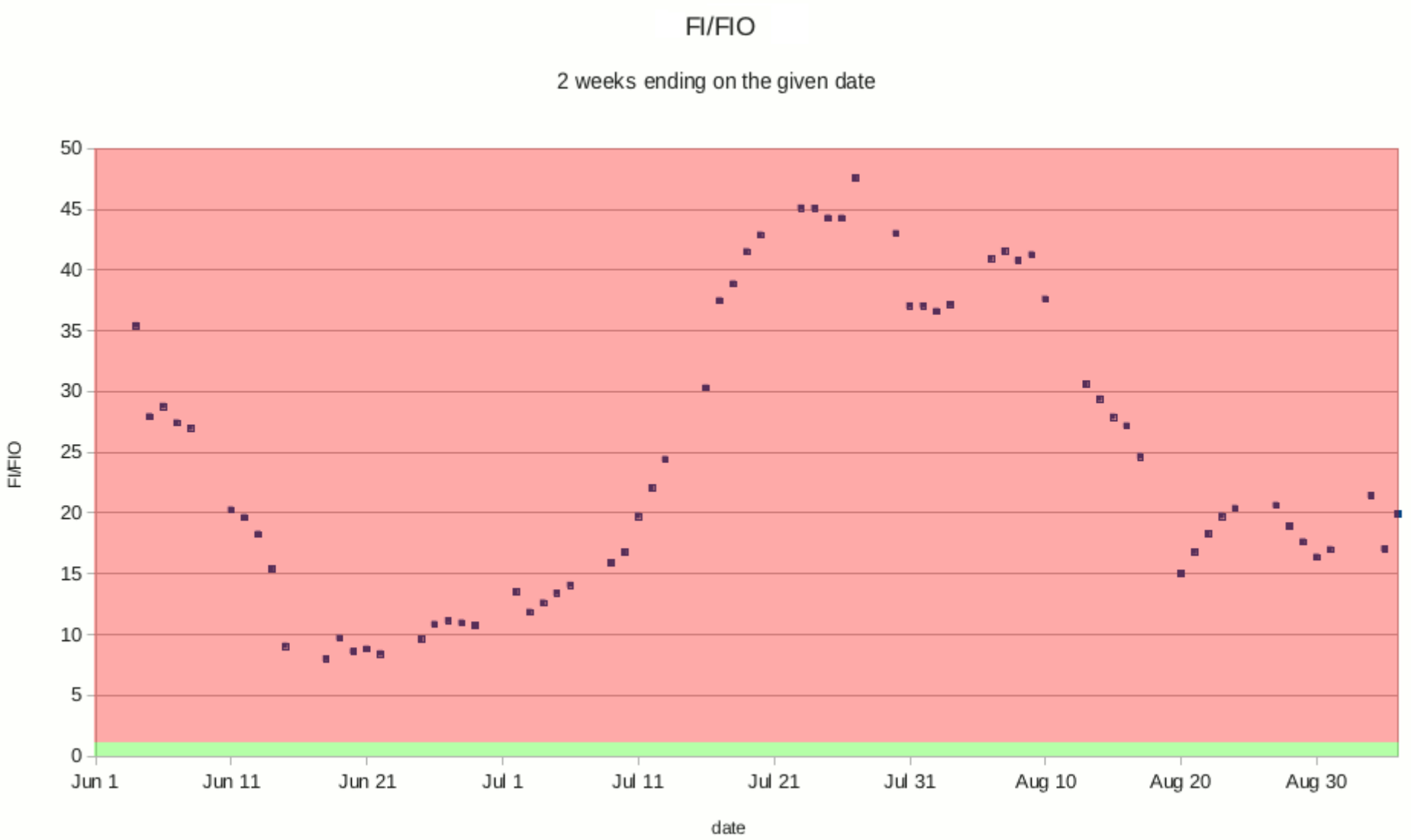
Avoiding Production Fires

Jim Scarborough
Senior Software Developer, Red Hat
August 18, 2014

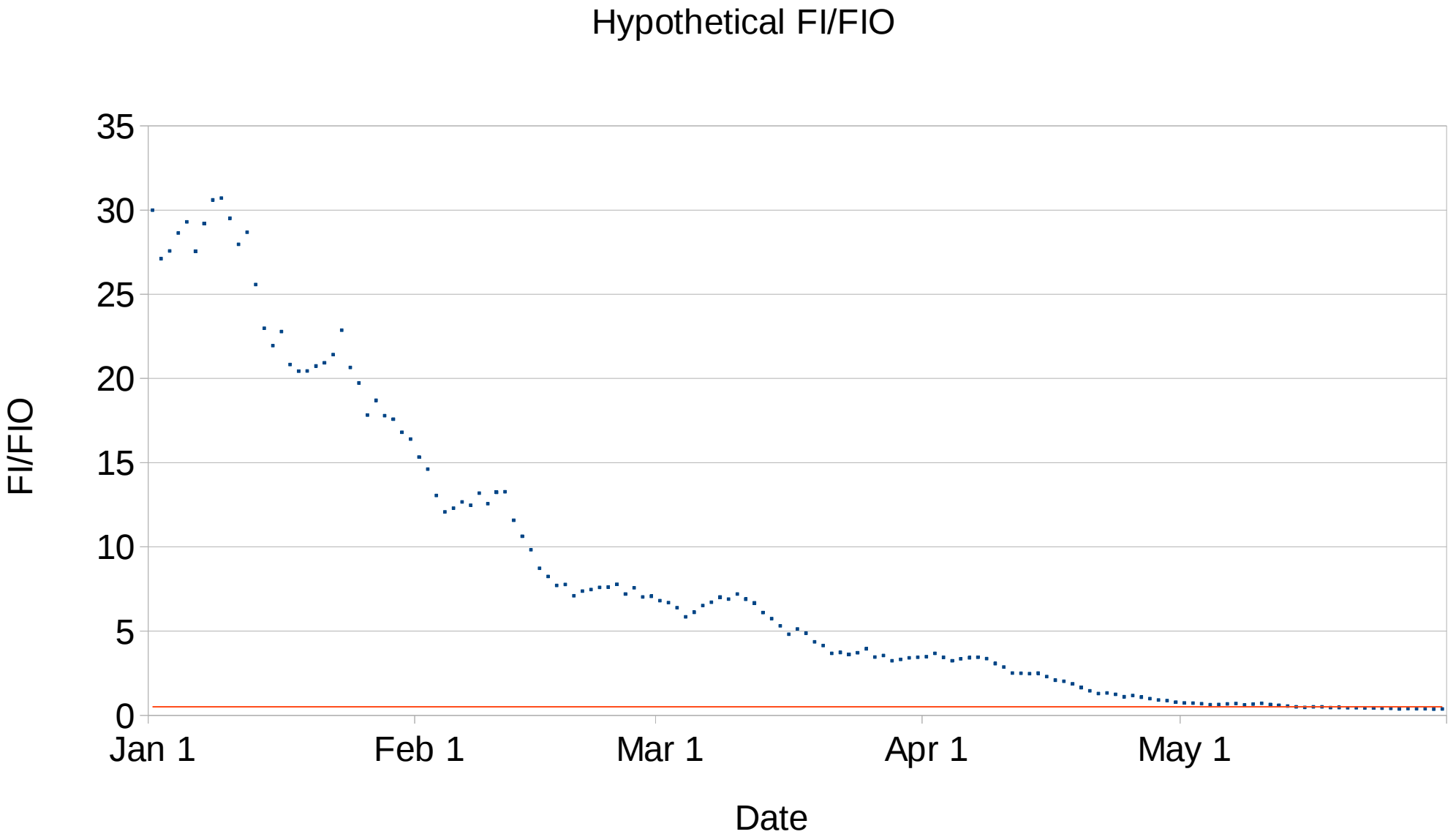
Ready to release?



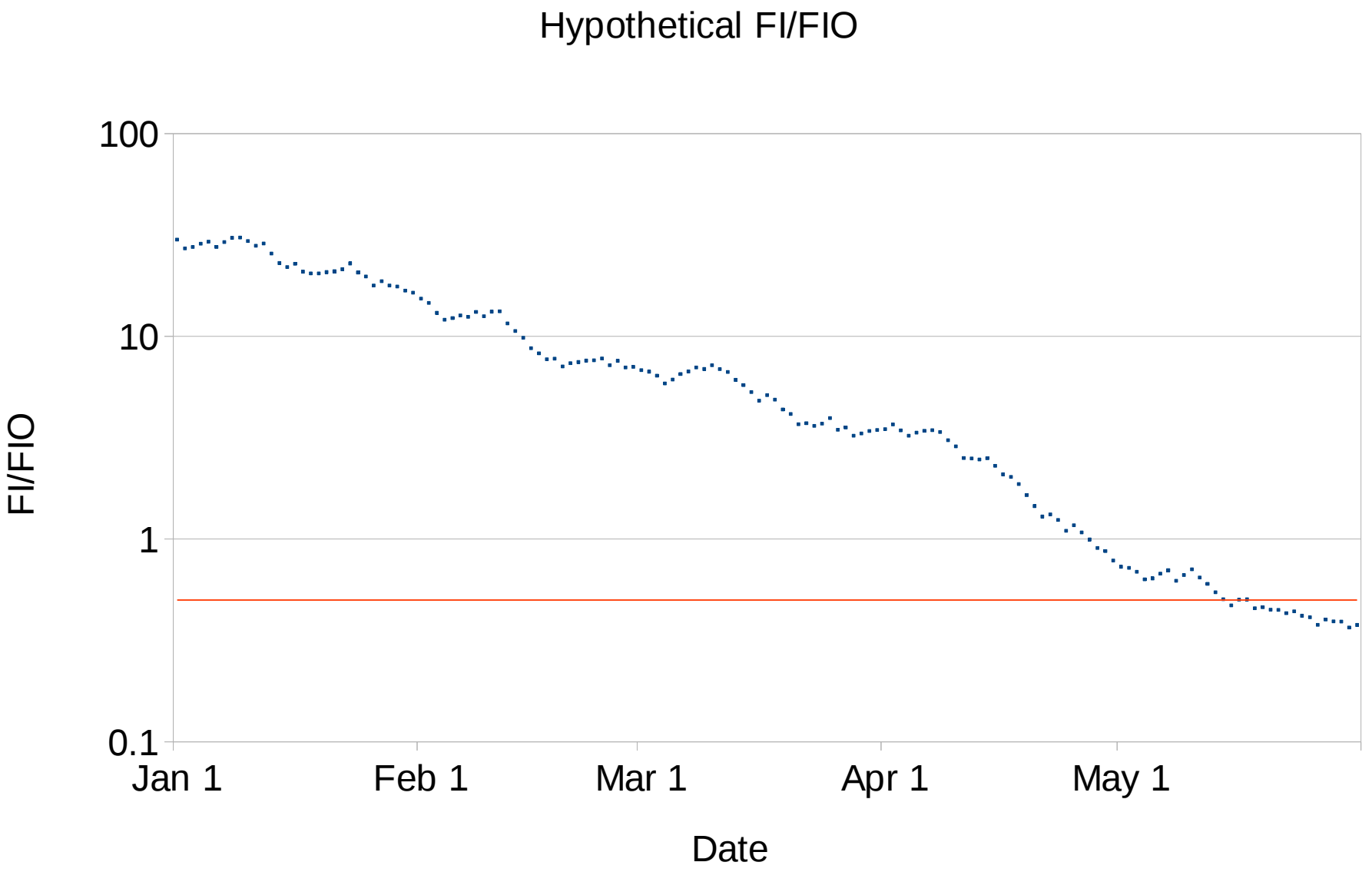
Ready to release?



Ready to release?



Ready to release?



A Practical Alternative to Bug-Free Software

Solution

1. Understand the System
2. Test the System
3. Evaluate the tests

Step 1: Understand the system

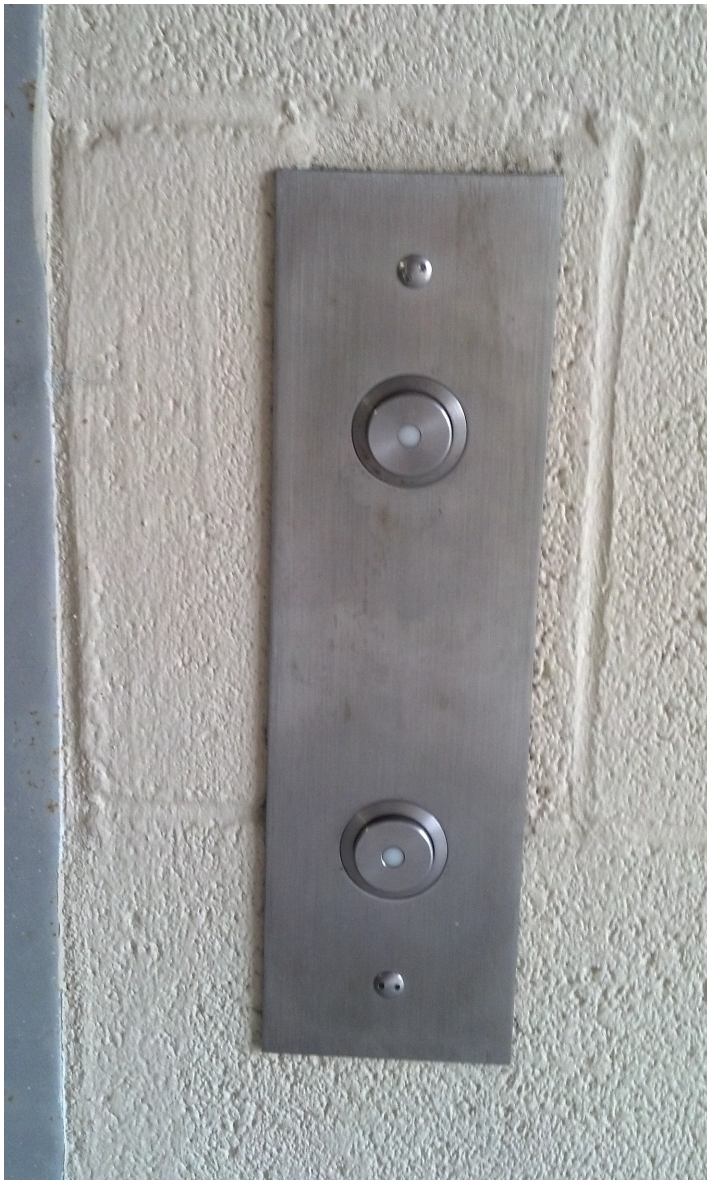
- How good does it need to be?

Event	Failure intensity per 10^6 hrs
Space Shuttle catastrophic failure during launch	53,000
Potentially significant nuclear safety incidents 1984-1987	3,400
Horseback riding injuries	2,860
Incandescent light bulb burnouts	1,000
Deaths at age 35 (per person)	1.02
US Commercial airline fatal accidents	.5
Deaths by fire (US per person)	.0023
Deaths by electrocution (US per person)	.000563

Source: Musa 2004

Step 1: Understand the system

- What are the natural units?
- What's the worst-case failure?
- How much failure is break-even?
  - This is your **Failure Intensity Objective** (FIO)



Step 1: Understand the system – Cost of failure

Cost of death: $1M -- 1 per 10k failures

Typical injury: $20k -- 1 per 1000 failures

Typical repair cost: $500 (parts and labor)

Cost per failure is 1M /10K + 20k/1k + $500

= $100+20+500 = $620

Step 1: Understand the system - FIO

60,000 working elevators throughout NYC [1]

43 reported incidents in 2011 [1]

43/60,000 = .7 failures/thousand elevators

3 minutes/trip, trips solid from 9-5 = 160 trips/day

160 trips/day * 6 days/week * 52 weeks/year = 50k trips/elevator/year

60k elevators * 50k trips/elevator = 3 billion trips

43 failures in 3 billion trips = 1.4e-8 failures/trip

[1] HuffPo http://tinyurl.com/7gksnhj

Elevator functionality

Step 1: Understand the system – Functions

Functions	Per trip	% of trip
come for a call	3	16.66%
go to the desired floor	3	16.66%
open the doors	6	33.33%
close the doors	6	33.33%
alarm bell	0.002	0.01%
fire key	1e-6	0.00001%
maintenance	0.0001	0.00056%

Step 1: Understand the system – Functions

Functions	Per trip	% of trip	Severity	% * severity	% test budget
come for a call	3	16.66%	0.1	0.017	2.4%
go to the desired floor	3	16.66%	0.1	0.017	2.4%
open the doors	6	33.33%	1	0.33	48%
close the doors	6	33.33%	1	0.33	48%
alarm bell	0.002	0.01%	0.01	1.1 e-6	epsilon
fire key	1e-6	0.00001%	1	5.6e-8	epsilon
maintenance	0.0001	0.00056%	1	5.6e-6	epsilon

Step 2: Test the system

- Test everything at least once!
- Allocate the test budget proportionally to the important and/or critical functions.
- The test budget must contain enough to test all the “functions” once.
- Simulate real usage

Step 3: Evaluate tests

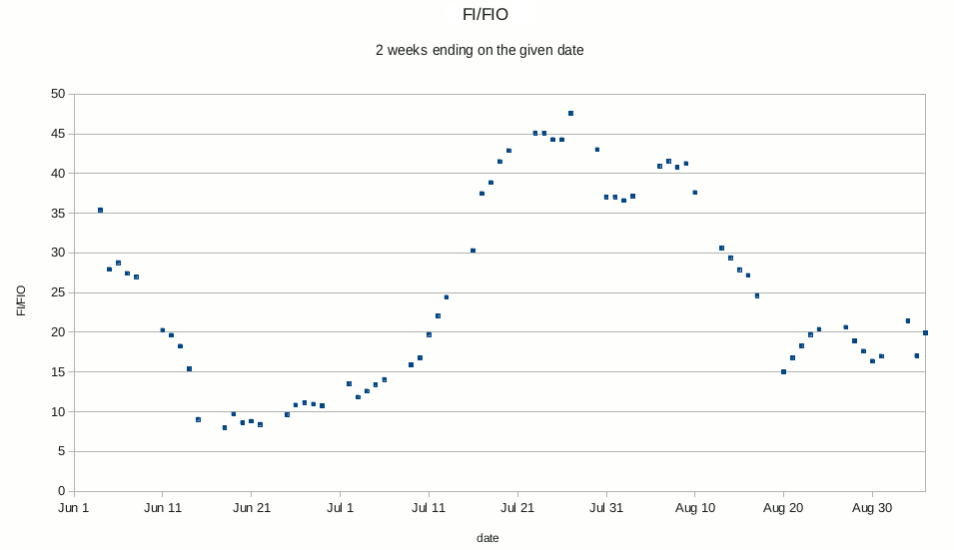
- Compute the failure intensity
- Graph it over time

Measure YOUR system!

- Calculate your FIO
- Measure your failure intensity
- Monitor FI/FIO, release at ½
- Read more
  - Musa, John D. *Software Reliability Engineering: More reliable Software Faster and Cheaper*. 2nd Ed. Author House, Bloomington. 2004.

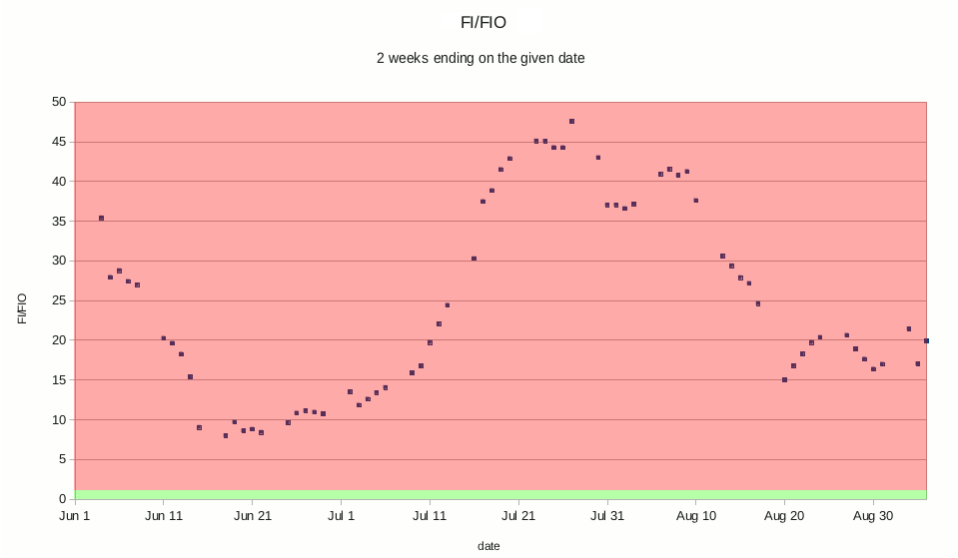
Hi! We've all had software releases that resulted in a flurry of production fires and bug fixes. That's reality – but it's not necessity. Today I'm going to show you how to tell if your software is good enough to release, so that you can reliably avoid those production fires.

Ready to release?

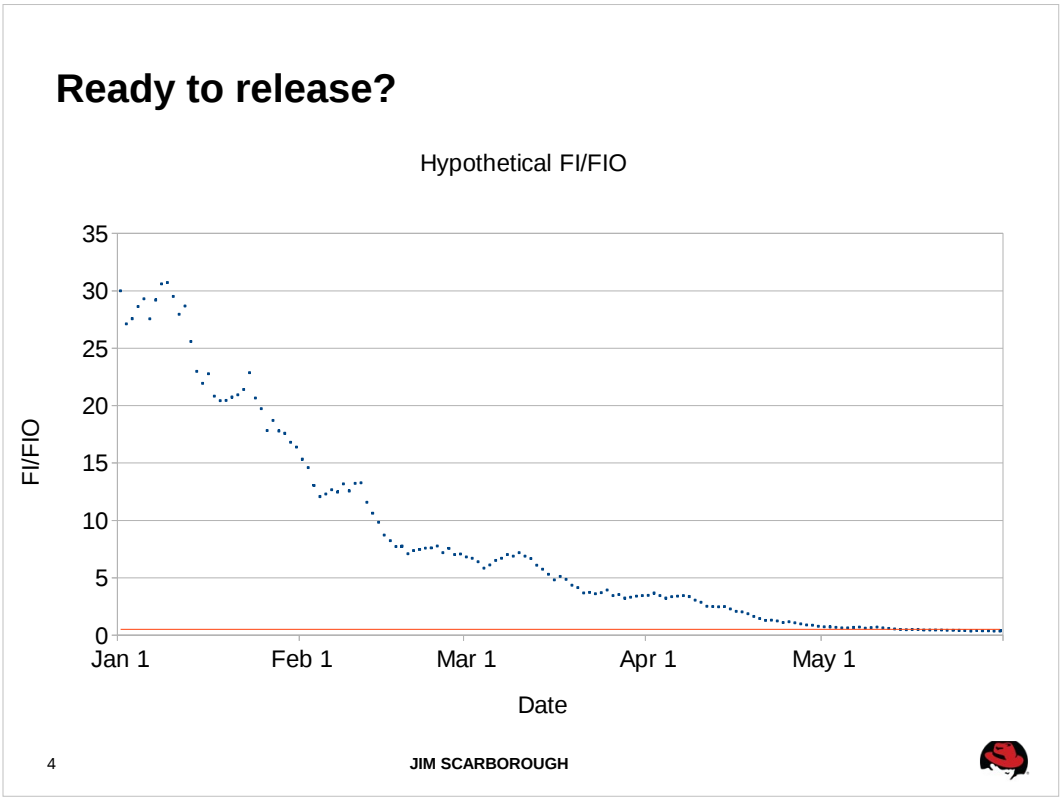


Let's start with the output. This is a graph, over time, of failure intensity over failure intensity objective (FI over FIO). This is from a real project with warts and all. We saw decreasing failures from the start of the graph through about June 21, then failures started to climb – for two reasons: inattention to failures and bad data – this was a system for taking training orders, and it required sessions be available. With fresh attention to the issues in July, things got better. But we were actively developing both the software and the test suite, so failures didn't get back down to late June levels by the end of August. But to the burning question: is this software ready to release?

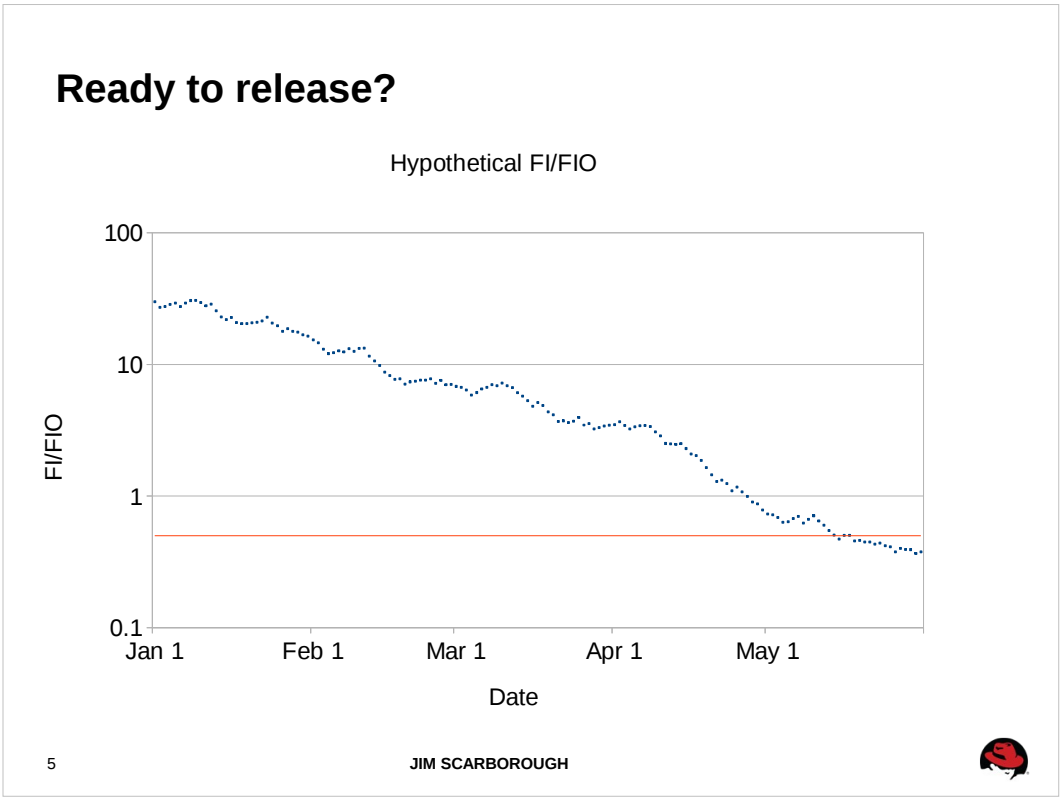
Ready to release?



No! We are looking for a FI/FIO of ½. Never did it approach that on this chart. We want the FI/FIO of ½ instead of 1 to allow for some bugs to go undetected before we release the software.



Slide 4: Here is a textbook version of an FI/FIO chart. The typical version shows an asymptotic approach to zero. If it got to zero, that would be either bug-free or insufficient tests (more on that in a bit). The red line is the FI/FIO of ½, and we see it is achieved in early May.



Slide 5: A log scale presenting textbook values clarifies the margin of safety.

A Practical Alternative to Bug-Free Software

Solution

1. Understand the System
2. Test the System
3. Evaluate the tests

Silde 6: Now that we've seen that there can be a clear line, let's see how to construct it. There are three main steps – Understand the system, where we will compute the failure intensity objective; test the system, to compute the failure intensity; and evaluate the tests to see how the two compare.

Step 1: Understand the system

- How good does it need to be?

Event	Failure intensity per 10^6 hrs
Space Shuttle catastrophic failure during launch	53,000
Potentially significant nuclear safety incidents 1984-1987	3,400
Horseback riding injuries	2,860
Incandescent light bulb burnouts	1,000
Deaths at age 35 (per person)	1.02
US Commercial airline fatal accidents	.5
Deaths by fire (US per person)	.0023
Deaths by electrocution (US per person)	.000563

Source: Musa 2004

Slide 7: What do these failure intensities look like? The Space Shuttle had 53,000 launch failures per million hours launching. Nuclear plants have lots of “potentially significant incidents,” too. But there are safer activities: Incandescent light bulbs burn out every thousand hours. 35-year-olds, arguably a relatively safe bunch, die at 1.02 per million hours, and so on.

Step 1: Understand the system

- What are the natural units?
- What's the worst-case failure?
- How much failure is break-even?
  - This is your **Failure Intensity Objective** (FIO)

To find your failure intensity objective, first you need to know the natural units. Units for a light bulb would be hours (or power cycles). Units for a telephone system would be calls placed. Units for a copy machine would be copies made. What are the natural units for an elevator?

Step 1: Understand the system – Cost of failure

Cost of death: $1M -- 1 per 10k failures

Typical injury: $20k -- 1 per 1000 failures

Typical repair cost: $500 (parts and labor)

Cost per failure is 1M /10K + 20k/1k + $500

= $100+20+500 = $620

How severe is a single failure? One way is to look at the costs of failure. Elevator failures can be lethal, and a death costs about $1 million. But not all failures are fatal. Some are just injuries that cost a few thousand dollars. So how much is the failure cost? The product of the actual cost times its probability.

Now, to figure out our failure intensity objective build an equation. On the left side of the equation is the cost of the status quo, and on the right side is the cost after going live, including failures. Solve for failures and estimate the rest.

Step 1: Understand the system - FIO

60,000 working elevators throughout NYC [1]

43 reported incidents in 2011 [1]

43/60,000 = .7 failures/thousand elevators

3 minutes/trip, trips solid from 9-5 = 160 trips/day

160 trips/day * 6 days/week * 52 weeks/year = 50k trips/elevator/year

60k elevators * 50k trips/elevator = 3 billion trips

43 failures in 3 billion trips = 1.4e-8 failures/trip

[1] HuffPo http://tinyurl.com/7gksnhj

Anothere way to look at the FIO is to take existing data and insist on something comparable. Based on a couple of stats from Huffington Post, we can work out the existing failure rate for elevators in New York City, about 1 in 143 million trips.

Elevator functionality

Slide 11: Next, we make a list of the major functions of the system, and count how often each one happens per natural unit. You see some familiar functions here.

Step 1: Understand the system – Functions

Functions	Per trip	% of trip
come for a call	3	16.66%
go to the desired floor	3	16.66%
open the doors	6	33.33%
close the doors	6	33.33%
alarm bell	0.002	0.01%
fire key	1e-6	0.00001%
maintenance	0.0001	0.00056%

Slide 12: Here are the functions for the elevator. Consider that the elevator trip is not a person's trip, but the car's trip. A trip might make 3 stops and 3 pickups. Then doors have to open and close twice for each passenger. And there's the fire key and maintenance, where a large share of the fatalities happen.

In the first column, we see how often these things happen per trip. In the next column, we see how often they happen as a percentage of all functions.

Step 1: Understand the system – Functions

Functions	Per trip	% of trip	Severity	% * severity	% test budget
come for a call	3	16.66%	0.1	0.017	2.4%
go to the desired floor	3	16.66%	0.1	0.017	2.4%
open the doors	6	33.33%	1	0.33	48%
close the doors	6	33.33%	1	0.33	48%
alarm bell	0.002	0.01%	0.01	1.1 e-6	epsilon
fire key	1e-6	0.00001%	1	5.6e-8	epsilon
maintenance	0.0001	0.00056%	1	5.6e-6	epsilon

Slide 14: The severity column shows how bad a failure in a particular function is, and the product of those tells us how much of our test budget to allocate to testing each function. Obviously we won't skip testing the fire key and maintenance modes, so we allocate a test to each.

Step 2: Test the system

- Test everything at least once!
- Allocate the test budget proportionally to the important and/or critical functions.
- The test budget must contain enough to test all the “functions” once.
- Simulate real usage

Slide 13: Now we need to plan testing. Test all your functions at least once, and allocate the test budget across the functions in proportion to their usage and severity of their failures. Ideal tests for this purpose simulate real usage.

Step 3: Evaluate tests

- Compute the failure intensity
- Graph it over time

Run the tests, logging what you do, and produce those graphs we saw at the beginning.

Measure YOUR system!

- Calculate your FIO
- Measure your failure intensity
- Monitor FI/FIO, release at ½
- Read more
  - Musa, John D. *Software Reliability Engineering: More reliable Software Faster and Cheaper*. 2nd Ed. Author House, Bloomington. 2004.

Take this and apply it to your system. Avoid costly releases and production fires. Find your natural units and calculate your failure intensity objective, then measure your system to see if it's ready!