The Logic of Verification

Michael Bolton
http://www.developsense.com
michael@developsense.com
Twitter: @michaelbolton

James Bach
http://www.satisfice.com
james@satisfice.com
Twitter: @jamesmarcusbach

The Big Ideas

- There is a logical basis to verification. The logic of verification is often misunderstood or ignored.
- Verification is a kind of tool. As with any useful and powerful tool, we must understand its capabilities and its limits to use it effectively.
- *Excellent* verification is *part* of a testing process.
- Testing includes not only questioning of the product, but also questioning of the ways in which we check it and test it.

Call this "Checking", not Testing

operating a product algorithmically to check specific facts about it…

Think "spelling checker"

means

Observe → **Evaluate** → **Report**

Interact with the product in specific, *algorithmic* ways to collect specific observations.

Apply *algorithmic* decision rules to those observations.

Report the outputs of the evaluations *algorithmically*.

A check can be performed…

by a machine
that *can't* think
(but that is quick and precise)

by a human
who has been told *not* to think
(and who is slow and variable)

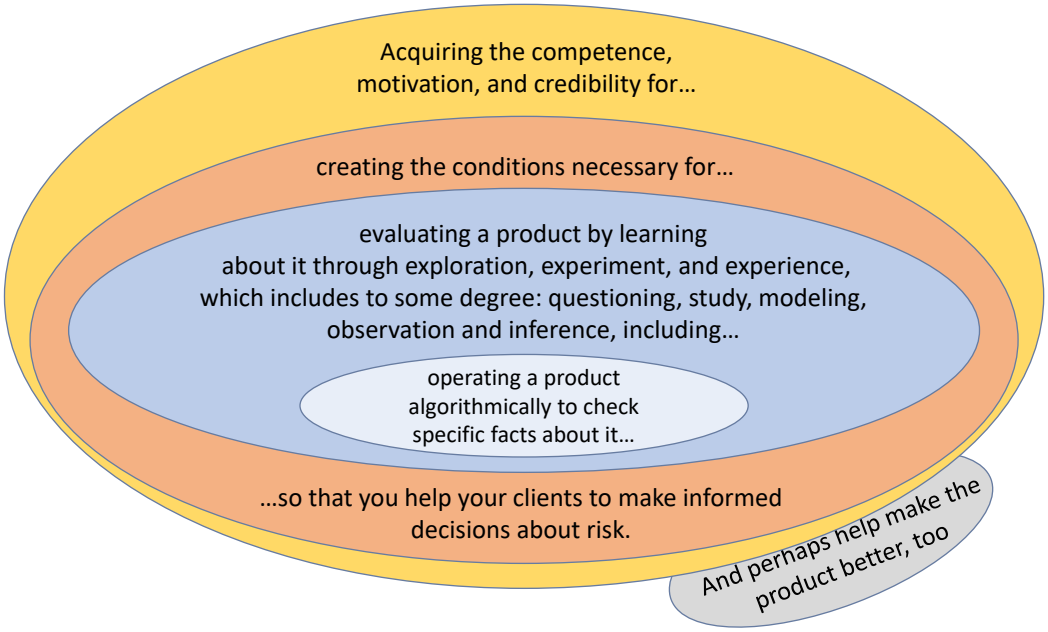
Notice that "quick" and "slow" refer only to the speed of observable behaviours and algorithmic evaluations.
The machine is *infinitely* slow at recognizing unanticipated trouble.

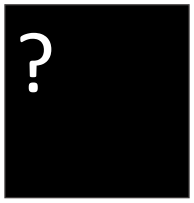
Testing Is *More Than Checking*

- *Checking* is okay, but it is mostly focused on demonstration, confirming what we know or hope to be true.
- To escape problems with verification and to find bugs that matter to people, we must do more than checking; we must *test*.
- And that makes sense, because checking is always embedded in testing!

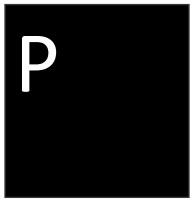
See http://www.developsense.com/2009/08/testing-vs-checking.html

Testing is…

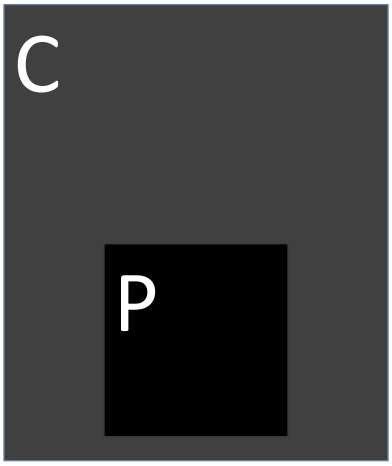




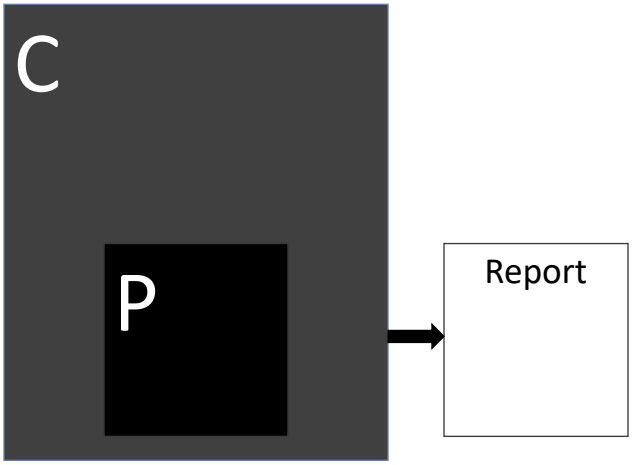
We have a product of uncertain quality.



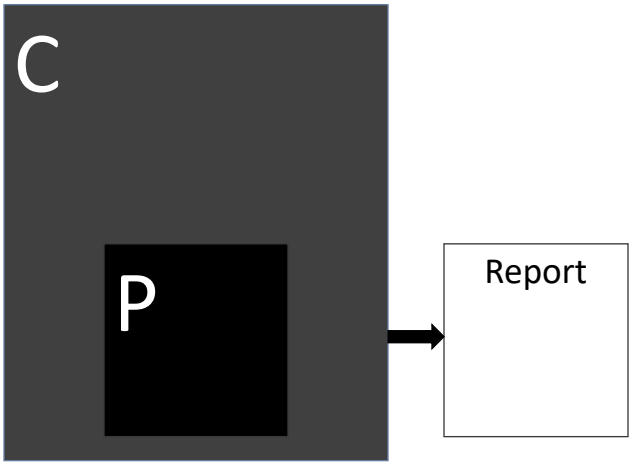
Let’s call it Product P.



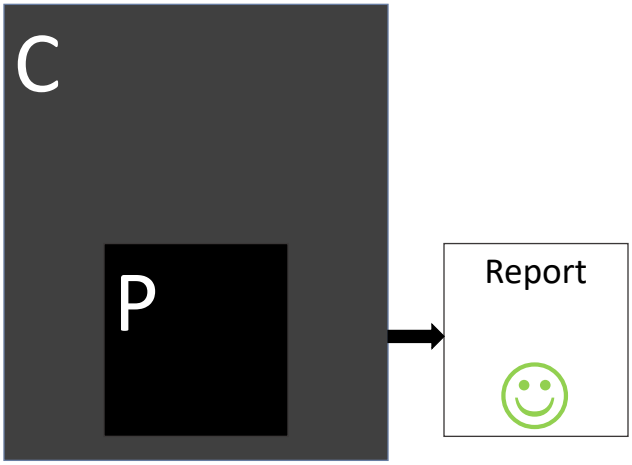
One way to evaluate P is to put it inside another system. We'll call that C (for "Checks").



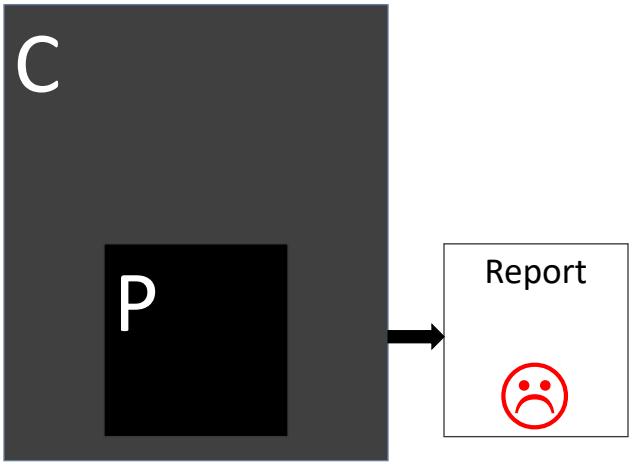
We design C to provide input to P; to operate it; to observe it; to compare the output to a specified result, and to report on the outcome of the comparison.



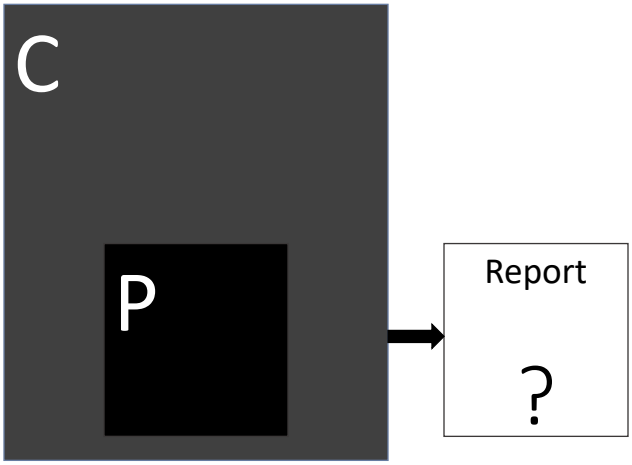
After this process, some people might be tempted to think this way:



"If C reports no problems,
we have verified that P is a good product."



"If C reports problems,
we have verified that P is a bad product."



But what is *really* going on here?

🕮 Verify (n.)

- To ascertain, confirm, check, or test the truth or accuracy of
- To assert or prove to be true
- To testify to the truth of, support (a statement, law)
- To check (items of data input) for accuracy eg by having the same data keyed twice, by two separate operators and then checked by computer for discrepancies (computing)

Weird!

—*Chambers Dictionary*

Verification (in the RST namespace)

🕮 Verification (n.)

1. The process of establishing the truth of a proposition
   (this is universal, rather than specific to software)
2. In regulated software development, the process of comparing a product to its immediate specification

Verification is distinct from "validation". We say this:

🕮 Validation (n.)

the process of assessing a product against how well it fulfills its ultimate purposes

What IS Verification?

- Something exists.
- Some of what exists can be known.
- Some of *that* can be described in words.
- Some of *that* can be expressed as propositions (statements with truth values) which are either true or false.

Verification is the process of establishing the truth or falsehood of a proposition.

Verification isn't a feeling.

Verification is reasoning via a logical process, within a logical system.

- X + Y = 10 has a **truth value** and **can be verified** as true or false **if** the values of X and Y are known, **and if** they are numbers, **and if** the conventions of arithmetic apply.
- X + Y = 10 may have a **truth value** that **cannot be verified** **if** the conventions of arithmetic apply, **and if** X and Y are numbers, **but** the value of X or of Y is not known.
- X + ☯ = 10 **does not have a verifiable truth value** **if** the conventions of mathematics apply.

(We chose the ☯ symbol because it looks nice, and yin/yang starts with Y, but the symbol doesn't stand for anything in particular here.)

DID work is not DOES work; CAN work is not WILL work.

In a system with a non-trivial state space, X + Y = 10 **may be true** ten times in a row, yet **may be false on the next iteration**.

- If you find X + Y = 10 to be true even once, then you have verified that it **CAN** be true.
- From that, you *could* make an inference that it will **PROBABLY** be true next time.
- But unless you check **EVERY POSSIBLE** state of the system, *including possible states that you don't even know are possible*, you cannot verify that X + Y = 10 will **CERTAINLY** be true.

Key questions: What assumptions are supporting your inferences? What could change that would cause your inferences to change?

Verifying Statements About The Future

- Obtain a time machine, and go to a set point in the future.
- Ask *all* customers and stakeholders "Were you satisfied with it?"
- Come back and report success! Huzzah!
- But even then, you can't verify that people would *remain* satisfied *after* you asked them.

Infinite Leap: situated fact ➔ abstract speculation

What I can observe is knowable here and now:

"The product does not currently appear to be in a crashed state."

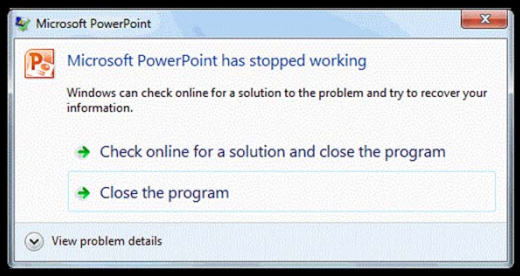
...but the fact that this is true does not mean that the product

- didn't crash without visible manifestations
- won't crash with different data
- won't crash right now if I move the mouse or type the wrong key
- won't crash five minutes from now

But what I care about may be timeless and universal:

"The product shall not crash."

This cannot be verified empirically.



Infinite Leap: situated fact ➔ abstract speculation

What I can observe is knowable here and now:

"I am able to read all the buttons on this screen."

… but the fact that this is true does not mean that it will be true

- for all buttons in the product
- at all times
- on all browsers, in every state
- for every kind of person
- under all lighting conditions

But what I care about may be timeless and universal:

"The product shall be reasonably easy to use."

This cannot be verified empirically.

Infinite Leap: situated fact ➔ abstract speculation

What I can observe is knowable here and now:

"I recognize the login prompt and see nothing wrong."

… but the fact that this is true does not mean that it will be true

- for every situation where the login prompt should be displayed
- that it is compatible with every browser
- that all the client-side JavaScript and all the PHP on the server do all the right things

But what I care about may be timeless and universal:

"The system shall always be in the appropriate state after logging in."

This cannot be verified empirically.

What I have learned from conferences, books, blogs, articles, and testing forums

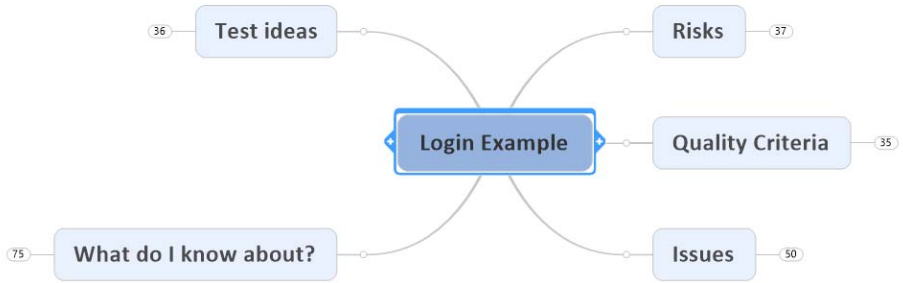
The most thoroughly tested part of any application!

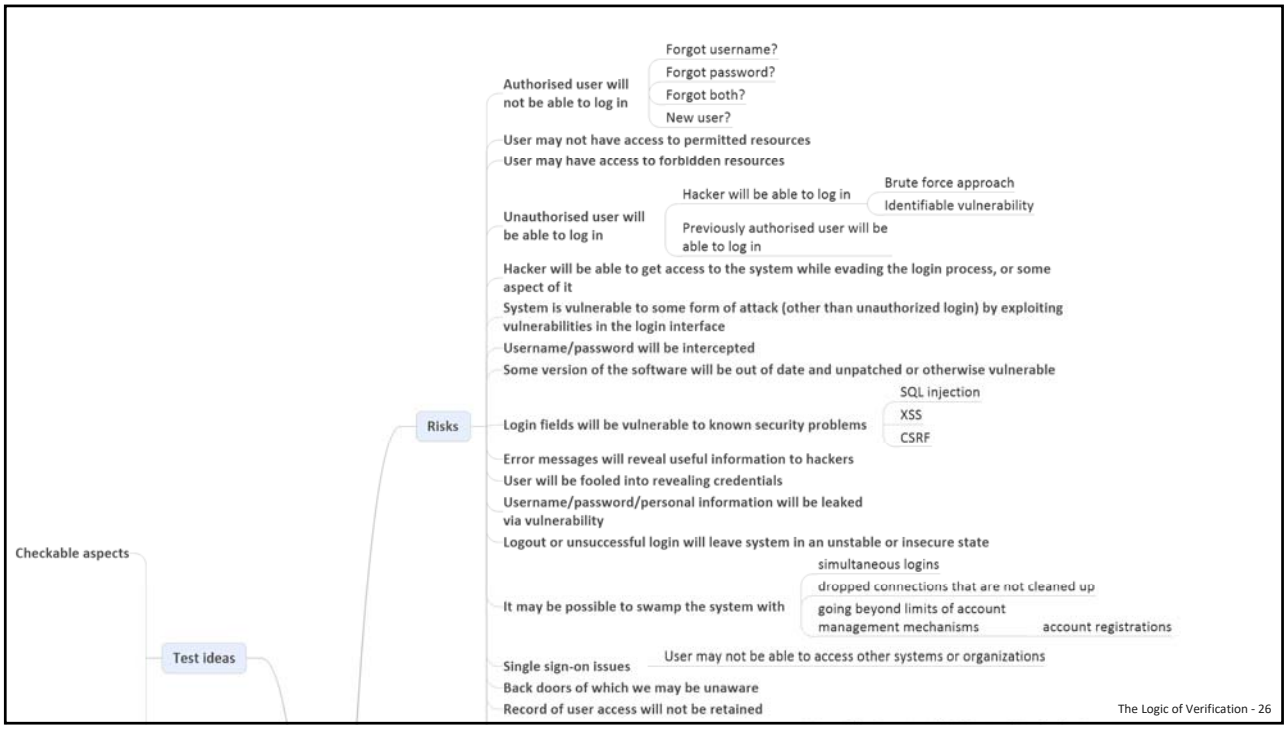
And thanks to GEMPUB, machines can log in hundreds of times a minute.

And people will say “Lo! There be testing!”

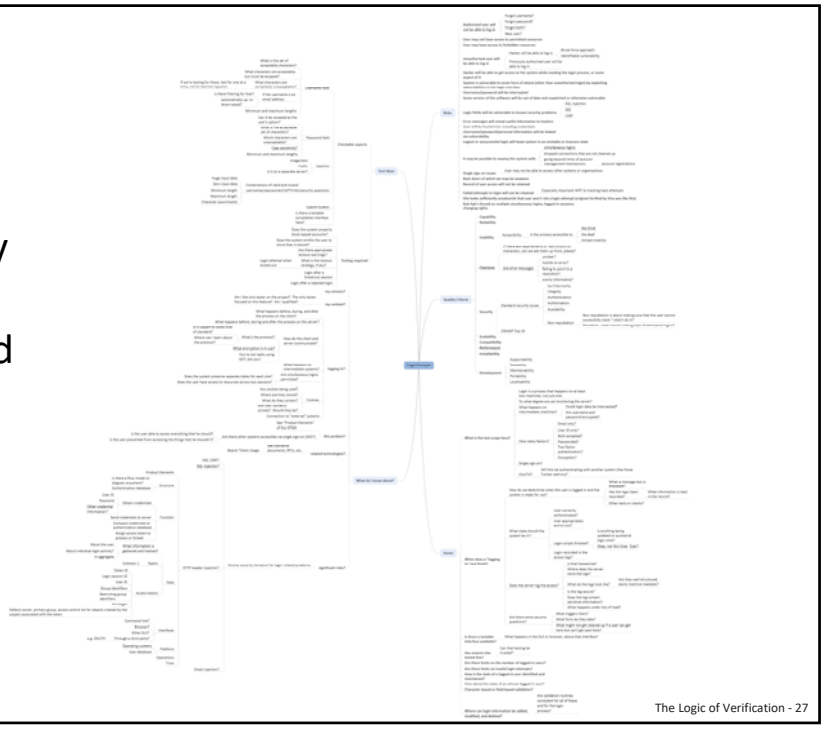
```
describe("Navigate to browser", function(){

    var fnAuthentication = require('../../../src/function/authentication.js');
    var fnLoading = require('../../../src/function/loading.js');
    var dt = require("../../../testData/data.js");
    var objfnAuthentication = new fnAuthentication;
    var objfnLoading = new fnLoading;
    var objdt = new dt;


    it('Load URL',function() {

        objfnLoading.loadUrl();

    });


    using(dt.Datadriven, function(data,description)
    {
        it('Verify user can login to the system',function() {

            objfnAuthentication.login(data.email, data.password);


        });

    });

});
```


Login Stuff We MIGHT Want To Test





Test ideas for login functionality after one hour of brainstorming and research.



Asymmetries: What We Can (and Can't) Verify

Verifiable	Not Verifiable
that there is a problem for some person	that there will be no problem for that person
that we are not aware of a problem for some person	that there is no problem for any person
that the product did something under specific conditions that we have observed	that the product will do the same thing under conditions that we have not yet observed
that the product DID do something	that the product DOES do something
that the product CAN do something	that the product WILL do something
that we were aware of certain conditions we believed to be relevant to the test	that we were aware of all of the conditions relevant to the test
that a product does not meet a requirement	that a product does meet a requirement
that the product *appears* to meet a requirement to some degree	that the product definitely meets a requirement
that the product has not crashed	that the product will not crash
that we have not observed a problem in a feature so far	that there is no problem in a feature
that someone is currently satisfied with the product, based on what they know at the moment	that someone will continue to be satisfied when new knowledge is revealed
facts that might influence decisions about quality	the product's quality

Verification isn't exactly testing.

To say "This product is very good" **is often like saying**
"This product is very ☯ based on known variables X and Y,
plus all our *assumptions* about *unknown* variables V_1 , V_2 , V_3 …, V_{10000} … etc."

This is **unverifiable**, but it may be **testable**.

To **test** the idea that the product is very good means to examine it and **challenge the assumption** that it's good. So: to test is to developing an understanding of the product, and to look for problems.

Problems?

Management wants an answer to these questions:

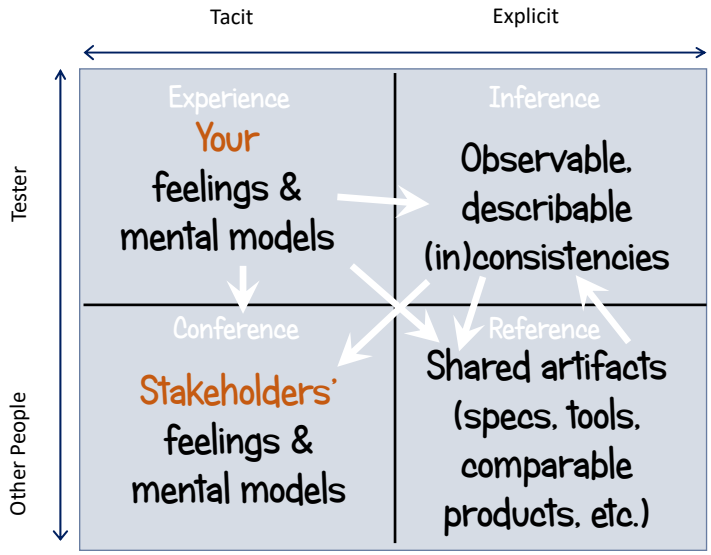
Are there problems that threaten the value of the product to people who matter?

Are there problems that threaten the on-time, successful completion of our work?

Heuristics and Oracles

- A **heuristic** is a way of solving a problem that can work and that might fail.
- An **oracle** is a heuristic for solving the problem "how do I recognize a bug when I encounter one?"
- A **trigger heuristic** is a means of becoming aware that a situation requires your attention.
- A **radiator heuristic** is a means of conveying or representing information that you need to solve a problem.
- A **decider heuristic** is a means of deciding what to do to solve a problem.
- Thus there are **trigger oracles** and **radiator oracles** and **decider oracles**.

Oracles from the Inside Out



Inconsistency ("this disagrees with that")

Heuristics for recognizing problems

- **Acceptability:** The product *is inconsistent* with how good it can reasonably be (not just good; good enough).
- **Familiarity:** The system ***is consistent*** with the pattern of any familiar problem.
- **Explainability:** The system *is inconsistent* with our ability to describe it clearly.
- **World:** The system is *inconsistent* with things or patterns that we recognize in the world.
- **History:** The present version of the system is *inconsistent* with past versions of it.
- **Image:** The system is *inconsistent* with an image that the organization wants to project.
- **Comparable Products:** The system is *inconsistent* with aspects of comparable systems, algorithms, etc.
- **Claims:** The system is *inconsistent* with what important people say it's supposed to be.
- **Users' Desires:** The system is *inconsistent* with what users want.
- **Product:** Each element of the system is *inconsistent* with comparable elements in the same system.
- **Purpose:** The system is *inconsistent* with its purposes, both explicit and implicit.
- **Standards:** The system is *inconsistent* with applicable laws, or relevant implicit or explicit standards.

Inconsistency heuristics rely on the quality of your models of the product and its context.

Verifications Can Be Good Triggers But Are Poor Decider

- A failing check definitely tells you that y
  investigate. That's a **trigger heuristic**
- Failing checks alm never **decide** tha
  first question did it fail? Could it b

a displays are **radia**
st be absorbed, inte

- **Triggers** combined with **radiators** make for especially powerful oracles.

A **decider heuristic** is a means of deciding what

A **radiator heuristic** is a means of conveying or representing information that you need to solve a problem.

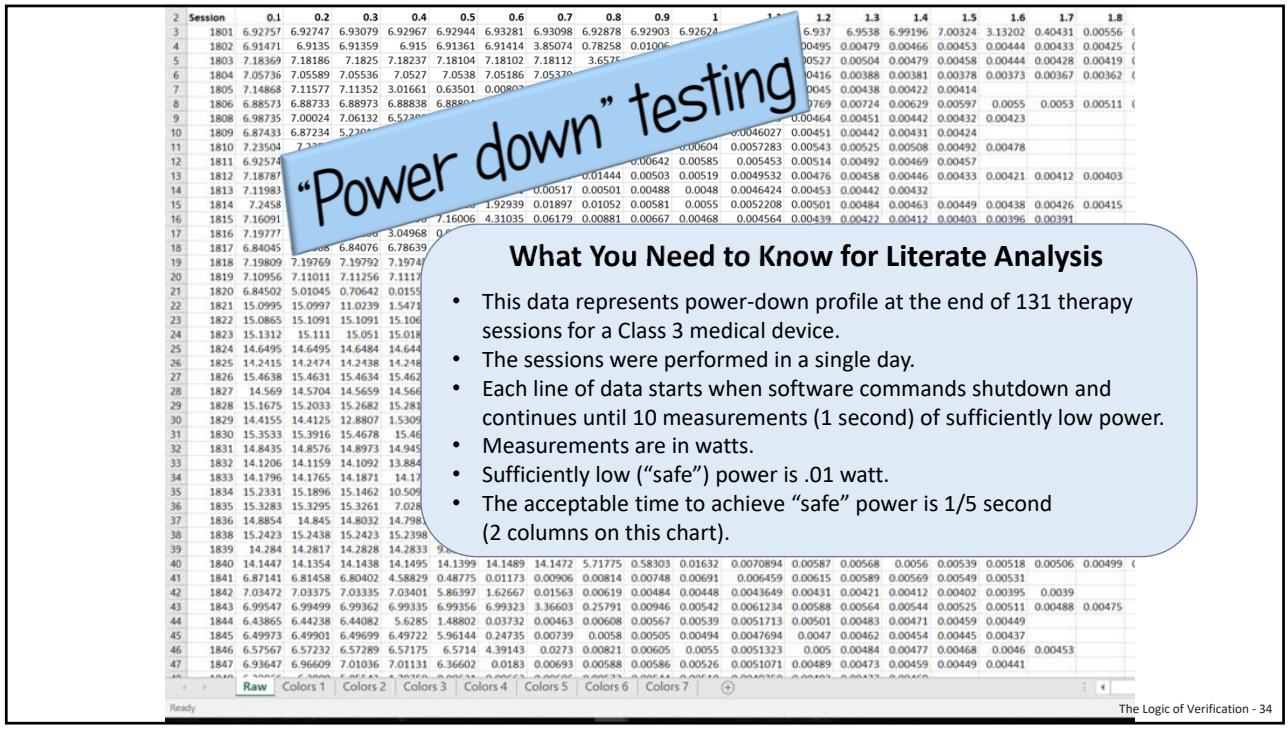
A **trigger heuristic** is a means of becoming aware that a situation requires your attention.

A **heuristic** is a way of solving a problem that can work and that might fail.

An oracle is a **heuristic** by which we recognize a problem —a bug — when we encounter one in testing.

Verifications Can Be Good Triggers But Are Poor Deciders or Radiators

- A failing check definitely tells you that you have work to do. You must investigate. That's a **trigger**.
- Failing checks never **decide** that the software IS bad, because our first question is "Why did it fail? Could it be broken?" ***HUMANS, not checks, decide***.
- Log files, screens, and data displays are **radiators**. They are not subject to "pass/fail" but rather must be absorbed, interpreted, pondered, in loops.
- **Triggers** combined with **radiators** make for especially powerful oracles.



Session	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
1801	6.92757	6.92747	6.93079	6.92967	6.92944	6.93281	6.93098	6.92878	6.92903	6.92624	6.937	6.9538	6.99196	7.00324	3.13202	0.40431	0.00556
1802	6.91471	6.9135	6.91359	6.915	6.91361	6.91414	3.85074	0.78258	0.01006		0.00495	0.00479	0.00466	0.00453	0.00444	0.00433	0.00425
1803	7.18369	7.18186	7.1825	7.18237	7.18104	7.18102	7.18112	3.6575			0.00527	0.00504	0.00479	0.00458	0.00444	0.00428	0.00419
1804	7.05736	7.05589	7.05536	7.0527	7.0538	7.05186	7.05370				0.00416	0.00388	0.00381	0.00378	0.00373	0.00367	0.00362
1805	7.14868	7.11577	7.11352	3.01661	0.63501	0.00803					0.00045	0.00438	0.00422	0.00414			
1806	6.88573	6.88733	6.88973	6.88838	6.88830						0.00769	0.00724	0.00629	0.00597	0.0055	0.0053	0.00511
1808	6.98735	7.00024	7.06132	6.52335							0.00464	0.00451	0.00442	0.00432	0.00423		
1809	6.87433	6.87234	5.2201							0.0046027	0.00451	0.00442	0.00431	0.00424			
1810	7.23504	7.23							0.00604	0.0057283	0.00543	0.00525	0.00508	0.00492	0.00478		
1811	6.92574							0.00642	0.00585	0.005453	0.00514	0.00492	0.00469	0.00457			
1812	7.18787						0.01444	0.00503	0.00519	0.0049532	0.00476	0.00458	0.00446	0.00433	0.00421	0.00412	0.00403
1813	7.11983					0.00517	0.00501	0.00488	0.0048	0.0046424	0.00453	0.00442	0.00432				
1814	7.2458					1.92939	0.01897	0.01052	0.00581	0.0055	0.0052208	0.00501	0.00484	0.00463	0.00449	0.00438	0.00426	0.00415
1815	7.16091				7.16006	4.31035	0.06179	0.00881	0.00667	0.00468	0.004564	0.00439	0.00422	0.00412	0.00403	0.00396	0.00391
1816	7.19777			3.04968													
1817	6.84045		6.84076	6.78639													
1818	7.19809	7.19769	7.19792	7.1974													
1819	7.10956	7.11011	7.11256	7.1117													
1820	6.84502	5.01045	0.70642	0.0155													
1821	15.0995	15.0997	11.0239	1.5471													
1822	15.0865	15.1091	15.1091	15.106													
1823	15.1312	15.111	15.051	15.018													
1824	14.6495	14.6495	14.6484	14.644													
1825	14.2415	14.2474	14.2438	14.248													
1826	15.4638	15.4631	15.4634	15.462													
1827	14.569	14.5704	14.5659	14.566													
1828	15.1675	15.2033	15.2682	15.281													
1829	14.4155	14.4125	12.8807	1.5309													
1830	15.3533	15.3916	15.4678	15.46													
1831	14.8435	14.8576	14.8973	14.945													
1832	14.1206	14.1159	14.1092	13.884													
1833	14.1796	14.1765	14.1871	14.17													
1834	15.2331	15.1896	15.1462	10.509													
1835	15.3283	15.3295	15.3261	7.028													
1836	14.8854	14.845	14.8032	14.798													
1838	15.2423	15.2438	15.2423	15.2398													
1839	14.284	14.2817	14.2828	14.2833													
1840	14.1447	14.1354	14.1438	14.1495	14.1399	14.1489	14.1472	5.71775	0.58303	0.01632	0.0070894	0.00587	0.00568	0.0056	0.00539	0.00518	0.00506	0.00499
1841	6.87141	6.81458	6.80402	4.58829	0.48775	0.01173	0.00906	0.00814	0.00748	0.00691	0.006459	0.00615	0.00589	0.00569	0.00549	0.00531	
1842	7.03472	7.03375	7.03335	7.03401	5.86397	1.62667	0.01563	0.00619	0.00484	0.00448	0.0043649	0.00431	0.00421	0.00412	0.00402	0.00395	0.0039	
1843	6.99547	6.99499	6.99362	6.99335	6.99356	6.99323	3.36603	0.25791	0.00946	0.00542	0.0061234	0.00588	0.00564	0.00544	0.00525	0.00511	0.00488	0.00475
1844	6.43865	6.44238	6.44082	5.6285	1.48802	0.03732	0.00463	0.00608	0.00567	0.00539	0.0051713	0.00501	0.00483	0.00471	0.00459	0.00449	
1845	6.49973	6.49901	6.49699	6.49722	5.96144	0.24735	0.00739	0.0058	0.00505	0.00494	0.0047694	0.0047	0.00462	0.00454	0.00445	0.00437	
1846	6.57567	6.57232	6.57289	6.57175	6.5714	4.39143	0.0273	0.00821	0.00605	0.0055	0.0051323	0.005	0.00484	0.00477	0.00468	0.0046	0.00453	
1847	6.93647	6.96609	7.01036	7.01131	6.36602	0.0183	0.00693	0.00588	0.00586	0.00526	0.0051071	0.00489	0.00473	0.00459	0.00449	0.00441	

Raw | Colors 1 | Colors 2 | Colors 3 | Colors 4 | Colors 5 | Colors 6 | Colors 7

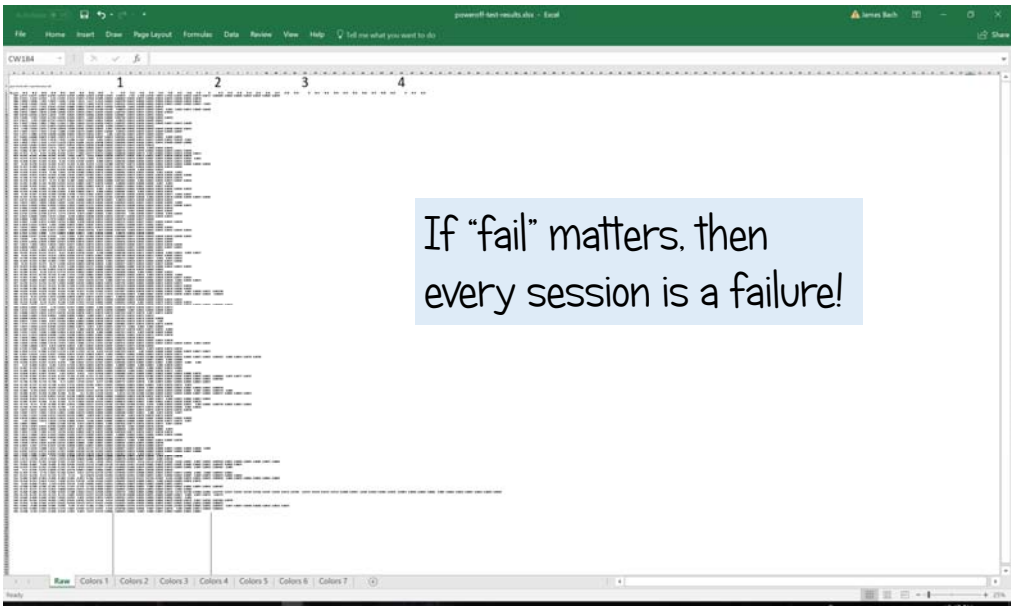
Ready

"Power down" testing

What You Need to Know for Literate Analysis

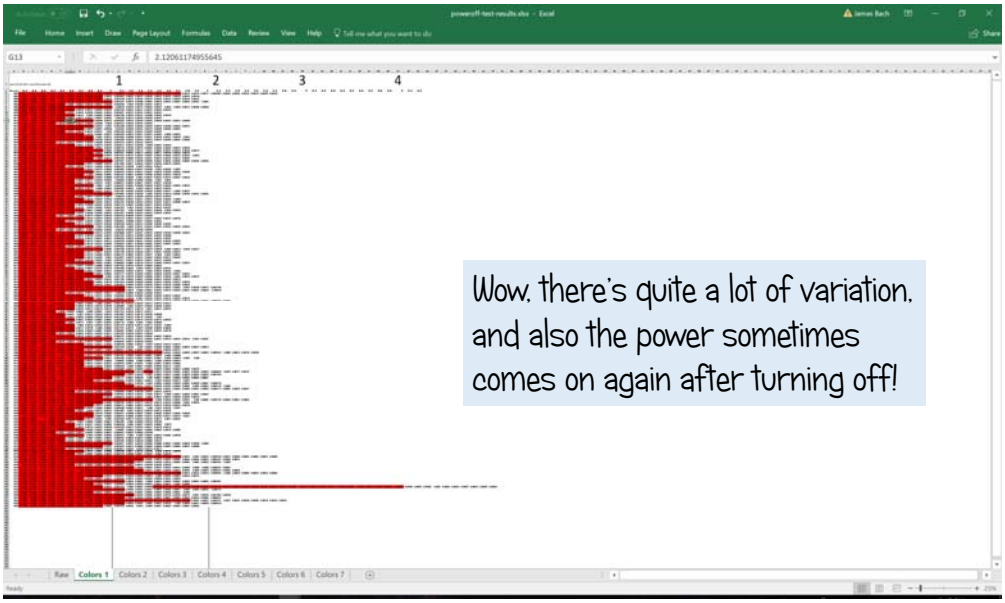
- This data represents power-down profile at the end of 131 therapy sessions for a Class 3 medical device.
- The sessions were performed in a single day.
- Each line of data starts when software commands shutdown and continues until 10 measurements (1 second) of sufficiently low power.
- Measurements are in watts.
- Sufficiently low ("safe") power is .01 watt.
- The acceptable time to achieve "safe" power is 1/5 second (2 columns on this chart).

A "zoom blink" radiator oracle



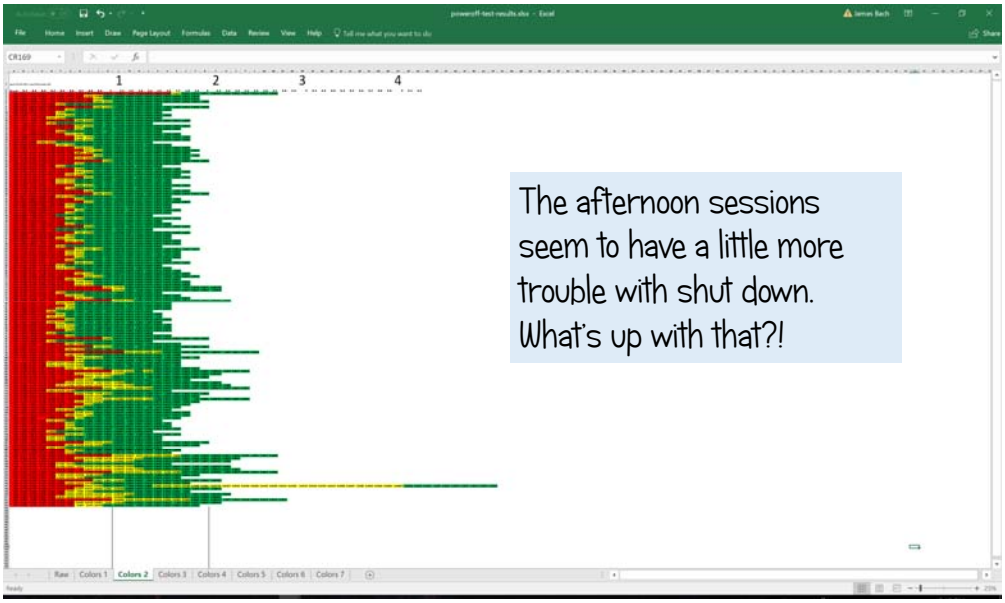
If "fail" matters, then every session is a failure!

>.01w **<=.01w**



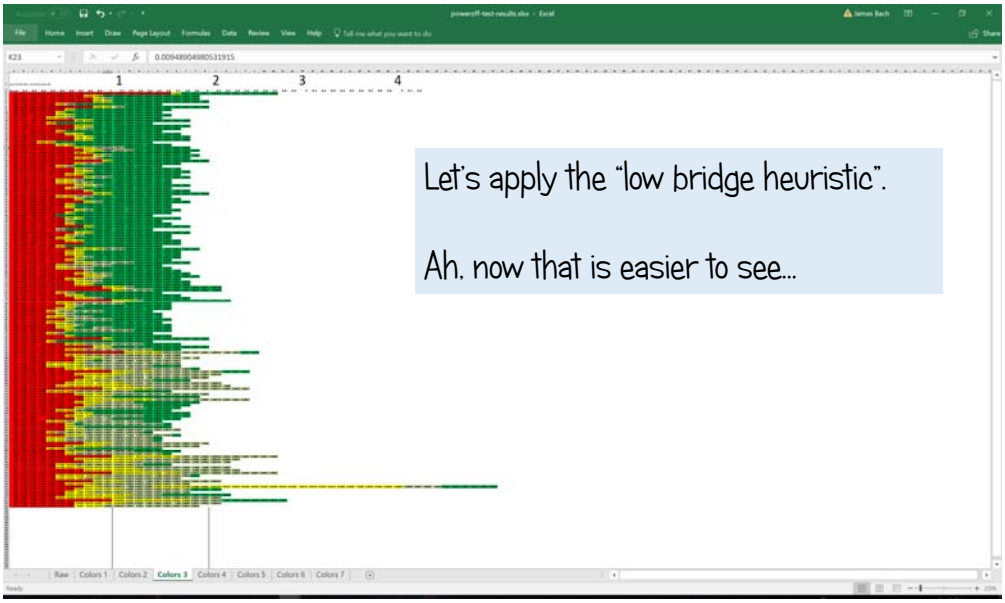
Wow, there's quite a lot of variation, and also the power sometimes comes on again after turning off!

>=1w **<1w & >.01w** **<= .01w**



The afternoon sessions seem to have a little more trouble with shut down. What's up with that?!

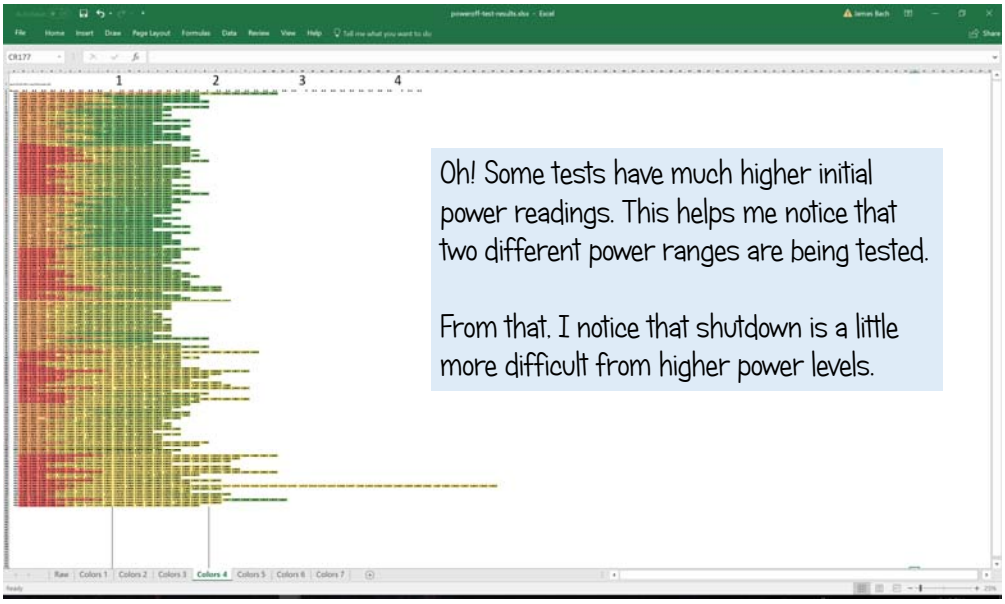
>=1w **<1w & >.1w** **=<.1w & >.01w** **<=.01w**



Let's apply the "low bridge heuristic".

Ah, now that is easier to see...

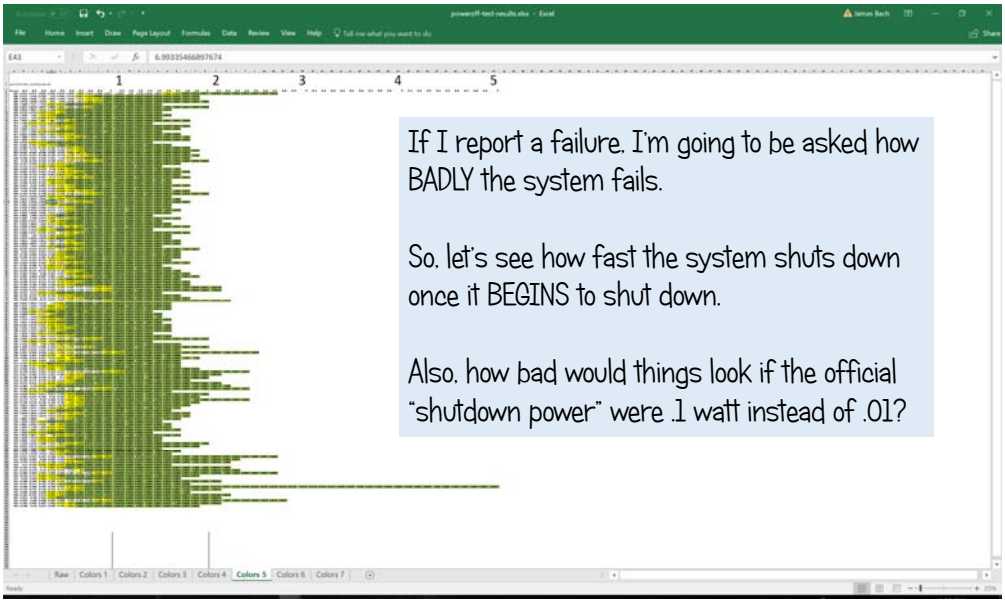
Defocusing: let Excel choose the coloring



Oh! Some tests have much higher initial power readings. This helps me notice that two different power ranges are being tested.

From that, I notice that shutdown is a little more difficult from higher power levels.

initial power <= 90% of initial power >= 0.1w

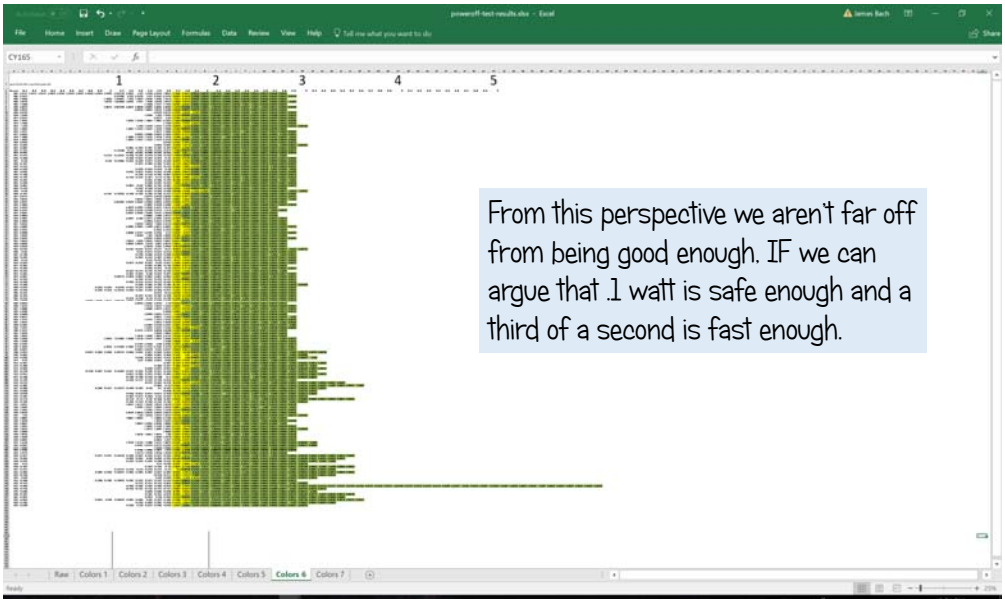


If I report a failure. I'm going to be asked how BADLY the system fails.

So, let's see how fast the system shuts down once it BEGINS to shut down.

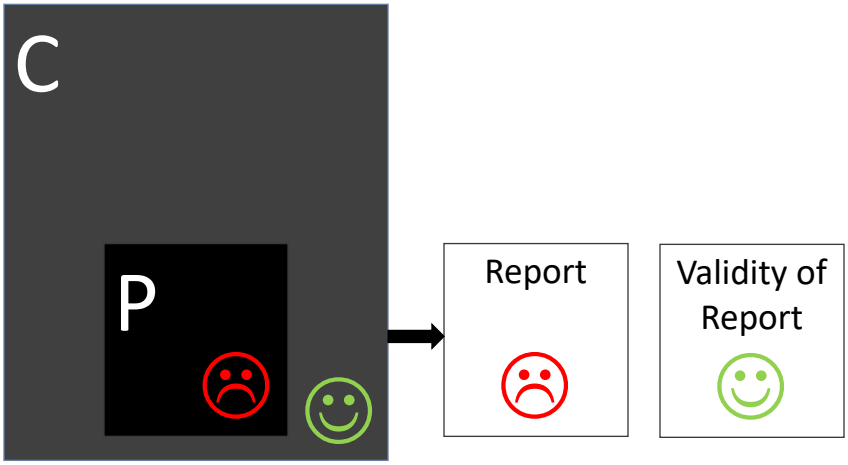
Also, how bad would things look if the official "shutdown power" were .1 watt instead of .01?

Centered on first <= 90% measurement

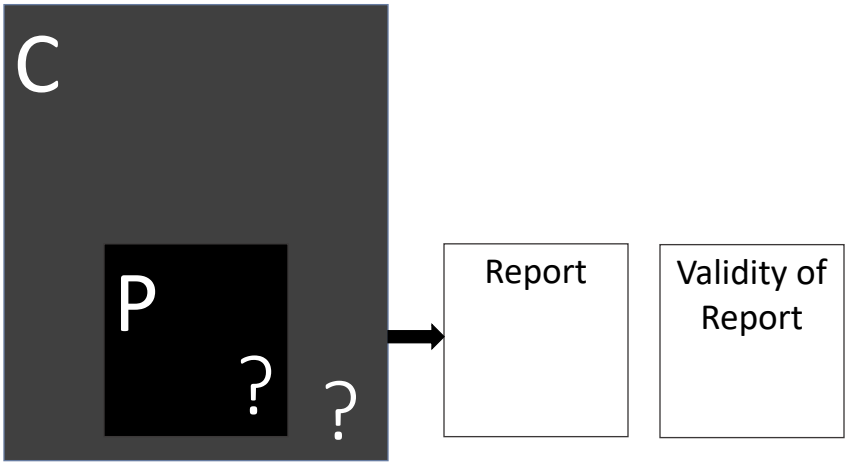


From this perspective we aren't far off from being good enough. IF we can argue that .1 watt is safe enough and a third of a second is fast enough.

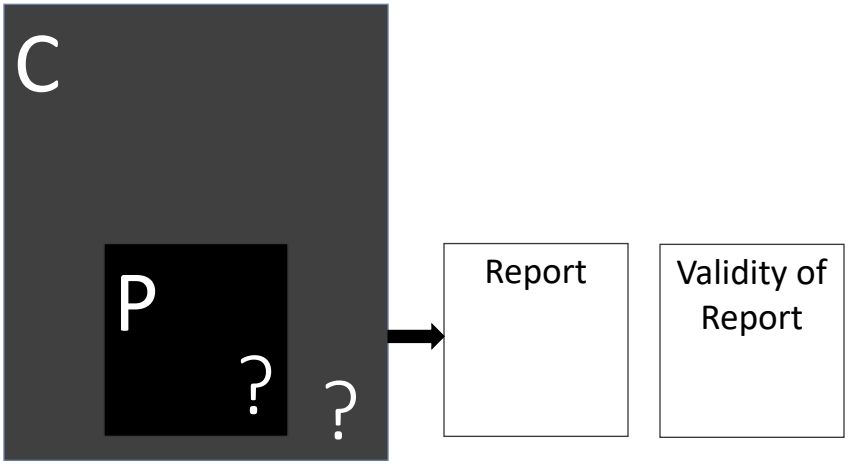
C reports that P is good. If and *only if* C is good AND P is good too, that report is valid



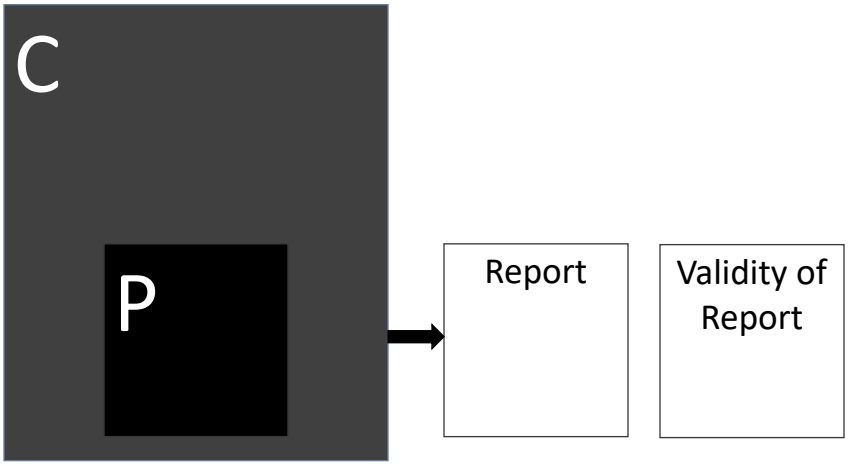
If C is good, it will correctly report that P is bad when P's quality is bad; C's report will be valid.



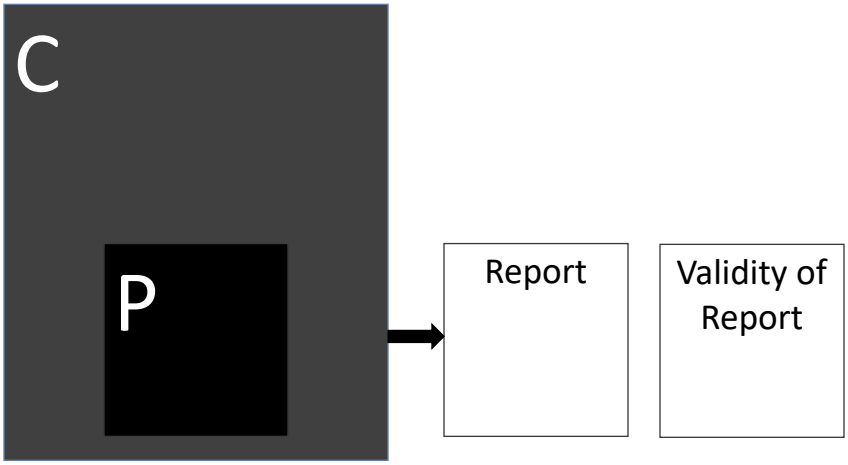
Code is often written quickly, or under pressure, or both—whether it's product code or check code.



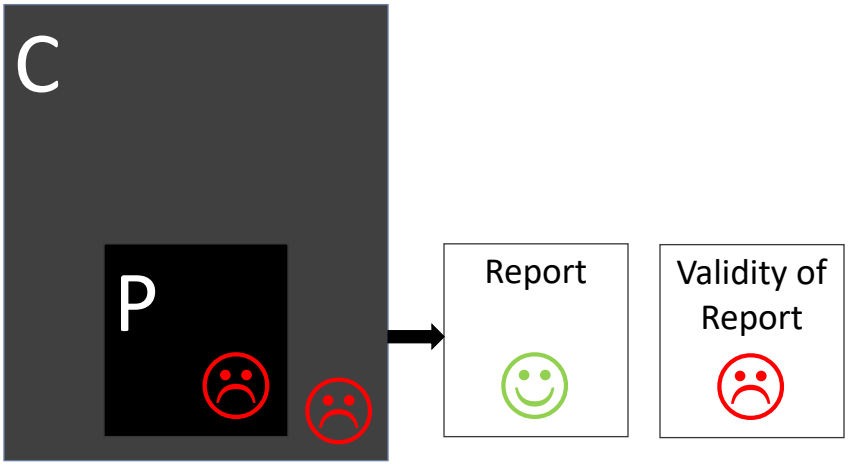
It is tempting to believe that
product code is more important than check code.
Customers don't see check code.



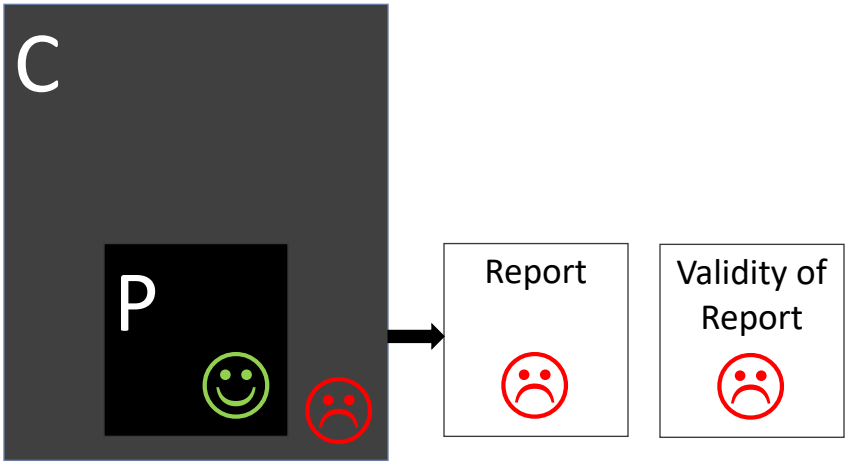
But conclusions we might make about P are risky, because the quality of *both* P *and* C is uncertain.



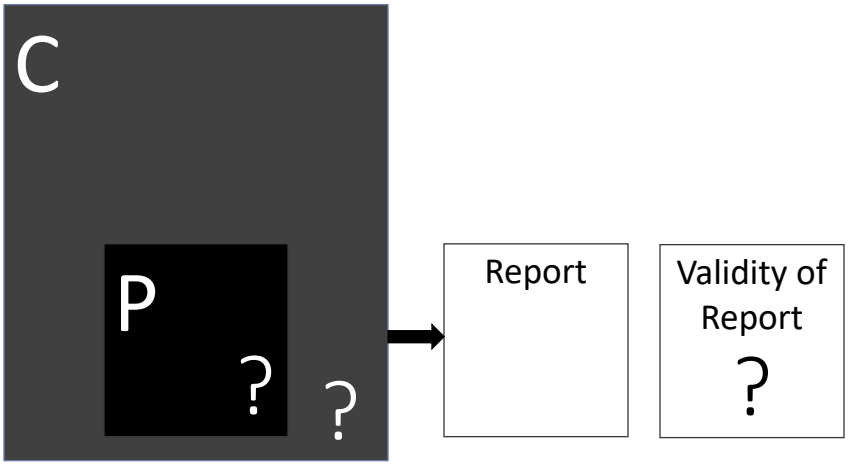
Conclusions we might make about P are even more risky when C is not developed carefully and skillfully.



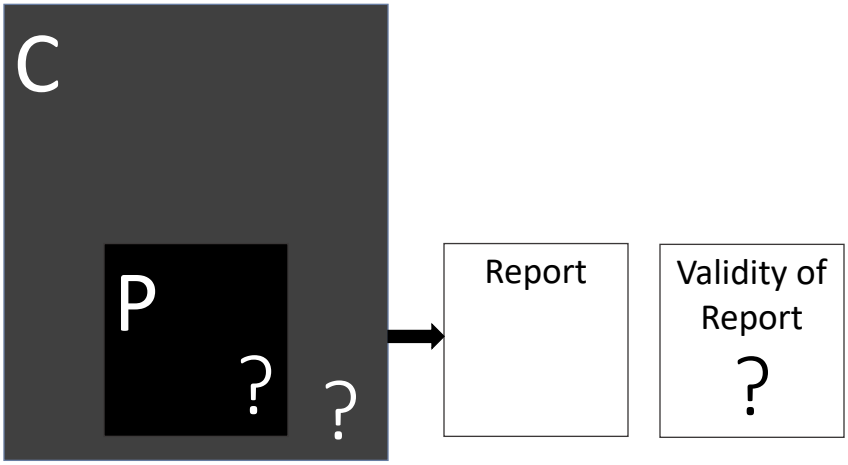
If C is not good, C will incorrectly report that P is good. It may be that P's quality is bad.



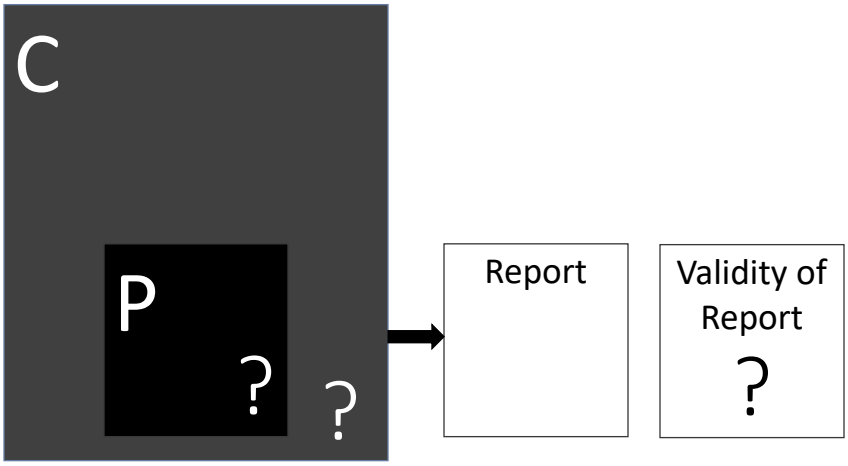
If C is not good, C may incorrectly report that P is bad, even when P's quality is good.



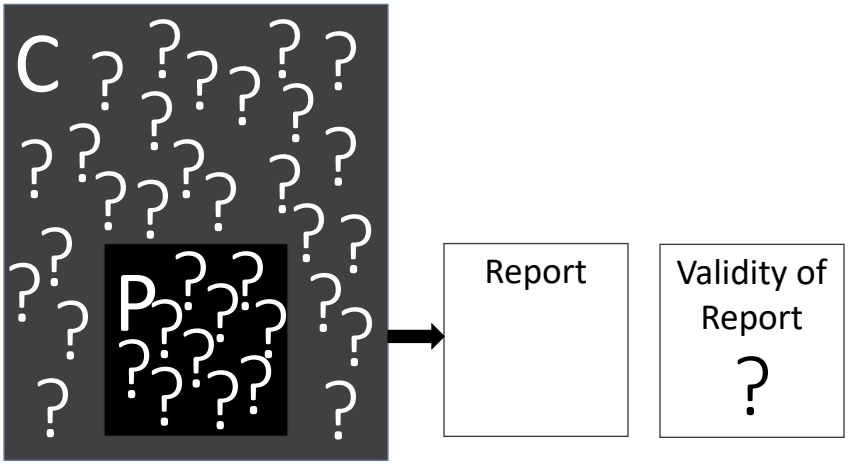
That is: unless we *know* C to be good,
we can't be sure about the validity of the report!



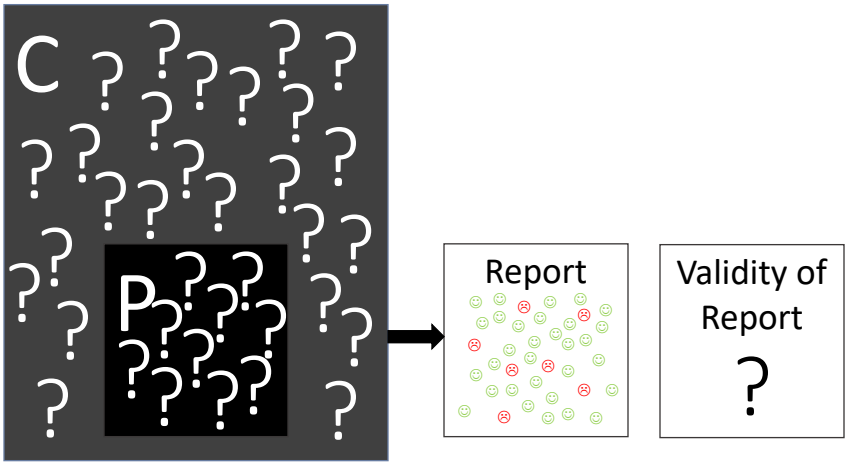
In other words: without investigation and analysis,
we can't be sure of the quality of *either* C or P.



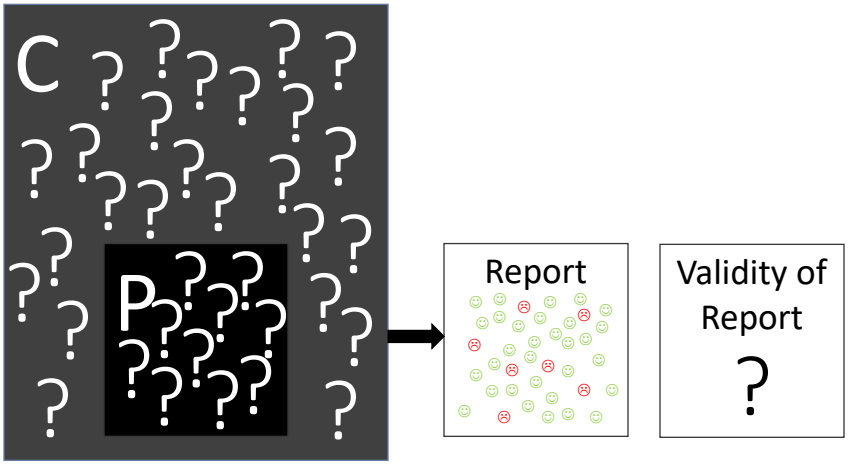
If you think that's bad,
our troubles are only beginning.



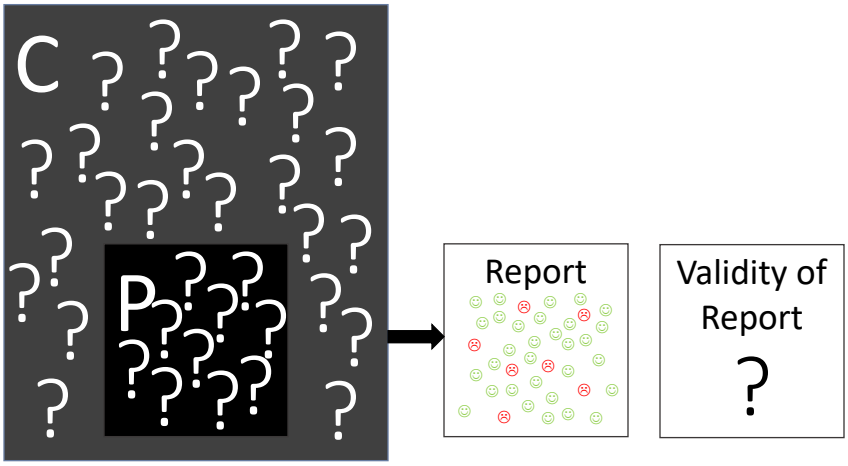
C reports not just a single bit, but on a collection of
hundreds or thousands of individual check results.



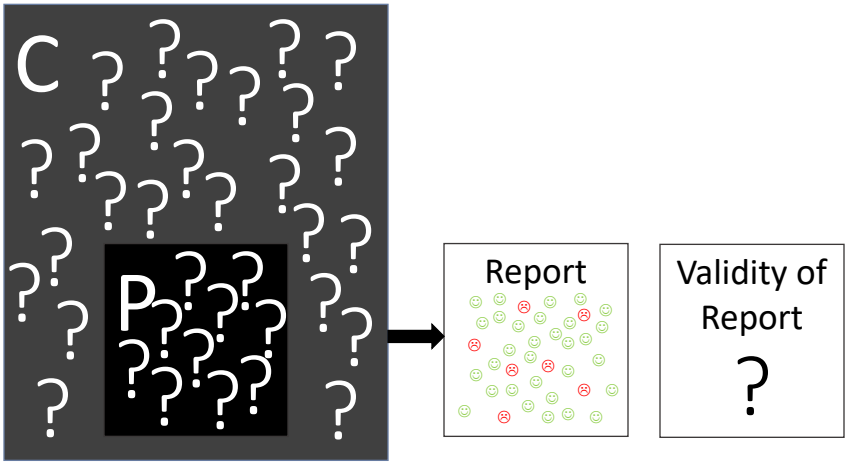
C reports not just a single bit, but on a collection of hundreds or thousands of individual check results.



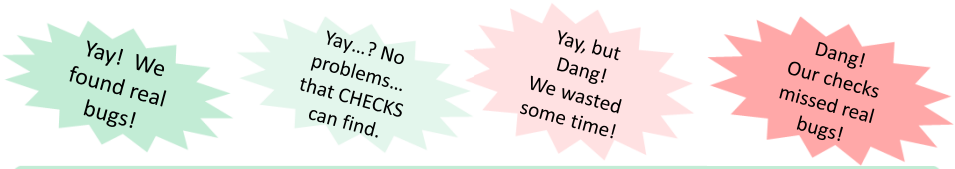
When a set of checks reports a failure, a responsible tester will not immediately report a bug.



Instead, the tester must investigate and ask if this is a real problem in the product, or a problem with C.



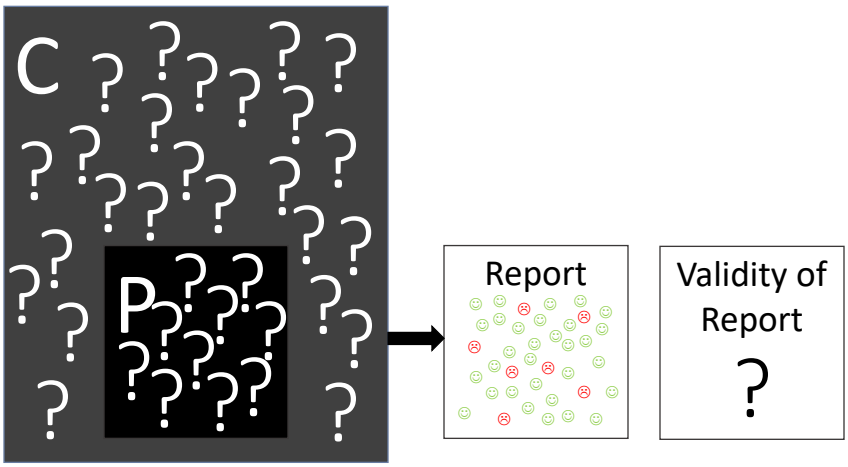
But we've already seen that neither "failing" checks *nor* "passing" ones are always valid.



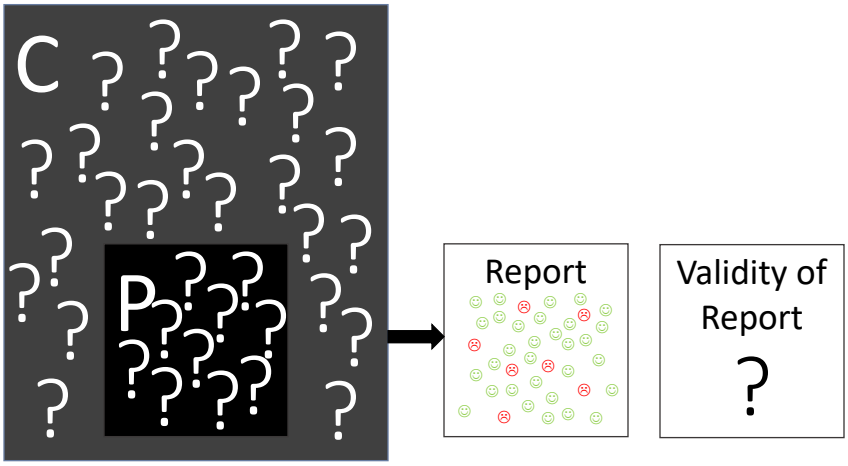
1. A bad product gets categorized as bad because our checks detect problems.
2. A good product gets categorized as not-known-to-be-bad because no problems were found.
3. A good product gets categorized as bad because one or more of our checks is wrong. (Type I error)
4. A bad product gets categorized as not-known-to-be-bad because none of our checks were good enough to detect its particular badness. (Type II Error)

We will probably consider our checks good if…

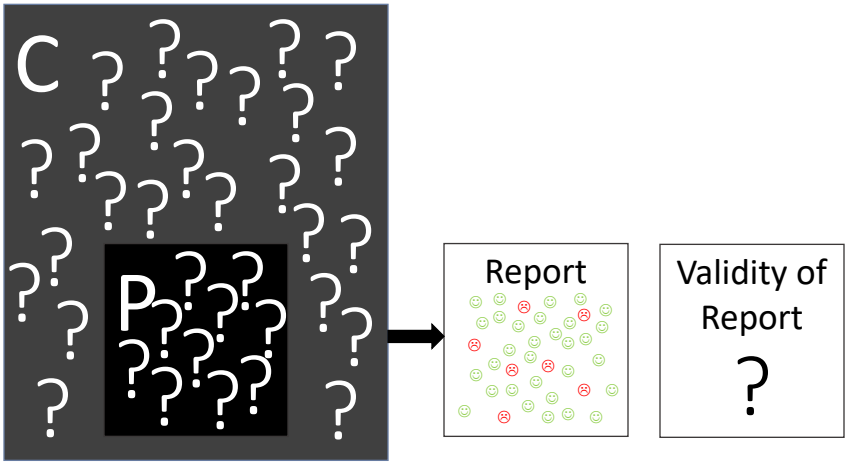
A. We believe 1 and 2 are likely (validity);
B. We believe 3 and 4 are *un*likely (reliability); AND
C. The checks don't cost too much. (utility)



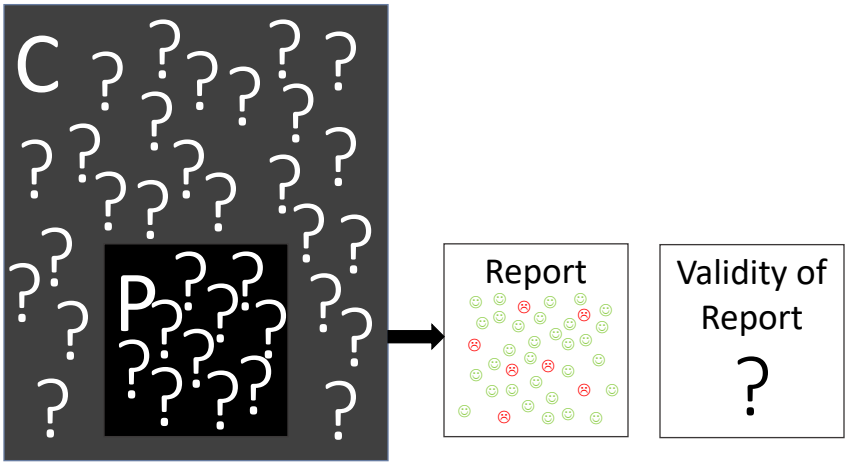
Some people dismiss checks that intermittently report failures as "flaky checks".



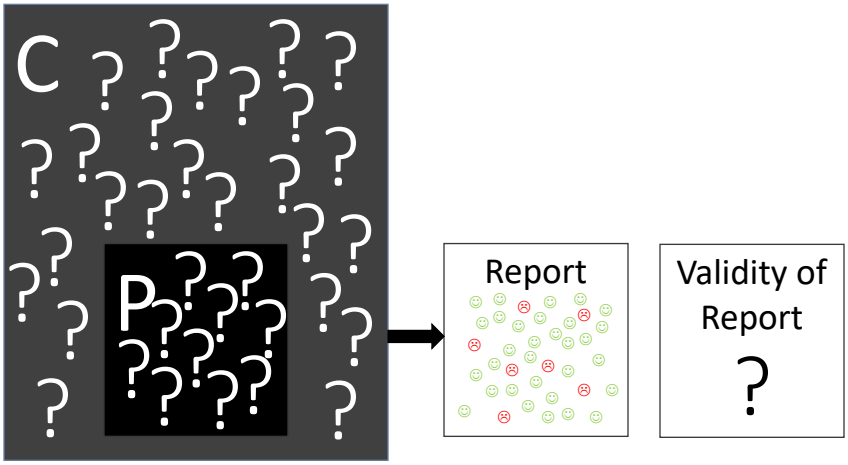
How do we know that “flaky checks” are not problems in the product? Have we tested that idea?



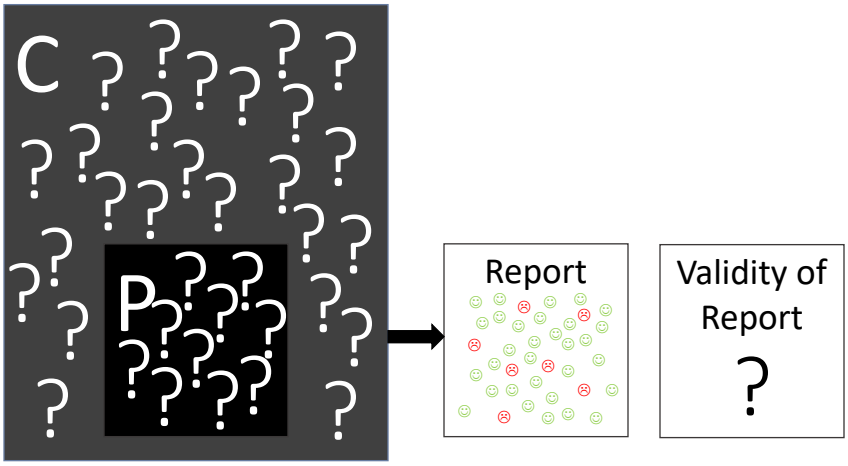
And if checks are consistently “flaky”, why run them at all?



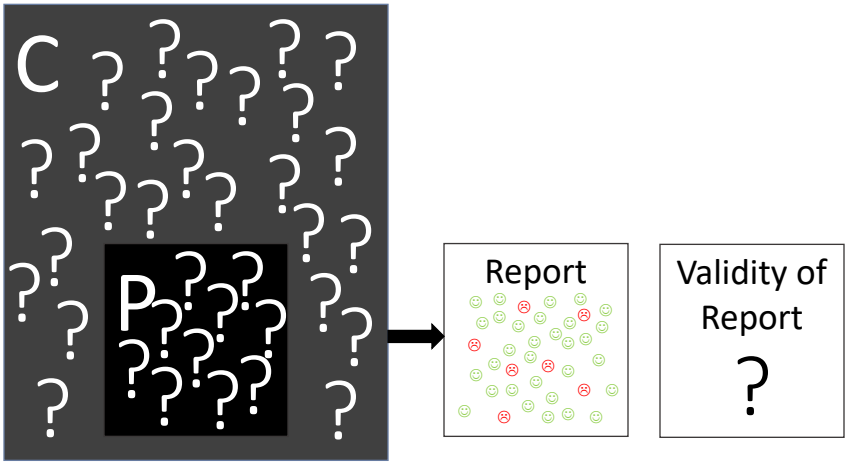
If we think that tests can fail What are we doing to test the idea that reports of “passing” checks are valid?



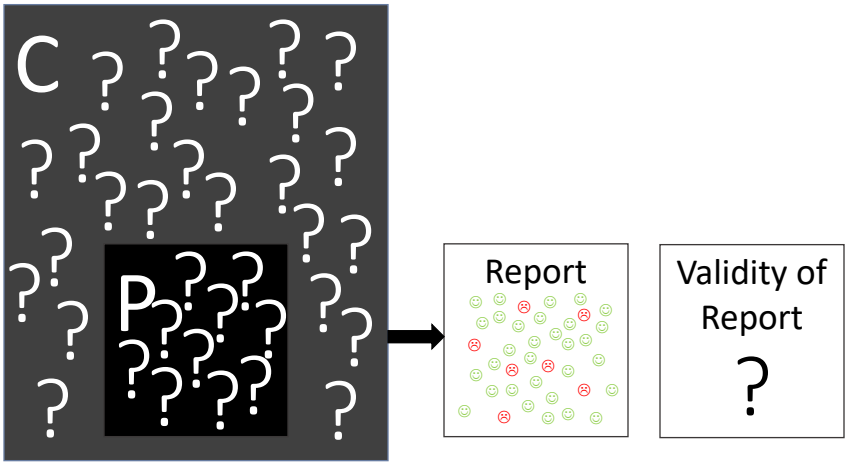
What are we doing to use checks more powerfully—to check more broadly and deeply?



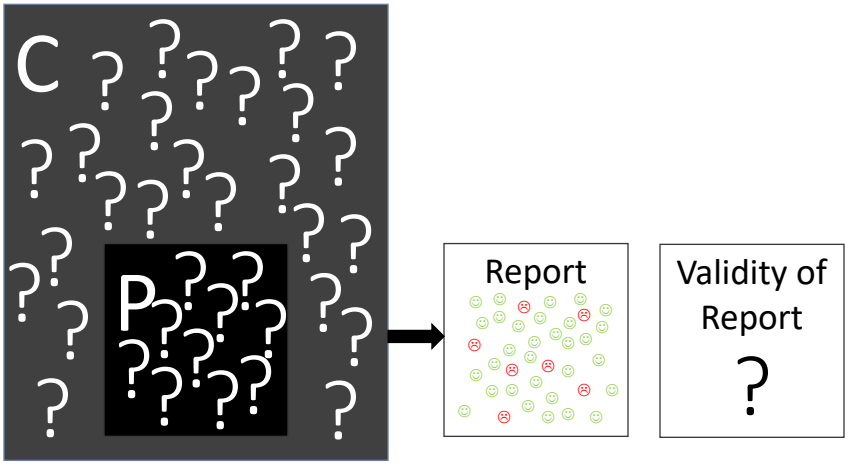
There are many quality criteria that cannot be checked easily: testability, maintainability…



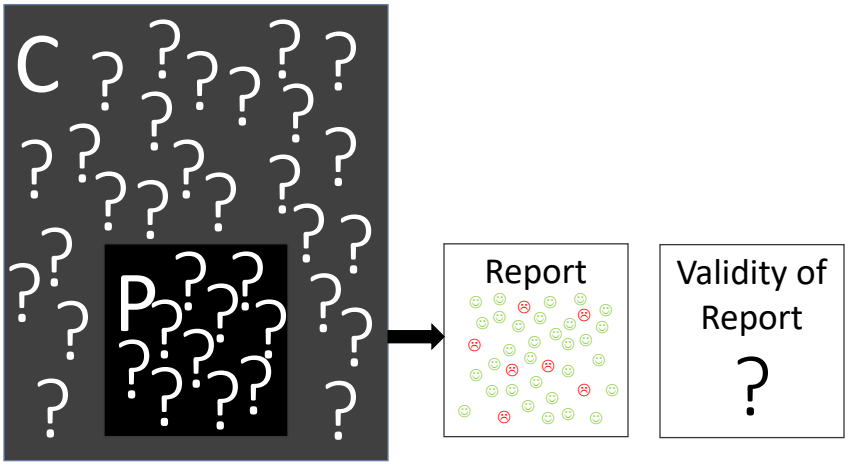
What are we doing to find problems that are not found by automated checking?



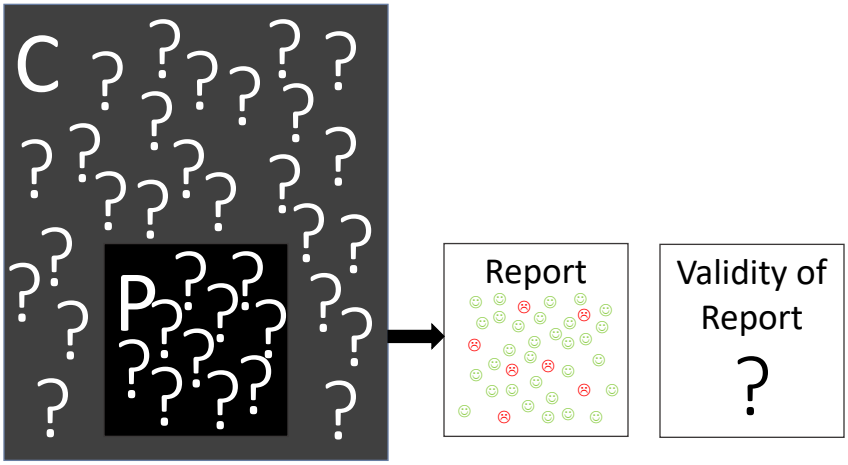
Many people say automated checking allows more time for “exploratory” testing. Is that true?



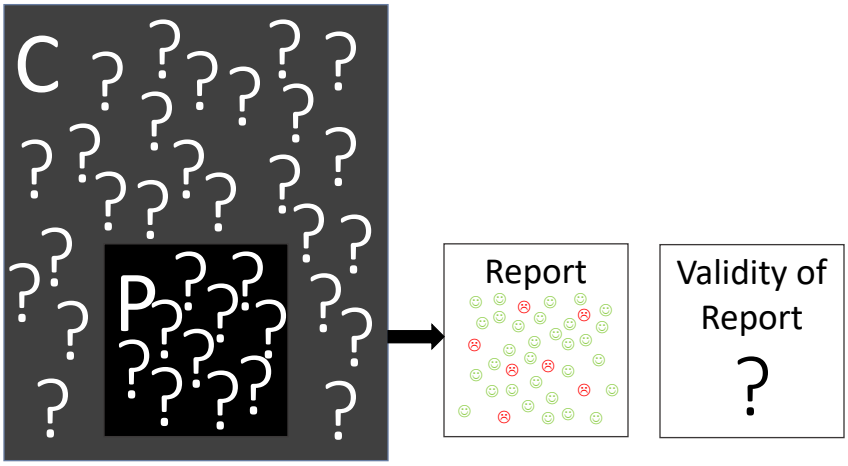
We MIGHT have more time for “exploratory” testing if checks are inexpensive to develop, quick to run, and easy to interpret and analyze.



We MIGHT have more time for “exploratory” testing if we are not investigating too many “failing” checks.



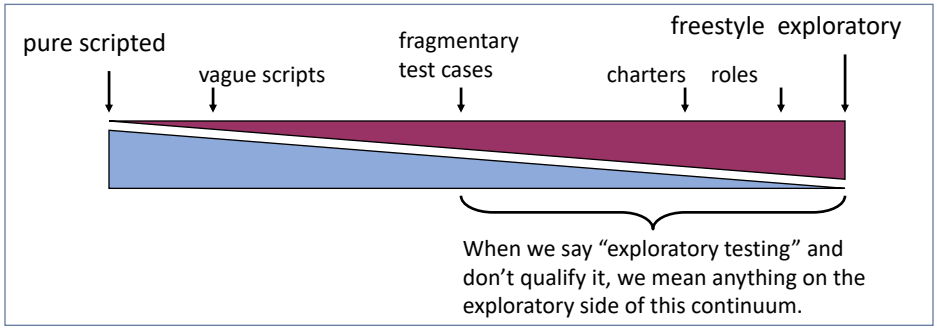
But we might have fewer “failing” checks if we do more “exploratory” testing earlier!



To understand how to do "exploratory" testing before checking, we must learn what testing really is.

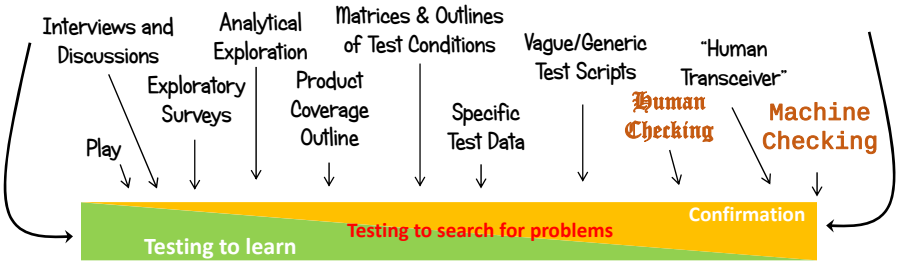
The Scripted/Exploratory Continuum, 2003

- When James Bach was describing testing in 2003, he put scripted testing on the left and exploratory testing on the right.
- Turns out there was a huge bug in this idea that we didn't notice *for years*.
- It looks like scripted testing comes first! But it doesn't!



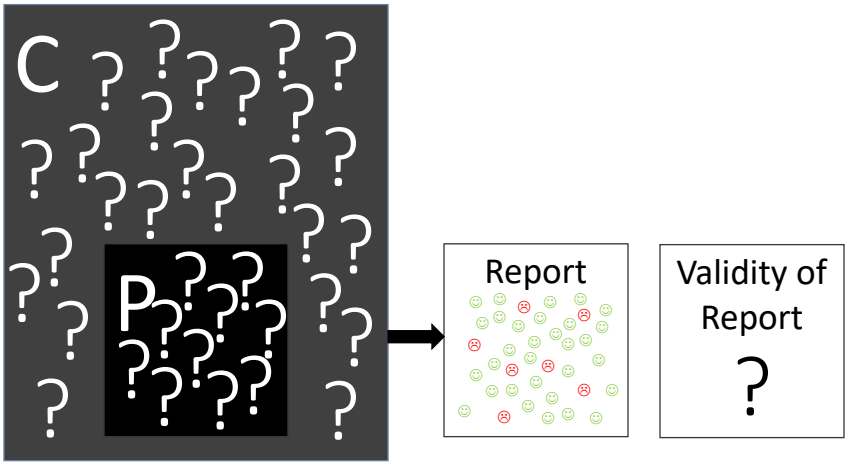
James Bach: The Scripted/Exploratory Continuum from 2003

The *Formality* Continuum, 2014



Loops of testing start with informal, exploratory work. If you want to do excellent formal testing (like automated checking), it must begin with excellent informal work.

Bug fix: formality tends to intensify over time. Showing informality on the left and formality on the right makes more sense.



Now we can talk about what
"doing exploratory testing earlier" means.

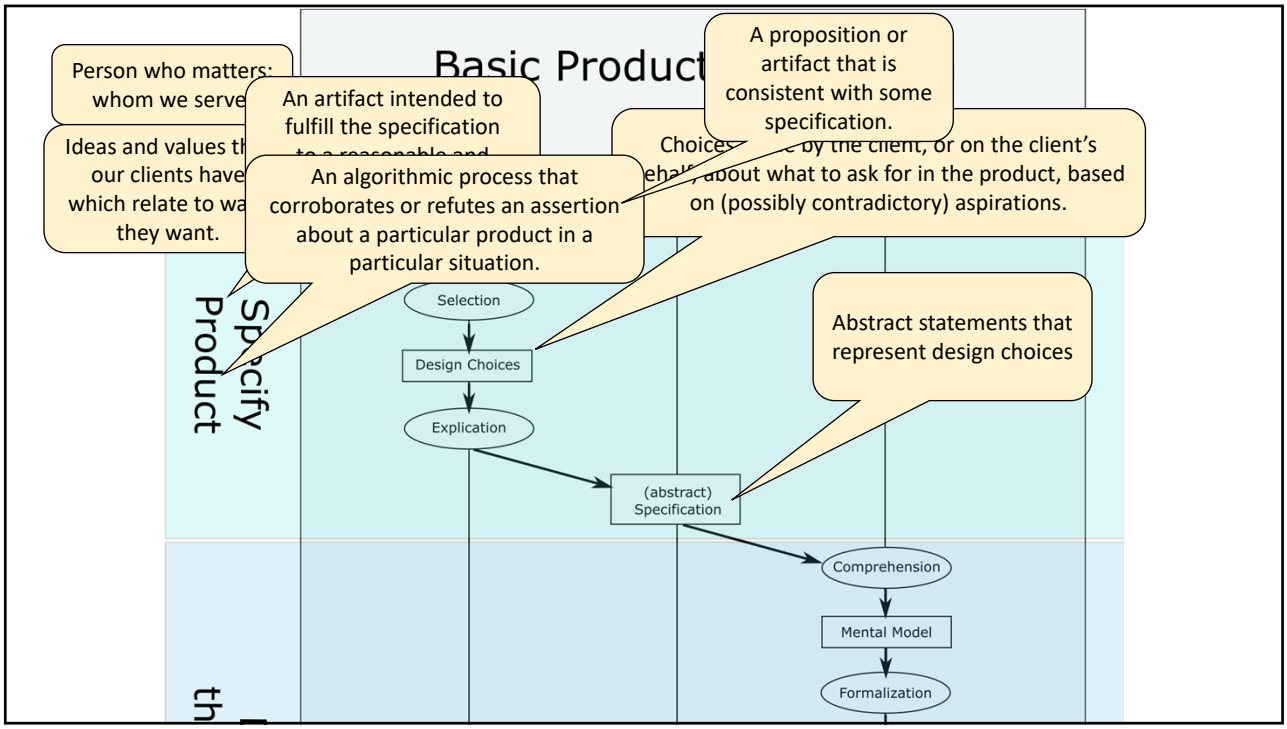
Exploratory testing includes…

In other words…

Exploratory testing is *testing*. Verification is more like *demonstration*.

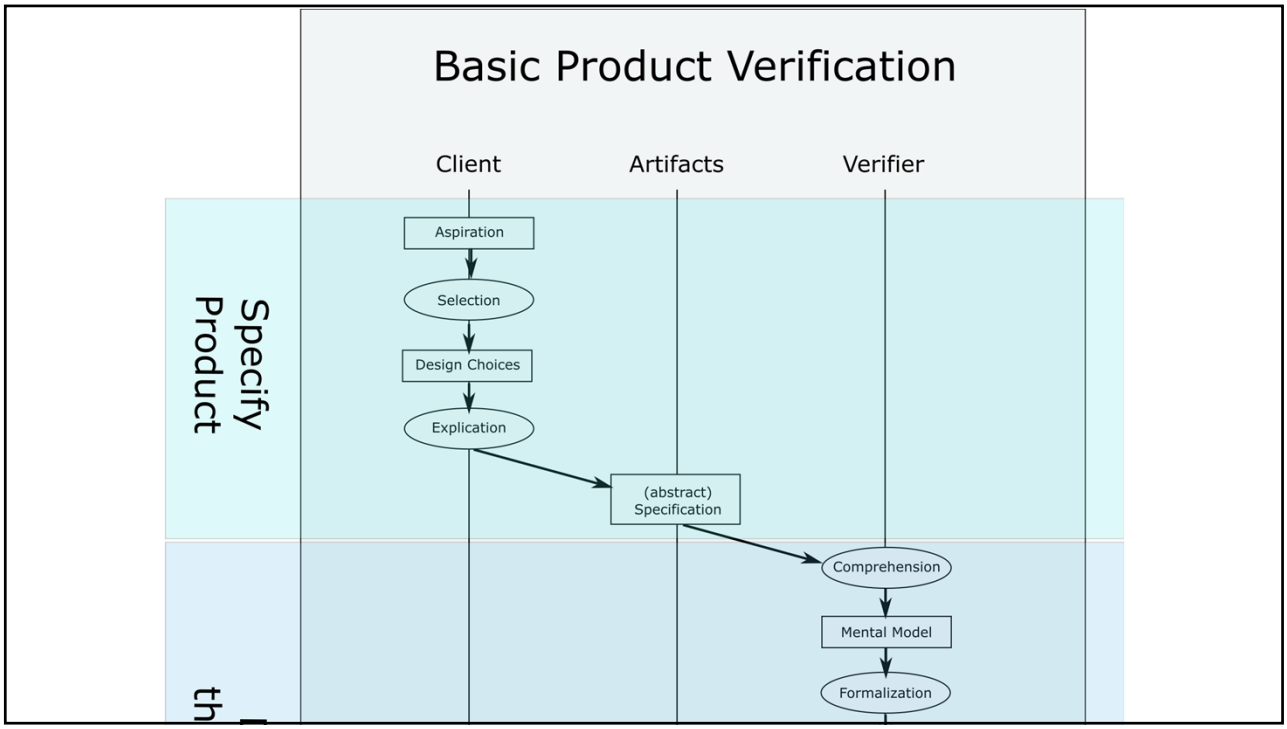
You need to do excellent exploratory work *before* you can do excellent verification *and afterwards* too.

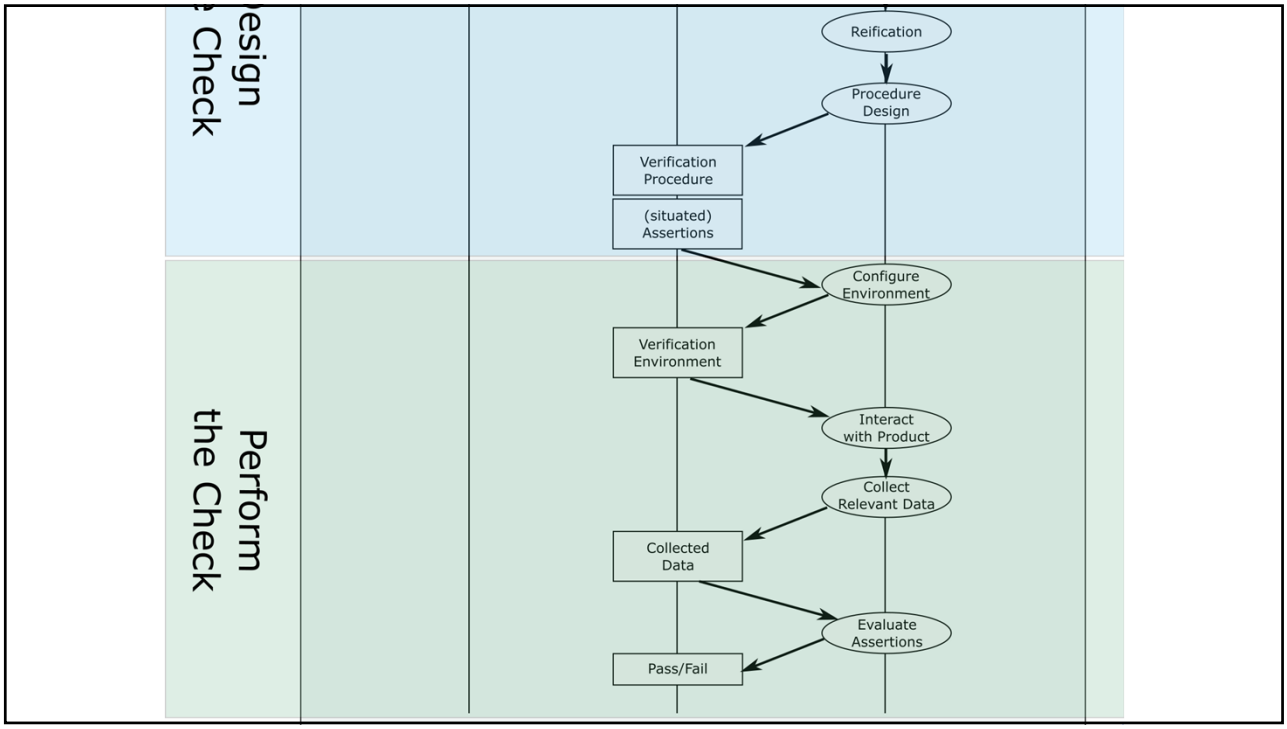
Review
and evaluation
and learning
and sensemaking
and modeling
and studying of the specs
and risk analysis
and recruiting of supporting testers
and observation of the product
and inference-drawing
and questioning
and task prioritization
and coverage analysis
and pattern recognition
and pair development
and decision making
and testability advocacy
and design of the test lab
and preparation of the test lab
and test code development
and tool selection
and making test notes
and preparing simulations
and experimentation
and interacting with developers
and triage
and bug advocacy
and relationship building
and product configuration
and application of oracles
and designing visualizations
and spontaneous playful interaction with the product
and discovery of new information
and preparation of reports for management
and recording of problems
and investigation of problems and working out puzzling situations
and building the test team
and analyzing competitors
and resolving conflicting information
and benchmarking and…

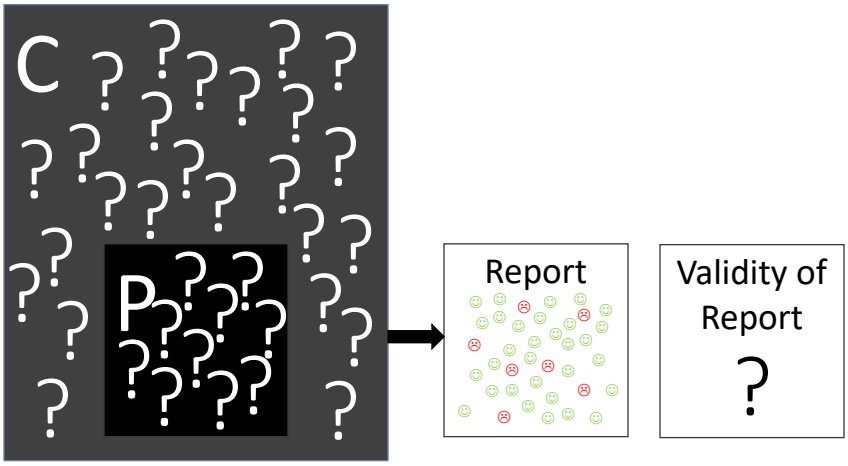


Entity	Definition
Client	Person who matters; whom we serve.
Aspirations	Ideas and values within our clients which relate to want they want.
Design Choices	Choices made by the client, or on the client's behalf, about what to ask for in the product, based on (possibly contradictory) aspirations.
Specification	Abstract statements that represent design choices
Assertion/Example	Situated proposition or artifact that is consistent with some specification.
Product	An artifact intended to fulfill the specification to a reasonable and acceptable degree.
Check (verification)	An algorithmic process that corroborates or refutes an assertion about a particular product in a particular situation.

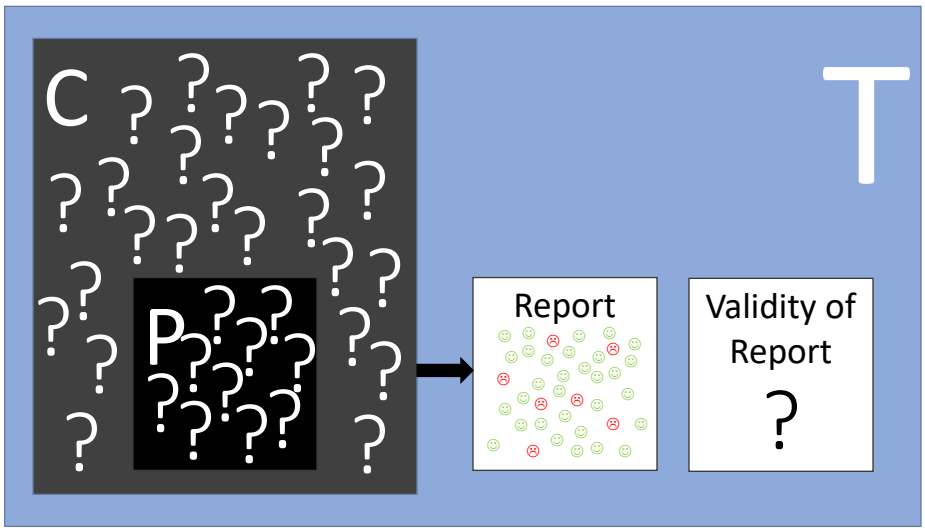
The Logic of Verification
Michael Bolton and James Bach



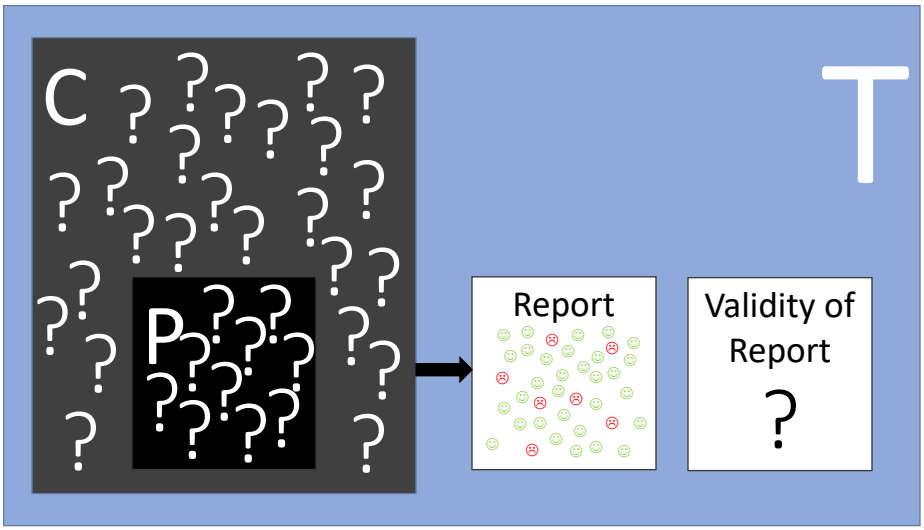




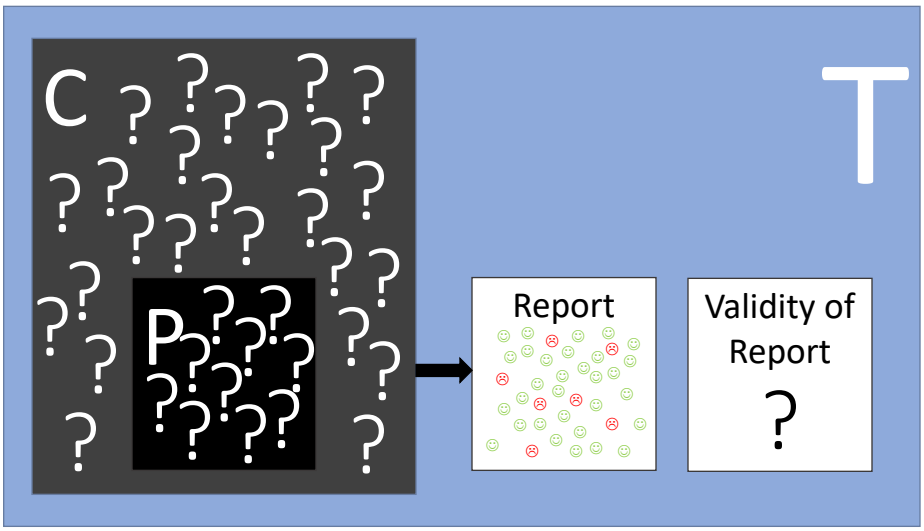
How do we come to a better understanding of the status of the product and the quality of our checks?



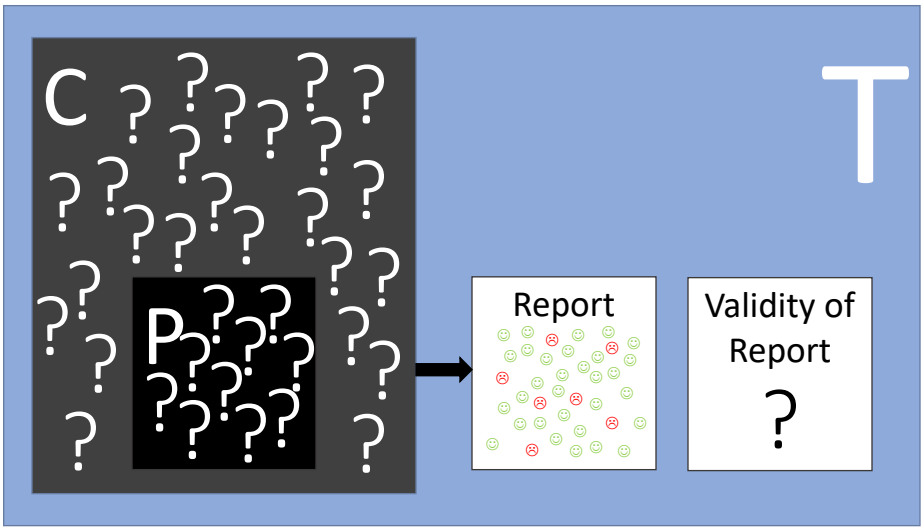
If automated checking is to be valid, reliable, and cost-effective, it MUST be embedded in TESTING.



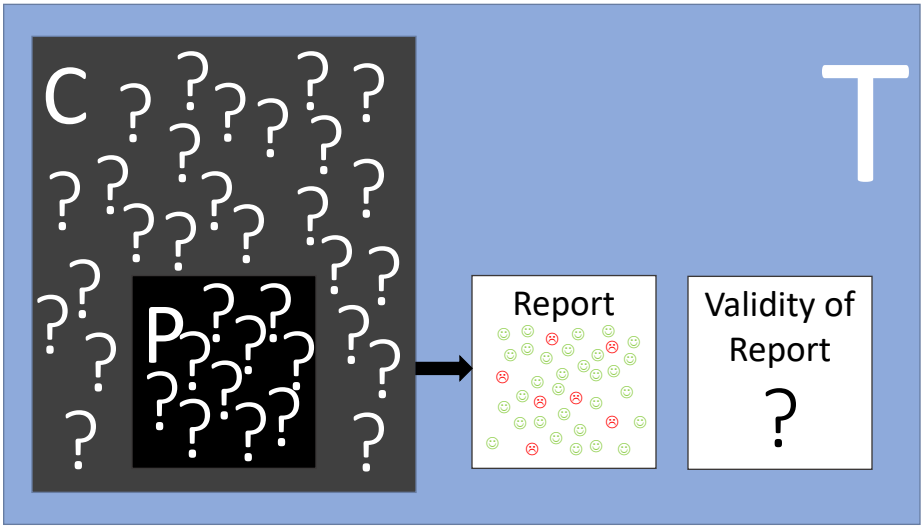
Excellent checking STARTS with exploratory work, and is informed by exploratory work all the way along.



When the product fails a check, excellent testing is required to investigate and pinpoint the failure.



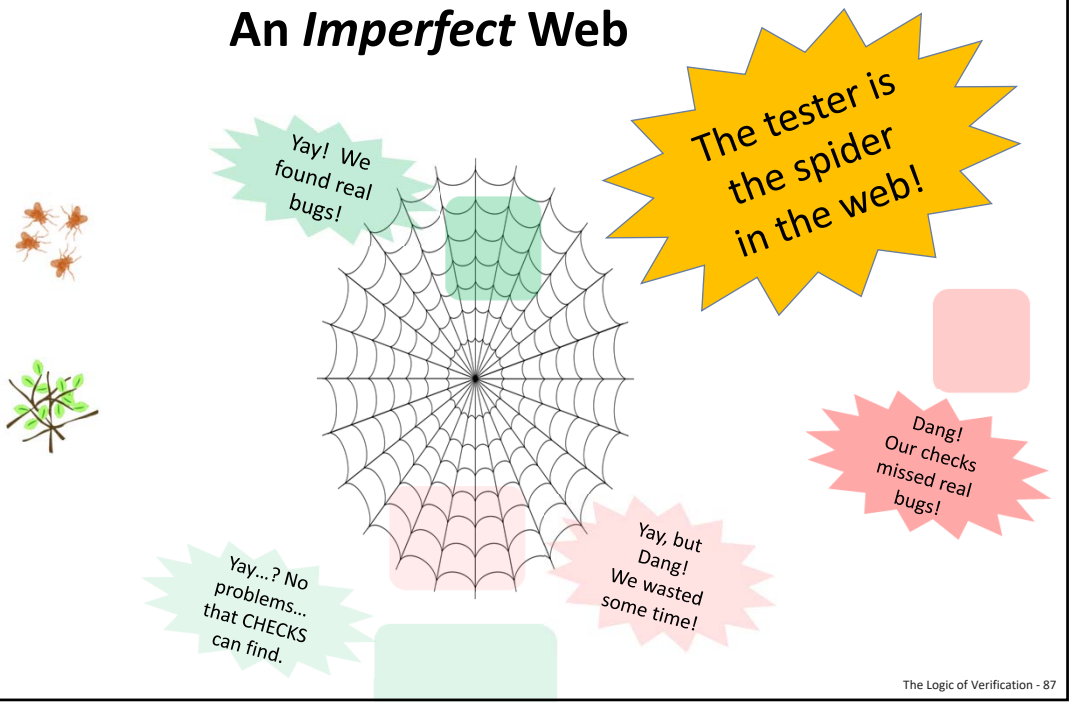
Excellent testing is required to continuously evaluate and maintain checks for accuracy and relevance.



We must question, study, investigate, observe, diversify and challenge our models, checks, and reports, *and* our ideas about them.

Verifications Form a Web

An *Imperfect* Web



"Researchers are increasingly coming to realise that social spiders also sort themselves according to their individual personalities…

"By paying close attention to individual spiders, [researchers] have discovered that certain spiders are more likely to spend their days attacking predators, while others are more likely to repair the webs, help keep parasites away, clean the web, rear the young, and so on."

http://www.bbc.com/earth/story/20160122-meet-the-spiders-that-have-formed-armies-50000-strong

Way More Than Verification!

Testers are the Spiders in the Web

- Testers prepare, supervise, interpret, and maintain **checks and tests**.
- Testers **explore and play and learn** and build mental models of the product and its risks.
- Testers explain and justify their **strategy and status**.
- Testers seek and remove **blinds spots** in test strategy.
- Testers look for ways to refresh and improve the **value of the testing** over time.
- Testers adapt test strategy to the best current knowledge of **product risk**.
- Testers adapt test strategy to the **project context**.

Fixation on verification can lead to inadequate coverage: poor sampling, low diversity and weak oracles.

Workarounds to the Limits of Verification

- Instead of verification, consider *falsification*.
  - We CAN'T verify the hypothesis that the product is okay, but we CAN falsify that hypothesis.
  - When we look for problems *diligently* and don't find them, we can make a better inference that the product is okay.
- Instead of validation, consider *assessment*
  - To assess something is to develop opinions on it.
  - You can have opinions about all kinds of things that cannot be verified
  - Our goal is to develop an *informed* opinion of the product.
- Apply safety language
  - "We *have not seen* any bugs *so far*."
  - "We *are not aware* of any problems *yet*."

Conclusions

- *Excellent* verification is *part* of a testing process that includes not only questioning of the product, but also questioning of the ways in which we check it and test it.
- Verification (in the form of automated checks, formally scripted checks performed by humans, or other forms of fact checking) may be useful, but it falls short of *testing*.
- To test is not only to verify, but to *investigate* and to *challenge*.
- Automating checks reduce execution time, but at some cost in development, maintenance, and interpretation. How big a cost?
- Automated checks can be used to test more broadly and more deeply. Consider diversifying the focus of your checks.
- Many things can't be checked. Excellent testing focuses on exploring and investigating many kinds of risk. Doing this requires many kinds of coverage—not only functional coverage.

The Logic of Verification

Michael Bolton
http://www.developsense.com
michael@developsense.com
Twitter: @michaelbolton

James Bach
http://www.satisfice.com
james@satisfice.com
Twitter: @jamesmarcusbach

A Word from Our Sponsor (Me)

- Rapid Software Testing is a course, a mind-set, and a skill set about how to do excellent software testing in a way that is very fast, inexpensive, credible, and accountable. I co-author RST with James Bach.
- I teach RST in classes for testers, developers, managers, business analysts, documenters, DevOps people, tech support…
- I also offer advice and consulting on testing and development to managers and executives.

http://www.developsense.com