



Getting Started with Requirements Based Verification

DAC2008

Dr. David Robinson

david.robinson@verilab.com

- Overview of the Requirements Based Verification Flow

- Requirements gathering
 - ▣ Brainstorming faults and failures

- Risk analysis and prioritisation

{requirement based verification}

*“Verification is a big job. Treat it like one.
Sweeping it under the rug won’t make it go away”*

What are we Trying to Solve?

4

Project Costs

Out of control, fire fighting, and cancellation

Quality Costs

Low quality and unverified designs shipped

Personnel Costs

High stress and high staff turnover

Just because you don't know about something doesn't mean it doesn't exist. You will find out at some point!

Whether or not you will have the time to deal with it is a different matter.

The Planning Phase

5

Verification requirements

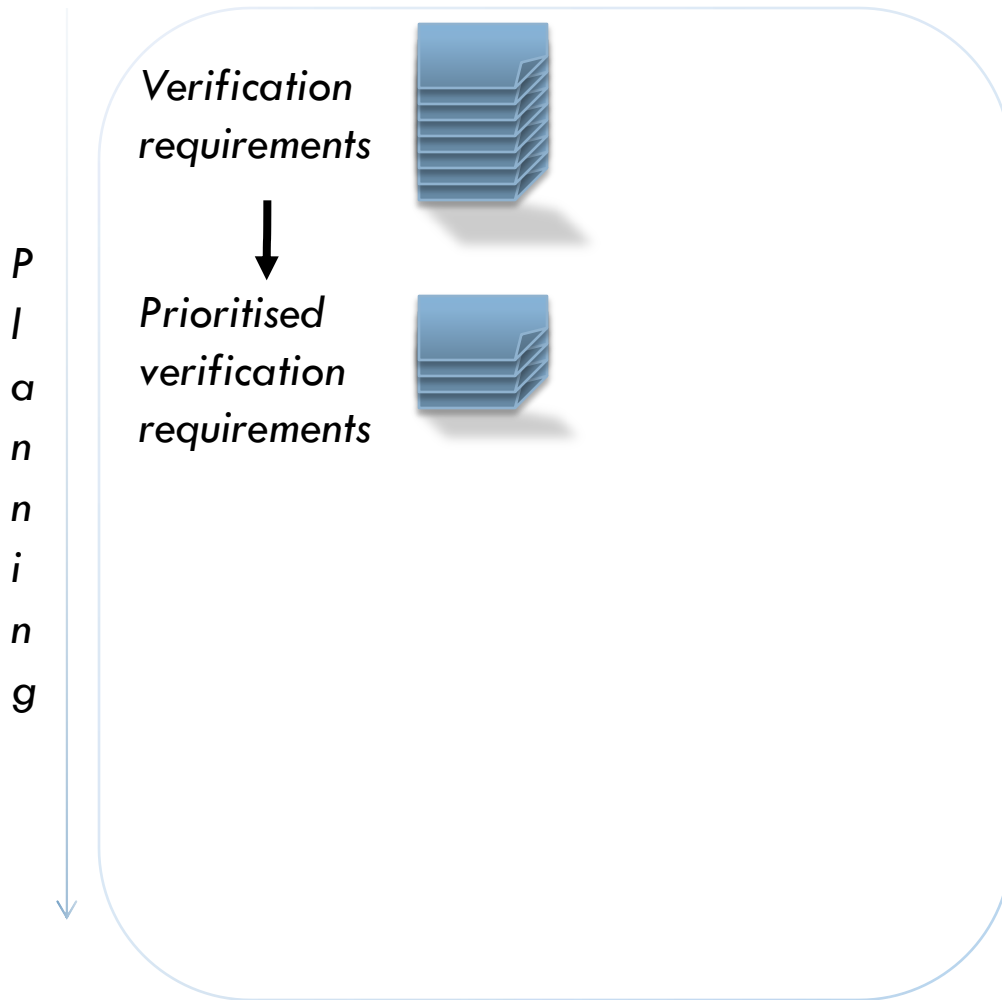


*P
l
a
n
n
i
n
g*

1. *Work out what you would verify given infinite resources*

The Planning Phase

6



1. Work out what you would verify given infinite resources

2. Decide what you will verify

The Planning Phase

7

P
l
a
n
n
i
n
g

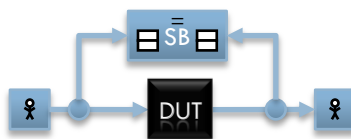
Verification requirements



Prioritised verification requirements



Testbench requirements



1. Work out what you would verify given infinite resources

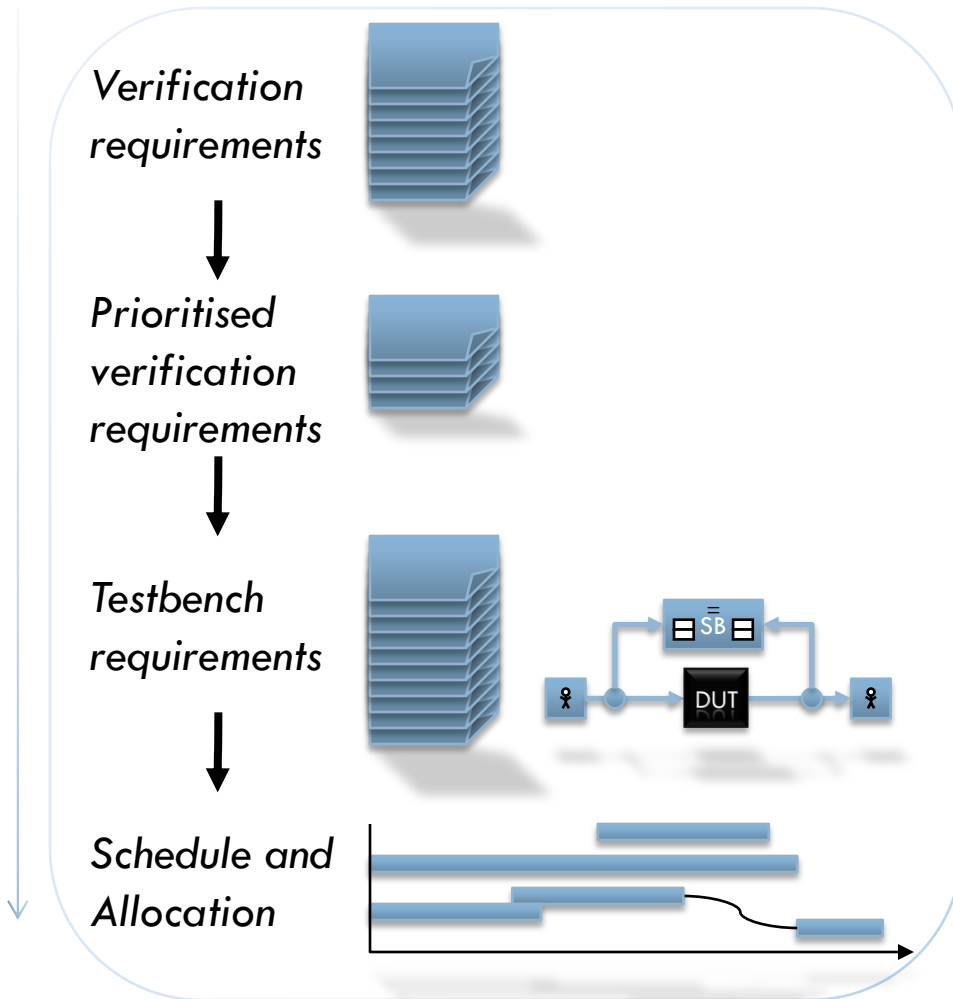
2. Decide what you will verify

3. Decide how you will verify it, and what's involved in that

The Planning Phase

8

P
l
a
n
n
i
n
g



1. Work out what you would verify given infinite resources

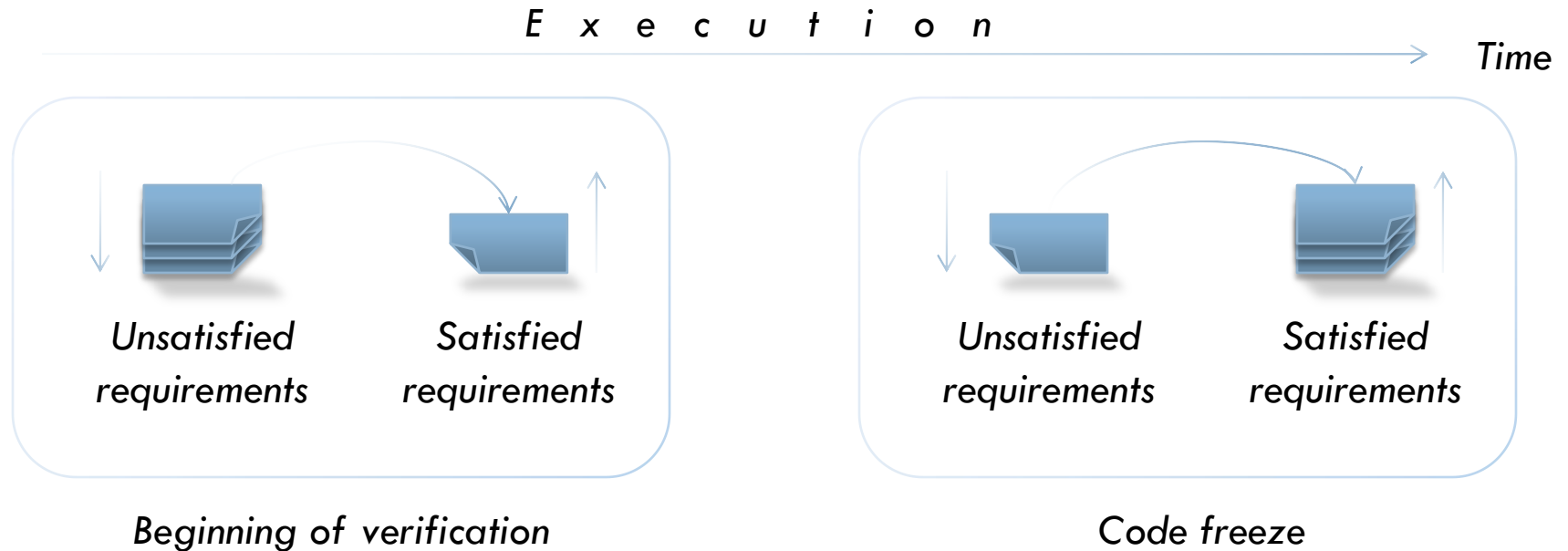
2. Decide what you will verify

3. Decide how you will verify it, and what's involved in that

4. Estimate how long it will take, and what resources you need

The Execution Phase

9



Unsatisfied Requirement : One which hasn't been fully verified

Satisfied Requirement : One which has been fully verified

The Payoffs

10

Better Schedule

Starts accurately, and stays there due to less rework

Better Quality

Early risk mitigation, and detailed verification

Better Visibility

Results (not effort), remaining work and risk

It's better to get bad news early than late

{requirement gathering}

“People talk about requirements gathering as if they’re just lying around, waiting for you to pick them up”

Example Verification Requirement

12

bus-if.read.1

Check: Check that we can only read from the registers when the slave is actually selected.

Conditions: htrans in [NONSEQ, SEQ, BUSY] x hsize in [8, 16, 32] x hburst in [SINGLE, INCR] x hsel in [0, 1];

Importance: 1

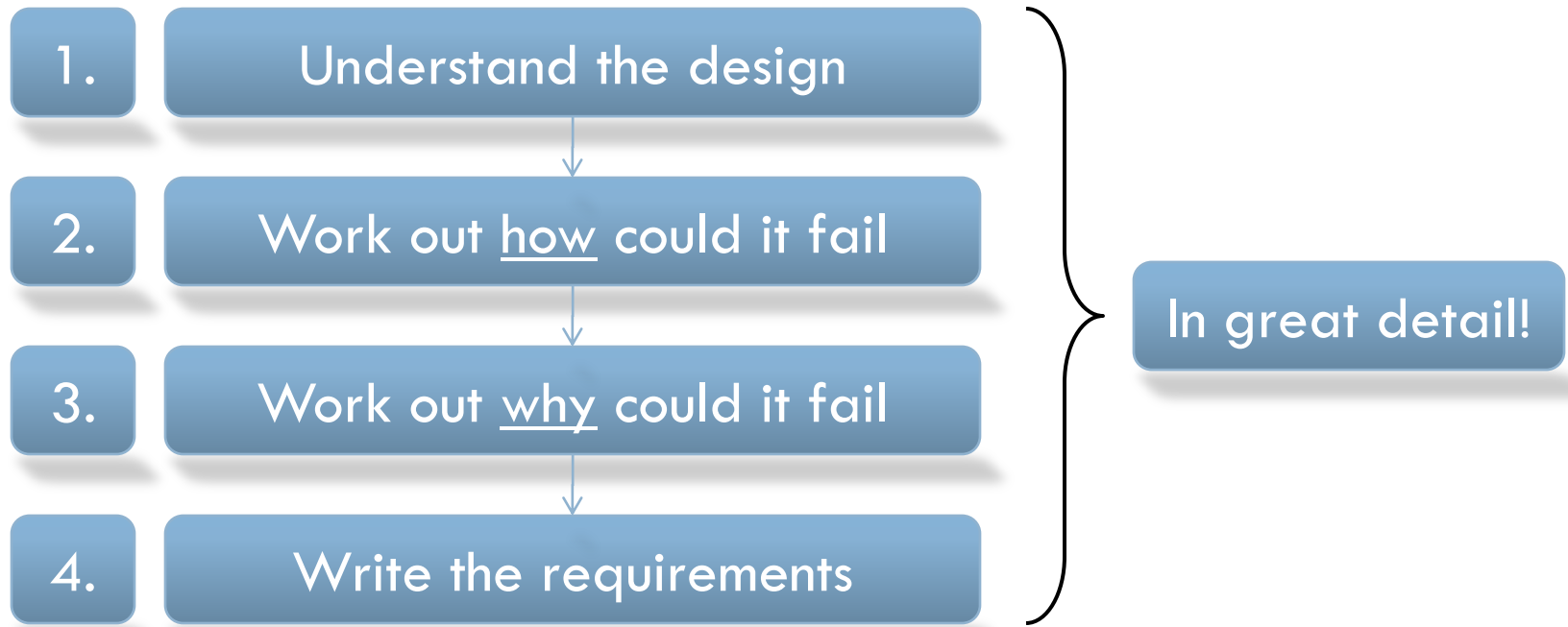
Risk: 3

Status: Reviewed

%Covered: 10%

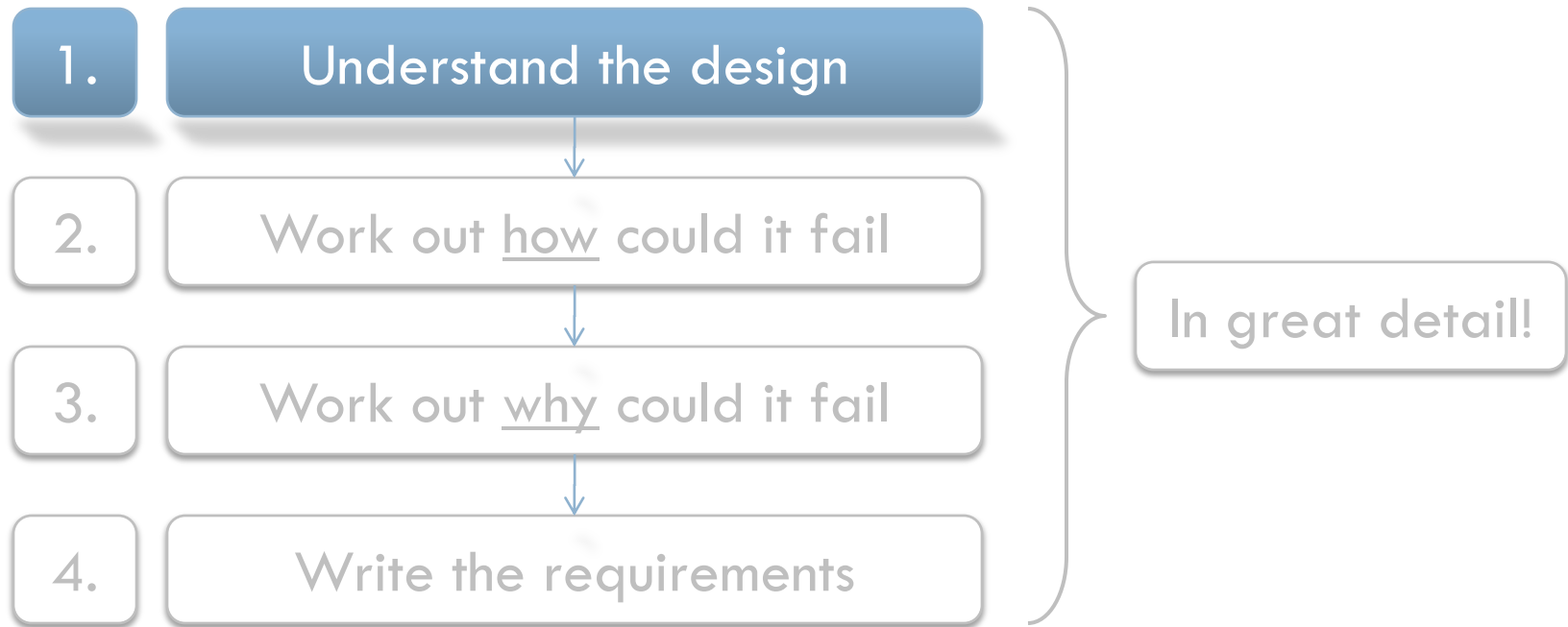
Finding Verification Requirements

13



Finding Verification Requirements

14



Operations

15

Operation

Transmit Data Frame

Operation : Something that the design is meant to do

Operation Outputs

16

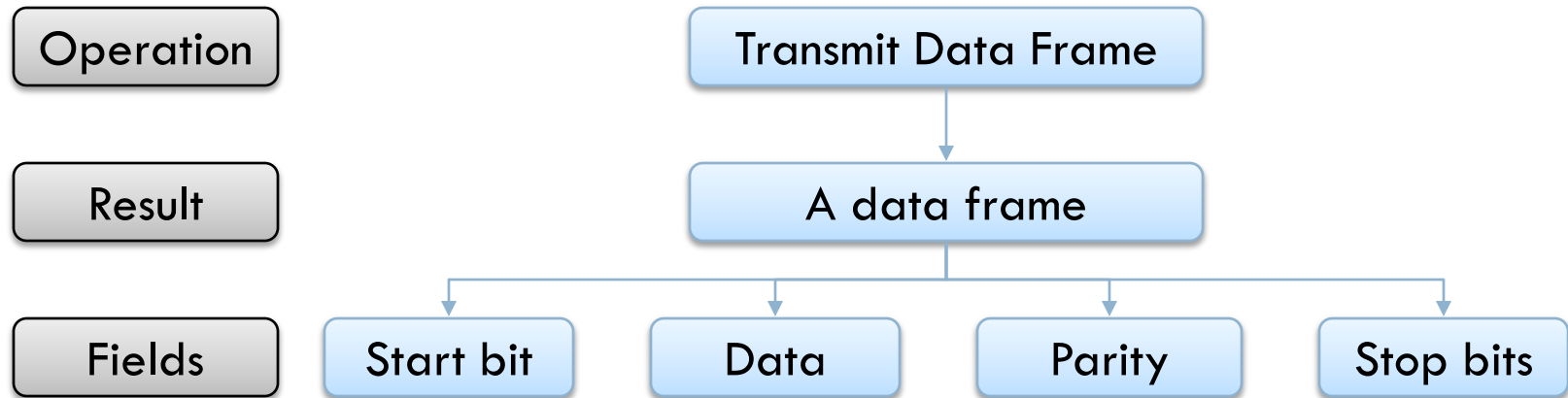


Operation : Something that the design is meant to do

Result : Something produced by an operation

Operation Outputs

17



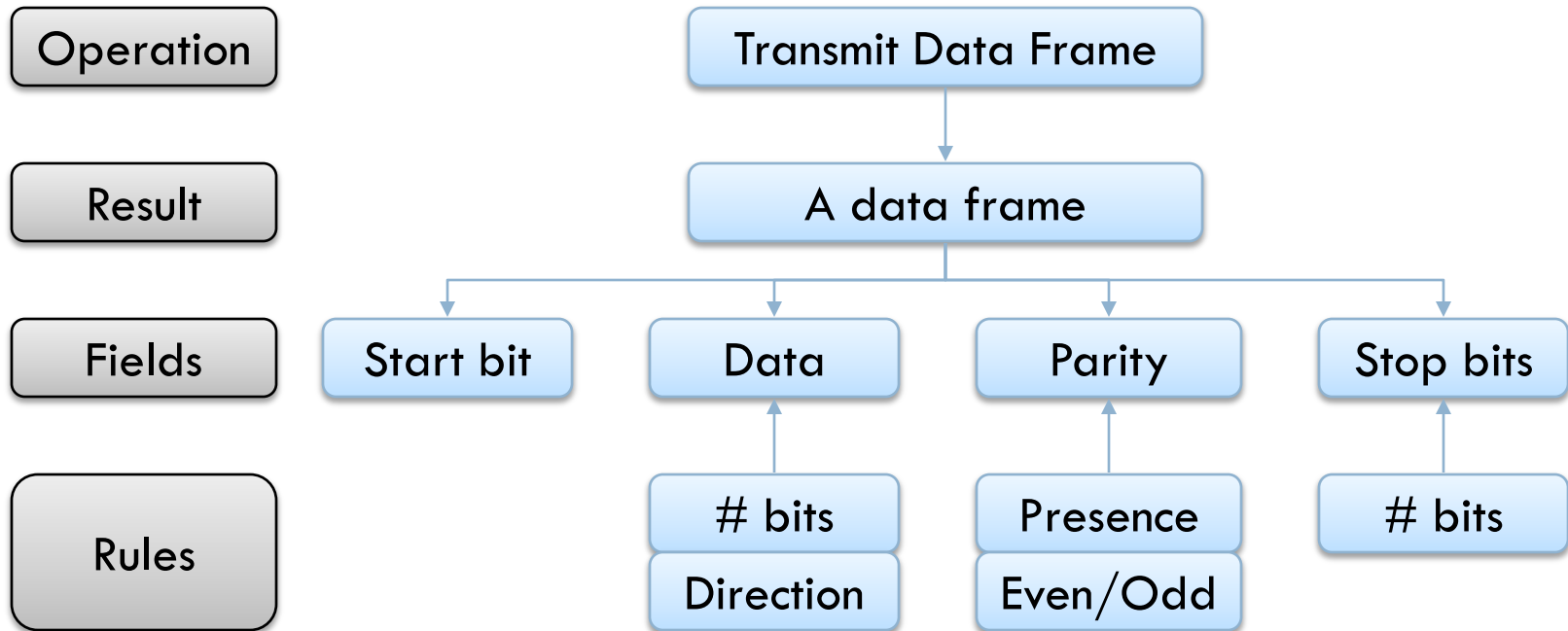
Operation : Something that the design is meant to do

Result : Something produced by an operation

Field : An identifiable part of a result

Operation Outputs

18



Operation : Something that the design is meant to do

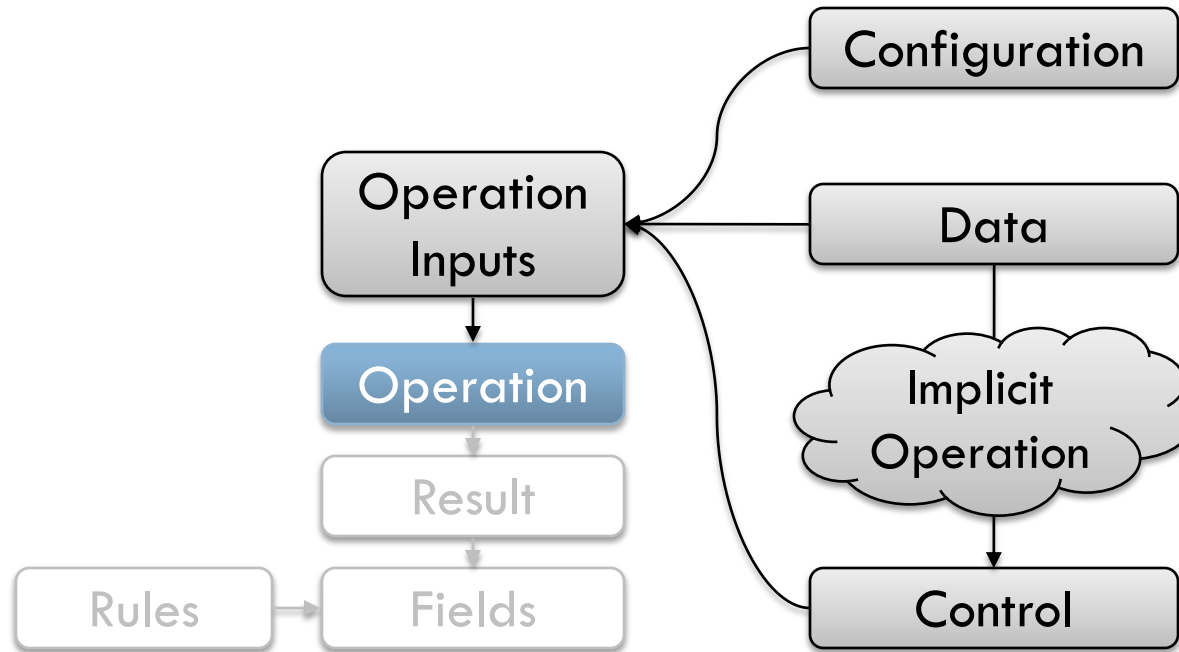
Result : Something produced by an operation

Field : An identifiable part of a result

Rules : How the data is stored, encoded or interpreted in a field

Operation Inputs

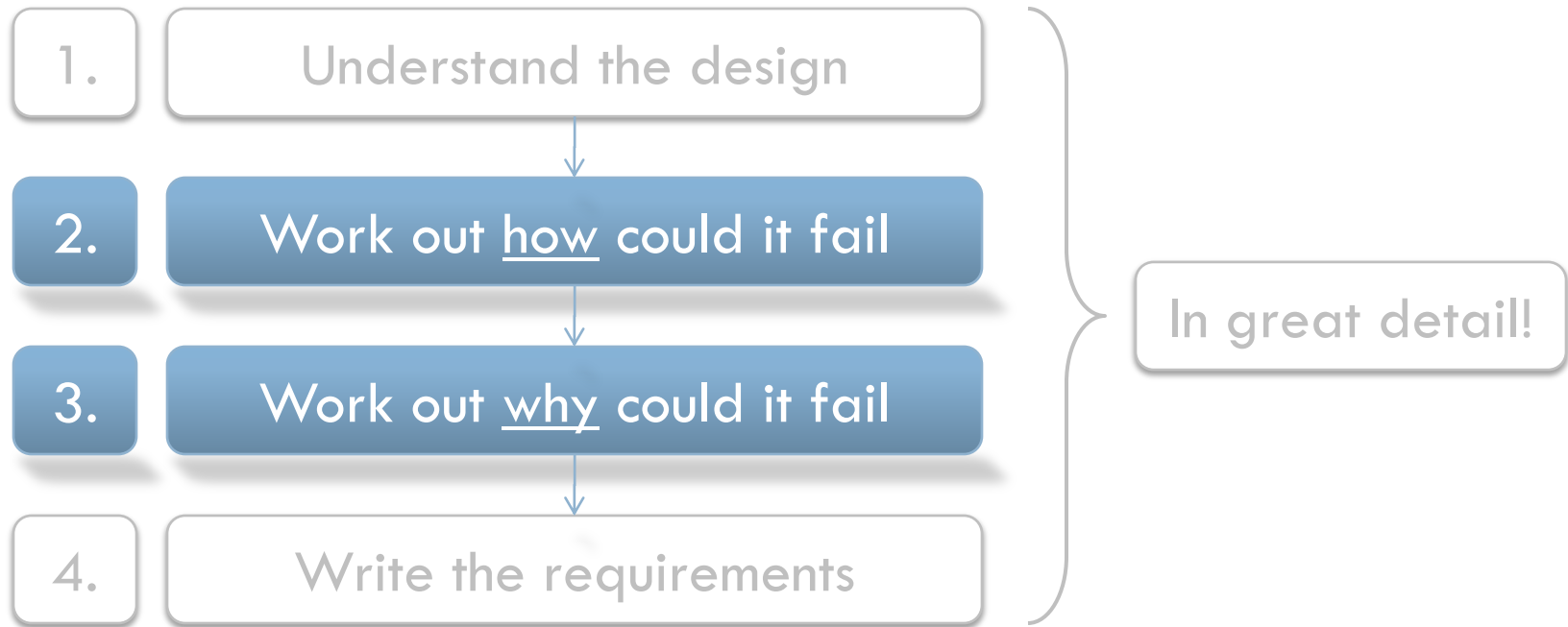
19



What can the inputs be?

Finding Verification Requirements

20



It's not enough to show the design works. You have to show that it doesn't fail.

Failure Modes and Effects Analysis (FMEA)

21



For each operation

How could it fail?

Why could it fail?

Do we care?

FMEA Output

22

What would a failure look like?

Operation	Failure Mode
Transmit Data Frame	Parity value incorrect



FMEA Output

23

What could cause it to fail?



Operation	Failure Mode	Failure Cause
Transmit Data Frame	Parity value incorrect	<p>Parity calculated over entire frame and not just data</p> <p>Parity type configuration register misinterpreted</p> <p>Parity type misunderstood</p> <p>Stick parity not implemented</p> <p>Parallel parity calculation & dataBits changes between frames (e.g. 8 to 5). The now unused bits corrupt the new value</p>

FMEA Output

24

What's the risk of it failing?

Operation	Failure Mode	Failure Cause	Risk Exposure
Transmit Data Frame	Parity value incorrect	Parity calculated over entire frame and not just data	6
		Parity type configuration register misinterpreted	15
		Parity type misunderstood	4
		Stick parity not implemented	20
		Parallel parity calculation & dataBits changes between frames (e.g. 8 to 5). The now unused bits corrupt the new value	4



Risk Exposure

6

15

4

20

4

FMEA Output

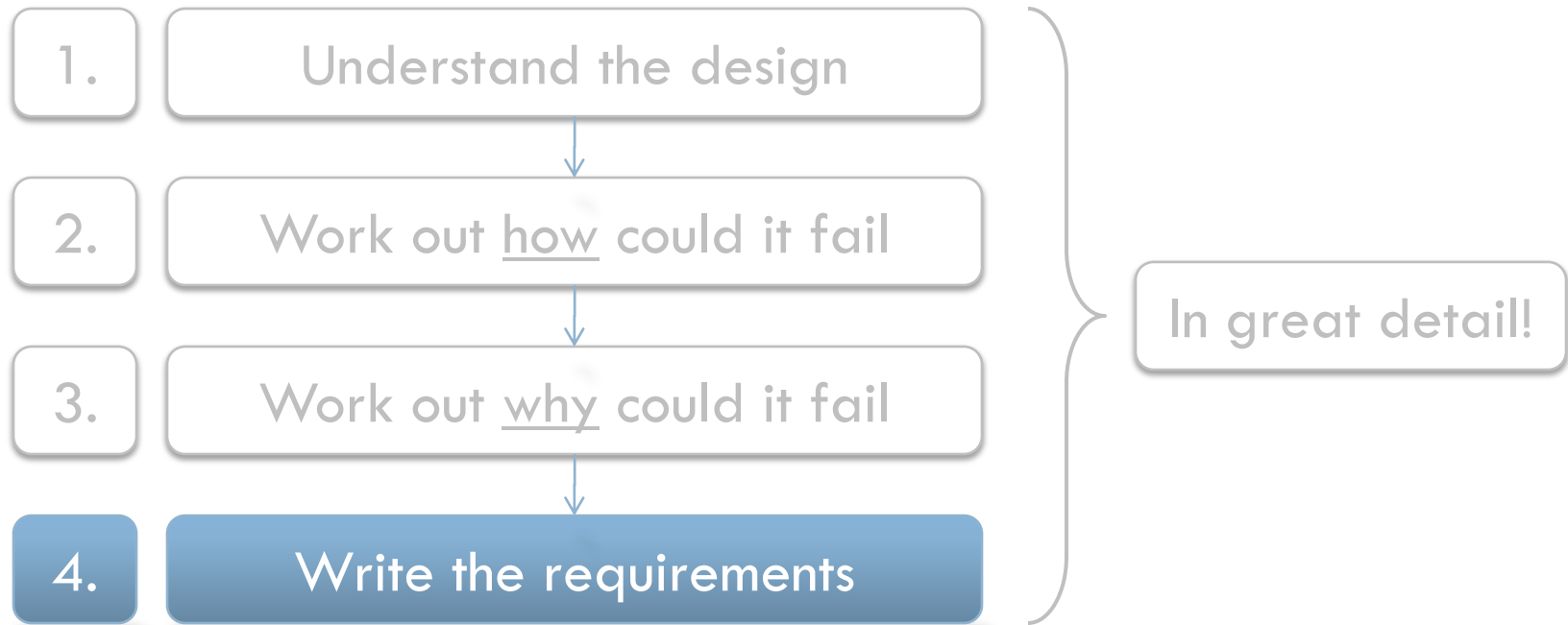
25

Ignore low risk items

Operation	Failure Mode	Failure Cause	Risk Exposure
Transmit Data Frame	Parity value incorrect	Parity calculated over entire frame and not just data	6
		Parity type configuration register misinterpreted	15
		Parity type misunderstood	4
		Stick parity not implemented	20
		Parallel parity calculation & dataBits changes between frames (e.g. 8 to 5). The now unused bits corrupt the new value	4

Finding Verification Requirements

26



- Add one requirement to check that the operation works

- Add one requirement per failure mode
 - ▣ if not covered above

- Add a new requirements for any failure-mode/failure-cause combinations that requires special attention

Verification Requirements

28

Req	Check	Conditions	Risk Priority
tx.frame.1	<p>Check that the UART transmits a data frame correctly:</p> <p>start → data → [parity] → stop</p> <p>Data must be sent LSB first with the correct # of bits...</p>	<p>dataBits = [5, 6, 7, 8]</p> <p>x data = [0..255]</p> <p>x parity = [OFF, ON]</p> <p>x pType = [ODD, EVEN]</p> <p>x stopBits = [1, 1.5, 2];</p>	1
tx.frame.2	<p>Check that previous data values do not affect the parity calculation</p>	<p>parity = [ON]</p> <p>x pType = [ODD, EVEN]</p> <p>x data = [0..255]*</p> <p>x (dataBits = [N]</p> <p>→ dataBits = [M (<N)];</p> <p>* Data must be chosen carefully so that legacy unused bits would affect new calculation</p>	4

{brainstorming failures and faults}

“Brainstorming is easier if you know what you’re looking for”

Brainstorming Failure Modes

30

- Know what a “correct” result looks like, and consider all deviations from it

- 4 failure classes to get you started:
 1. Unscheduled execution
 2. Failure to start when required
 3. Failure to stop when required
 4. Failure during execution

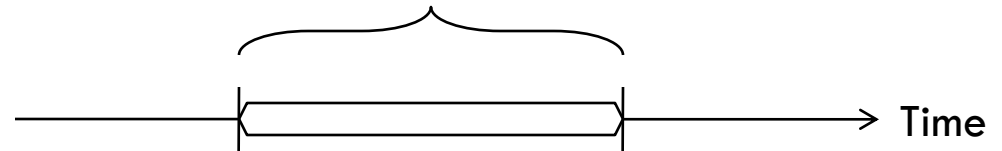
Temporal Failures

31

Operation

Access Registers

reg_sel field must only be active between these points



Premature Activation



Delayed Activation



Partial Activation

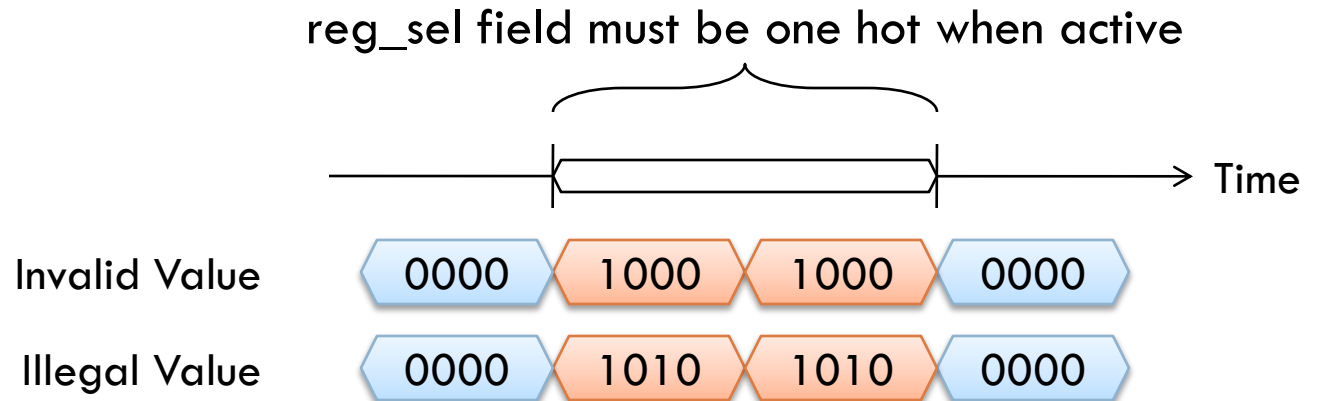


Consider how the field can vary temporally within the result

Value Failures

32

Address 0xF000 accessed. `reg_sel` should be 0001



Consider how the value of the field can fail

Encoding Failures

33

Data = -5, encoded as 2's complement

1011

1010

1's complement

1101

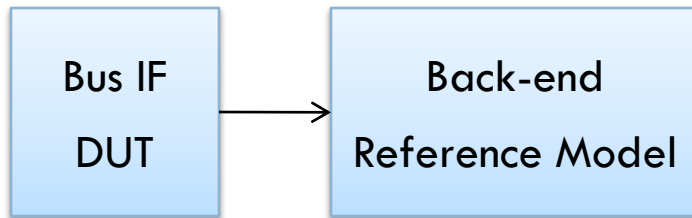
Sign and magnitude

Consider how the field can be incorrectly encoded

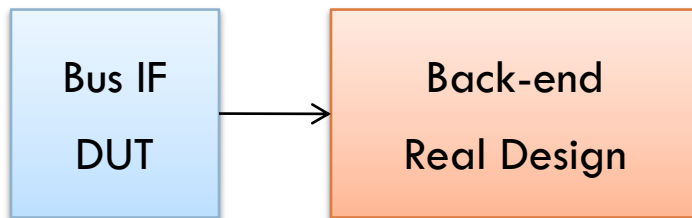
Too much Detail?

34

- What if your testbench is accidentally tolerant to these failures?
 - consider an illegal value on `reg_sel`



1. Uses "if" statements with `reg_sel`
2. `0011` treated as `xxx1`
3. No failure detected

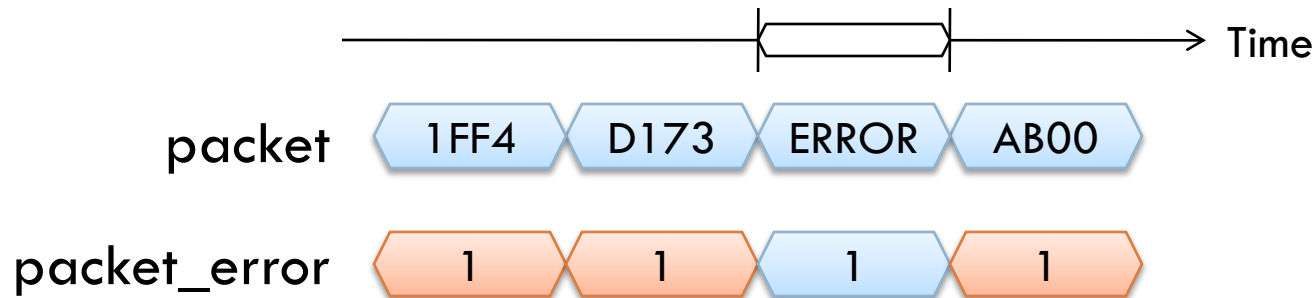


1. Uses "case" statements with `reg_sel`
2. `0011` causes deadlock (infinite wait states)
3. Customer isn't very happy

Too much Detail?

35

Operation	Report errors on incoming packets	
Req	Check	Conditions
rx.error.1	Check that packet_error is asserted when an error is detected in an incoming packet	packet_type = [A, B, C] x packet_error = [TRUE, FALSE]



1. *packet_error* wrong almost all of the time
2. No checker in place to catch that failure mode
3. No failure detected

Brainstorming Failure Causes

36

- Use “5 Whys” for root cause analysis

- It might help to:
 - ▣ understand the design’s implementation
 - ▣ consider external failures:
 - workflow management
 - change control
 - revision control
 - scripts
 - spreadsheets

{risk analysis and prioritisation}

“Not all requirements are created equal. Why waste time on the unimportant ones?”

Implementation Priority

38

- Implementation Priority lets the team specify the order in which to verify requirements
 - ▣ useful for phased releases
 - ▣ uncovers the hidden value of each requirement



Risk Exposure

39

Risk Exposure = *likelihood* of failure x *impact* of failure

Likely (1) → Not likely (5)

Unacceptable (1) → Acceptable (5)

1

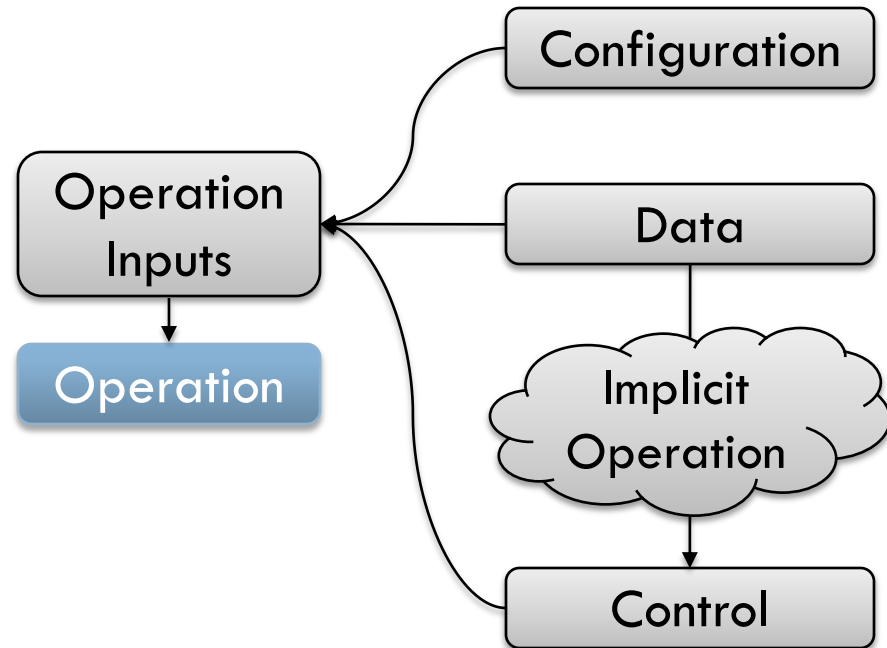
.... Extremely risky – lots of effort needed

25

.... Not worth any effort at all

Likelihood of a Failure Occurring

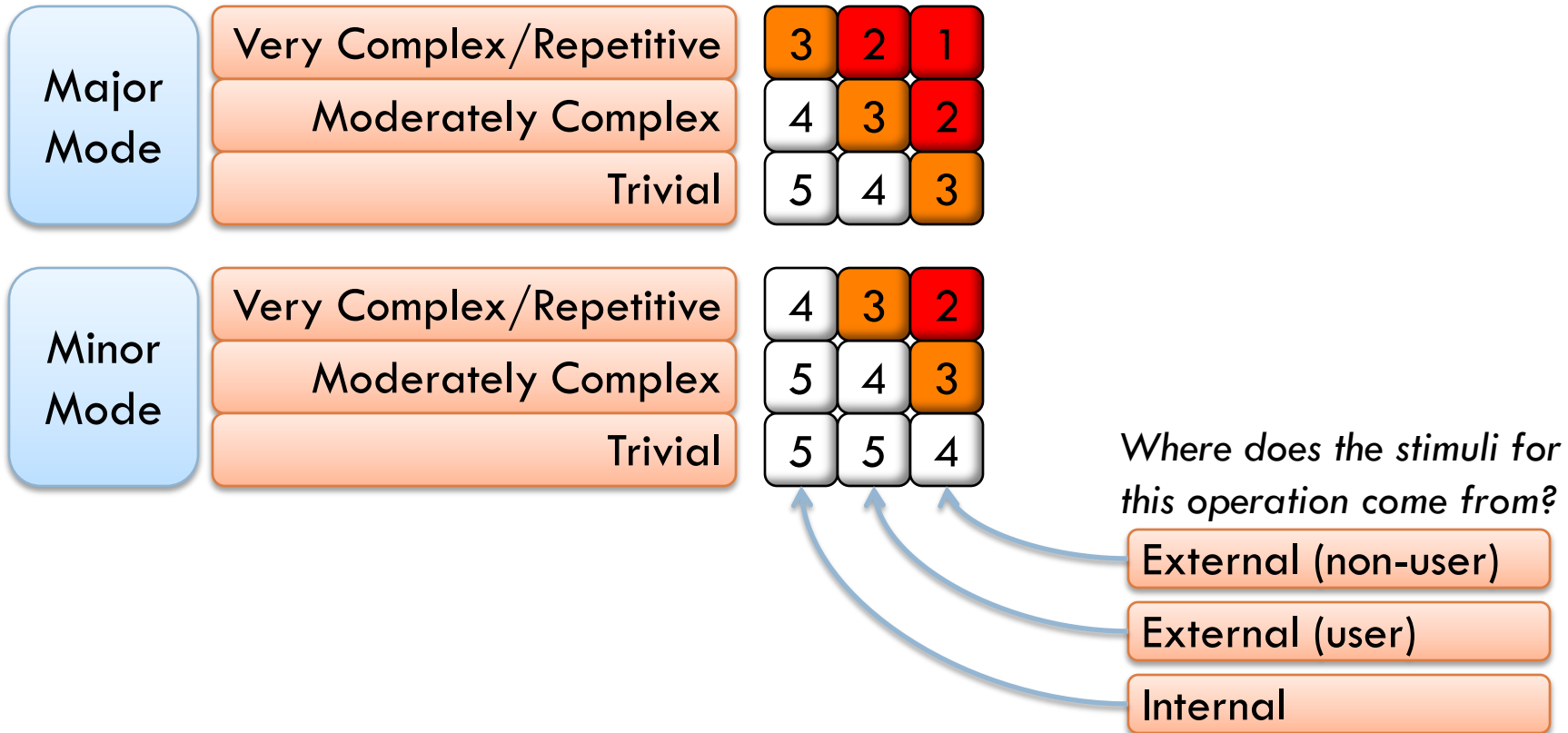
40



- What's the likelihood of:
 - ▣ the operation being wrong?
 - ▣ the inputs being wrong?
 - ▣ the operation being used?

Likelihood of a Failure Occurring

How complex is the design for this operation?



Impact of a Failure Occurring

42

What is the worst that will happen?

- Something will be damaged
- The chip won't work
- A major mode won't work
- A minor mode won't work

1	1	1	1
3	2	2	1
4	3	2	1
5	3	3	3

Is there a workaround?

No

User behaviour

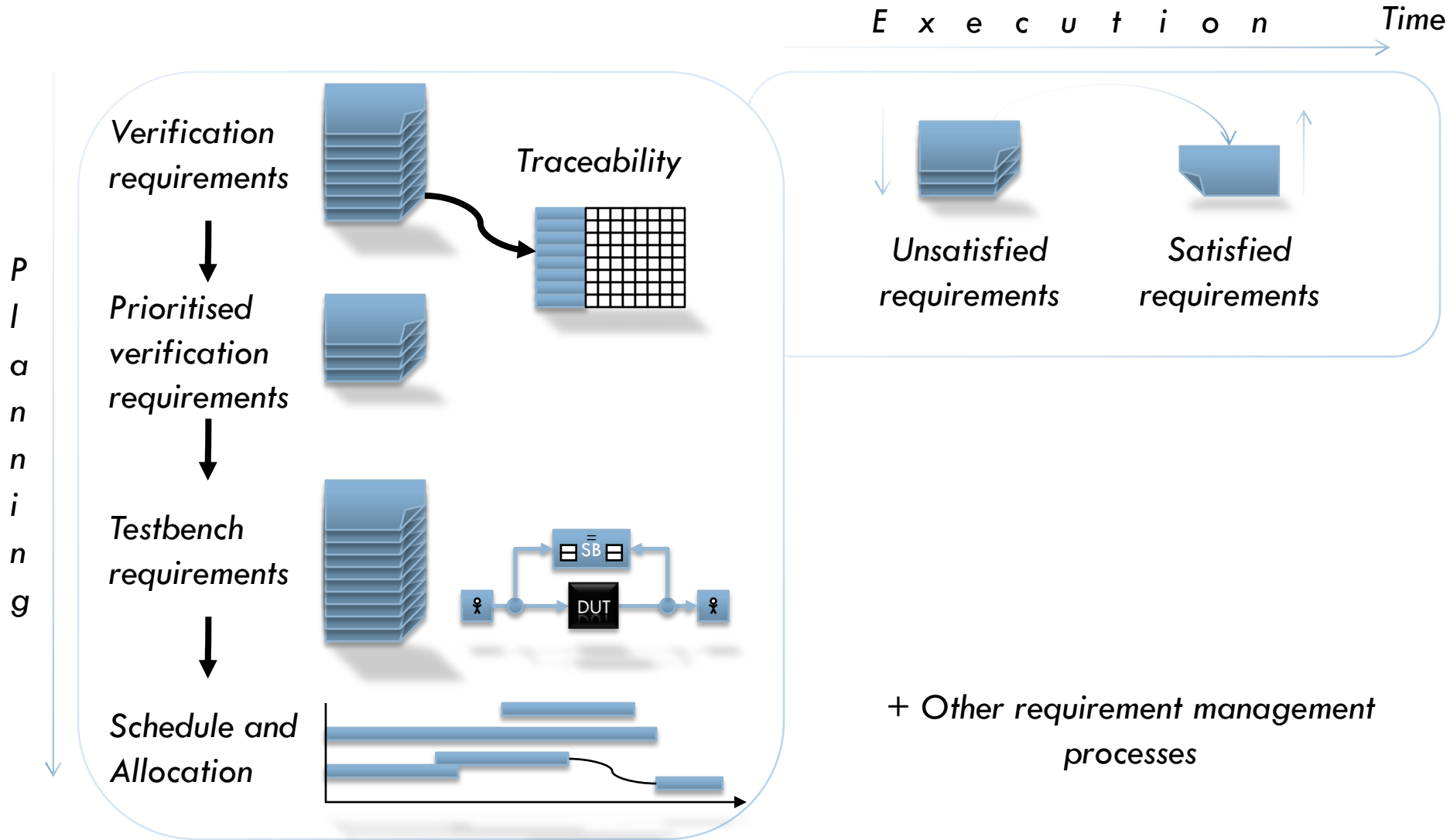
A hardware one

A software one

{where next?}

Requirement Based Verification

44



The End

□ david.robinson@verilab.com