Proven Strategies for Better Verification Planning

DVCon 2022 Workshop

accellera SYSTEMS INITIATIVE

verilab

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES VIRTUAL FEBRUARY 28 - MARCH 3, 2022

Presenters

Paul Marriott - Verilab Consultant

Jeff Vance - Verilab Consultant

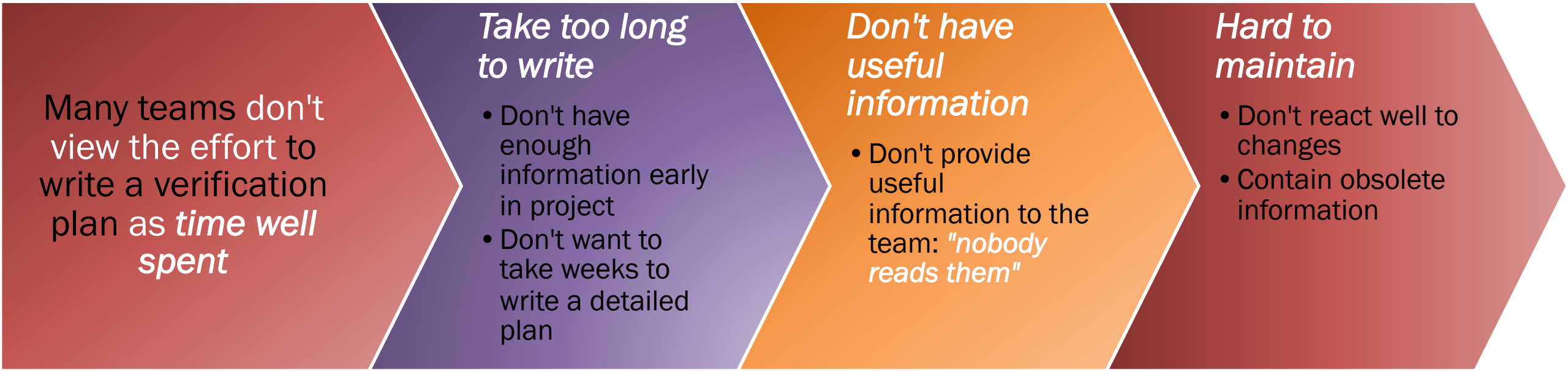
Jeff McNeal - Verilab Consultant

accellera SYSTEMS INITIATIVE

verilab

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES VIRTUAL FEBRUARY 28 - MARCH 3, 2022

Typical Engineering Team Situation



What is a Verification Plan?

Scope	Verification Requirements	*Project Plan**	*Implementation Plan**	Reusable Components
• Full Feature • Integration Only • Simulation • Formal	• Feature Extraction • Scenarios • *Coverage Plan**	• Resources • Budget • Schedule	• Testbench Environment • Stimulus Plan • Checking Plan	• Agents • Sequences • Assertions • VIP

*optional, high-level planning only

*can be separate document

Key Workshop Topics

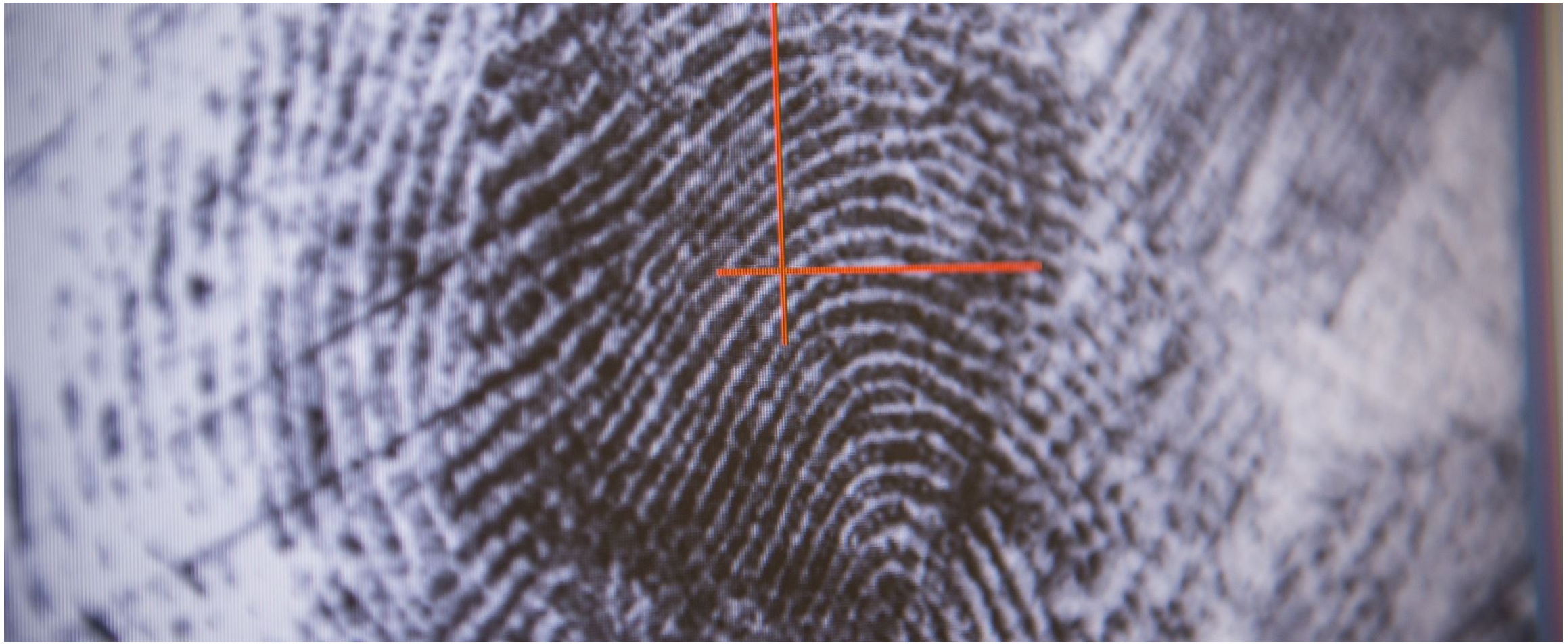
DUT Feature Identification

- Isolation
- Scenario Classification
- Weakness analysis

Scheduling

- Divide work into deliverables
- Organize deliverables for Linear Progress

DUT Feature Identification



Verification Planning Mindset

Avoid

Testbench Implementation (the HOW)

Features by testbench component

Too many details

Test Lists

Specific coverage bin values

How coverage is sampled

Do

DUT Mindset (the WHAT and WHY)

Features by DUT functionality

High-level decisions

Scenario descriptions

Quantity/kinds of coverage bins

When/where coverage is sampled

Feature Analysis Strategy

1

Feature Isolation

2

Classify Scenarios

3

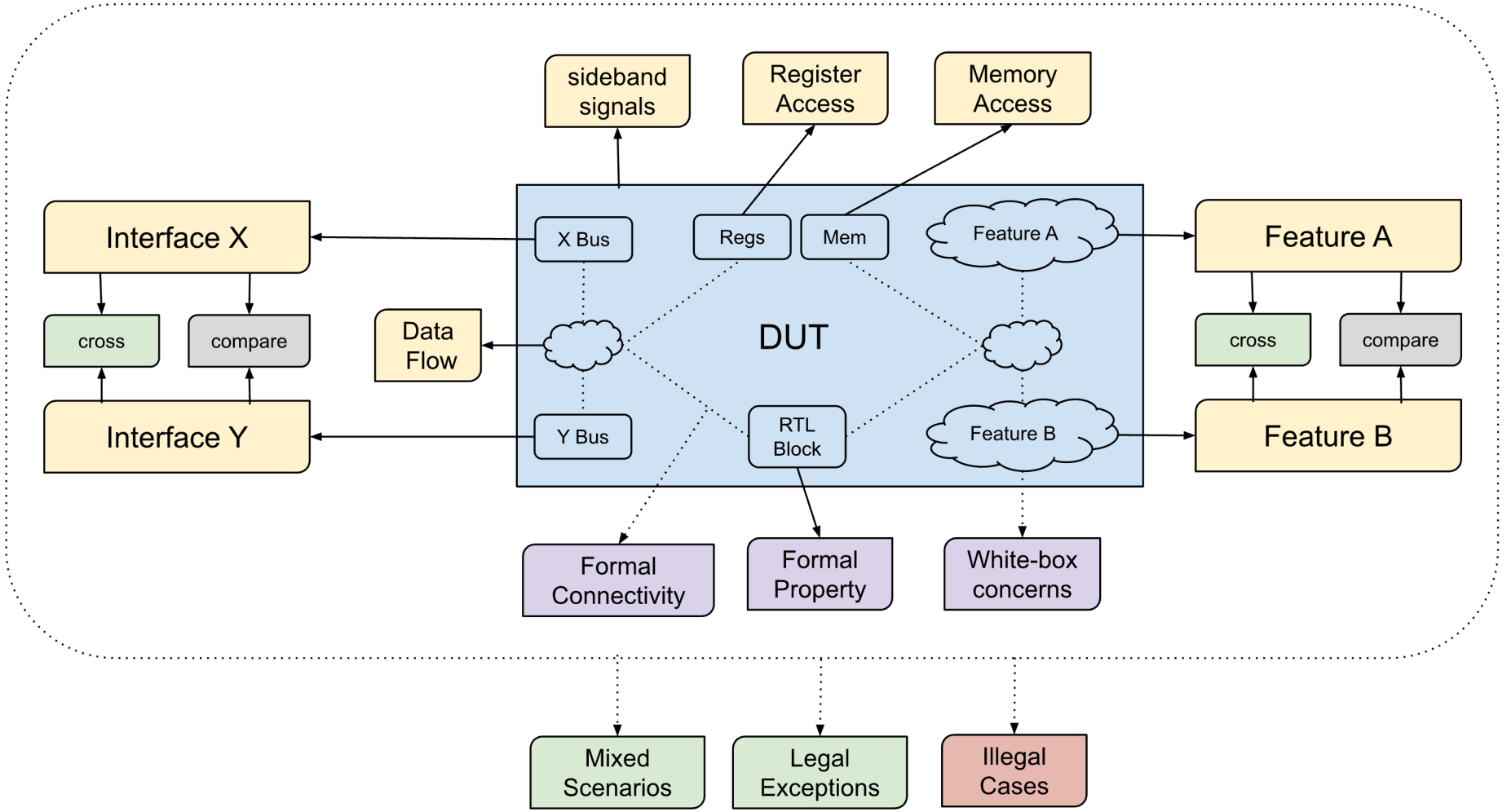
Inspect Weaknesses

Feature Isolation

Isolate focus to a portion of the DUT

- By Design Spec
- By RTL Block
- By Large-Scale feature (across blocks)
- By use-case
- By risk (bug / complexity risk)
- By special case

Feature Categories



Scenario Classifications

Isolated Features

- Analyze key behaviors individually
- Ideal for incremental progress, debug, and sanity regressions

busses | txn types | txn flows

blocks | configs | timings

Mixed Features

- Key combinations of isolated features
- Can be use-cases / special cases

mixed txn types | mixed txn flows

mixed cfg/timing | mixed blocks

Legal Exceptions

- Abnormal cases that are ***supported***
- Must be in design spec!

soft reset | protocol errors | FIFO full

Illegal Scenarios

- Unsupported by design
- Spec must say what is unsupported
- Some tests may stress the design

DUT ignores It | recovers on reset

Future Benefits

Faster Implementation

- Plan will influence Testbench design
- Reduce complexity
- Avoid work duplication

Better Communication

- Status for management & stakeholders
- Collaborate with design team
- Enable new teammates rapidly
- Review/close coverage faster

Execution Flexibility

- Isolate bugs
- Navigate around blocking issues
- Debug problems faster

Accurate Scheduling

- Accurate estimations of effort
- Better prioritization of tasks
- Stay on schedule

Find Weaknesses in the Plan

Analysis Toolbox

- **Correctness:** Is this *valid* ?
- **Precision:** Is this *specific* ?
- **Completeness:** Anything *missing* ?

Avoid Ad-hoc Thinking

- Luck has more influence
- Schedule Risk
- Testbench Rework
- Missed Verification Scenarios
- Bugs found late (or missed)

Apply Structured Analysis

- Directs our thinking to key areas
- Is organized
- Is consistent

Plan Correctness Assessment

Is this technically possible?

- DUT implements this option?
- Is behavior fully specified?
- RTL parameters allow this option?

Do we care?

- Is it a valid use-case?
- Are details relevant to verification?

Precision Assessment

What are we *specifically* verifying?

- DUT Processes Transactions
- Transactions Undisturbed by Event
- Value Ranges
- Event Timing
- Throughput
- Access Rights
- Resource Contention

What is the *context*?

- Feature Dependencies
- Disqualifying Conditions
- Any missing contexts?
- Different goals per context?

Completeness Assessment

List Influencing Variables

- How can variable change?
- How do changes impact DUT?
- List verification requirements per impact

Review each Scenario

- What ***do*** you expect?
- What ***don't*** you expect?
- List verification requirements per expectation

Analyze Feature Cross-Concerns

- Categorize feature combinations
- Categorize all unique outcomes
- List verification requirement per outcome

Scheduling Challenges

Common failure modes:

- Too much detail too soon
- Too little planning
- Too focused on testbench blocks

Too much detail too soon

- Usually good faith effort
- Labor intensive
  - Need to plan every item on feature list
  - Every change requires a detailed plan update
  - Every finished task requires plan update
- Granularity issues
  - Small tasks require small amounts of time (hours)
- Doesn't communicate well with team
  - Hard to tell exactly what is finished and what isn't

Too little detail

"We'll be done when we're done"

- Little to no organization of verification effort
- Can lead to poor communication with other teams
  - Confusion about what has been verified or not
  - Lack of trust in verification team
- Leads to using proxies for progress (coverage, tests)

Too focused on blocks

Suffers from estimation of finished before complete amount of work is known	Changes mean that things that were done now are not	Poor communication outside verification team
• Need substantial testbench architecture work up front • How many lines in 100% ?	• By some unknown amount	• Outsiders don't know testbench details

Clear Communication

- What have we verified?
- What is left to be verified?
- Are we on schedule?
- When will we be finished?
- We can build our own metric using the deliverables we've already defined.

Scheduling: Just Right

Low effort for fast results

- More detail easily added later if necessary

Clear communication of verification status

- Clear to all teams what is done and what is left to do

Flexibility

- Reacts well to changes
- Adapts to differing degrees of documentation completeness

Improving Scheduling Abilities

Two methodologies that reduce effort and increase effectiveness:

- Group work into deliverables
- Organize deliverables for linear progress

What do we mean by deliverable?

- Definition of Done - what will be done for this deliverable
- List of work - what is needed to complete the deliverable
  - Group things that are related to the same feature
  - TB infrastructure, Coverage, Checks, Stimulus
- Effort estimate

Definition of Done

- Unique to each deliverable
- Action not state
- Whole testbench, not smaller parts
- Not necessarily feature verification done

Better Definitions of Done

A block is done when it is completely coded, committed, running & passing in regressions.

Agent coding 80% done

75% of tests written

Boot micro-controller & read IO values

Send single packet through DUT

Register reset value test passes

Better Deliverables: Think Demonstration

A test that sends a single packet through the DUT

A test that checks reset values of all registers

A test that encrypts one AES transfer

100% coverage of a particular feature

Nightly regression script, including notification for the team

Published coverage report from a nightly regression

Web-page with generated documentation of TB

Better Deliverables: Completeness

List all the tasks that will need to be completed for the deliverable

- Testbench work
  - Limit to necessary functionality
  - Include all aspects across agents, stimulus, checkers, etc
- Compute & infrastructure work
  - Scripting, report generation, etc.

Remove anything not essential to this deliverable

Example

Definition of Done:

- Test which encrypts one AES transfer

Work:

- Add AES encryption method
- Add AES decryption method
- Update sequence item
  - update `do_compare()`

Update configuration object

New test

Coverage

- New DUT modes
- Keys

Program Keys

- New key sequence
- New virtual sequence (key + traffic)

Better Deliverables: Effort Estimate

Estimate how much time each deliverable takes to complete

- First time through, do a quick and rough estimate
- Put together rough schedule for all deliverables

Better Deliverables: Estimate

• Add AES encryption method (learning, arch. & coding)	21 days
• Add AES decryption method (learning, arch. & coding)	18 days
• Sequence item	
○ update `do_compare()` with encryption	5 days
• Program Keys	
○ New sequences	4 days
○ New virtual sequence (key + traffic)	1 day
• Update configuration object	½ day
• New test	½ day
• Coverage	
○ New DUT modes	1 day
○ Keys	1 day
• Total	52 days

Deliverables: Review & Refine

Review with stakeholders

- Will the schedule meet the requirements of the larger team?
- Will the team be able to meet the input requirements of the verification team?
- Do we need to reorder things to avoid downtime?

Refine

- Likely need to modify some deliverables based on feedback
- May need to divide some deliverables

Linear Progress

What do we mean by Linear Progress?

- Evenly sized deliverables
- Evenly spaced delivery dates
- Team works together on single deliverable
- Later deliverables build on earlier work

Coverage & Scheduling

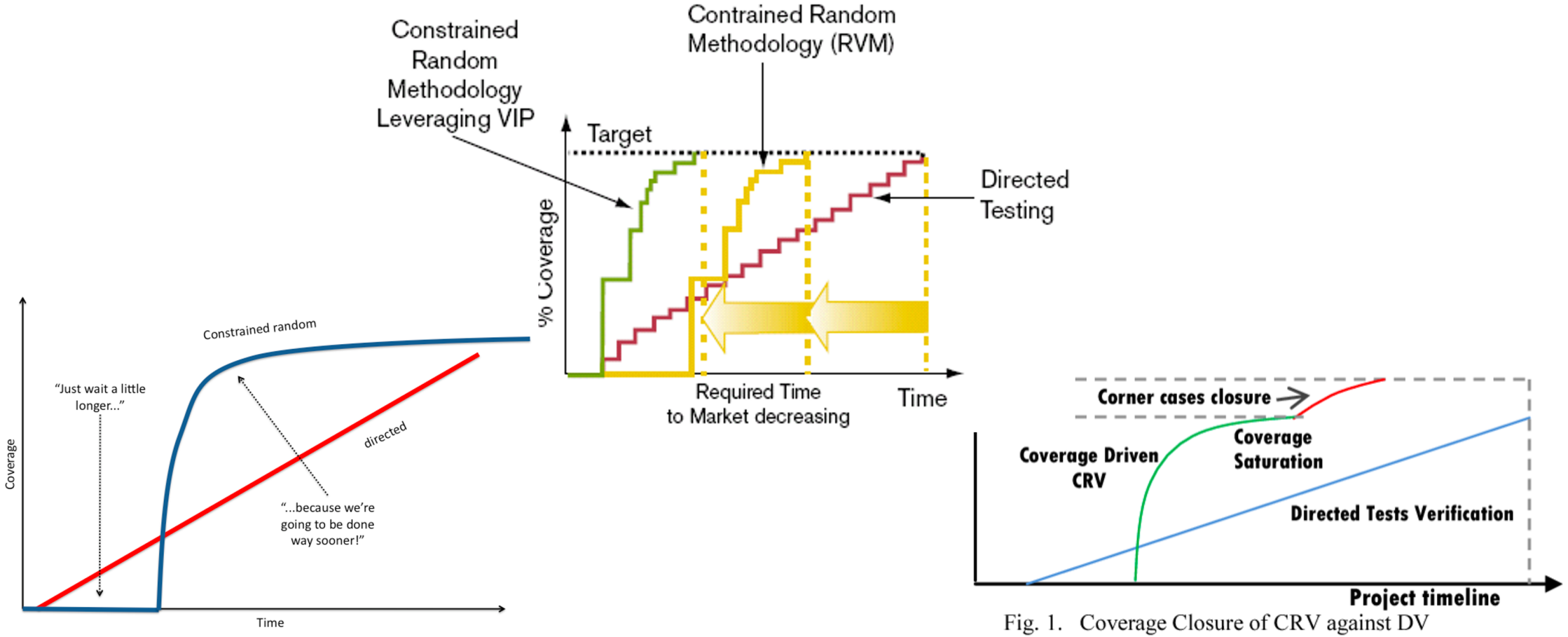
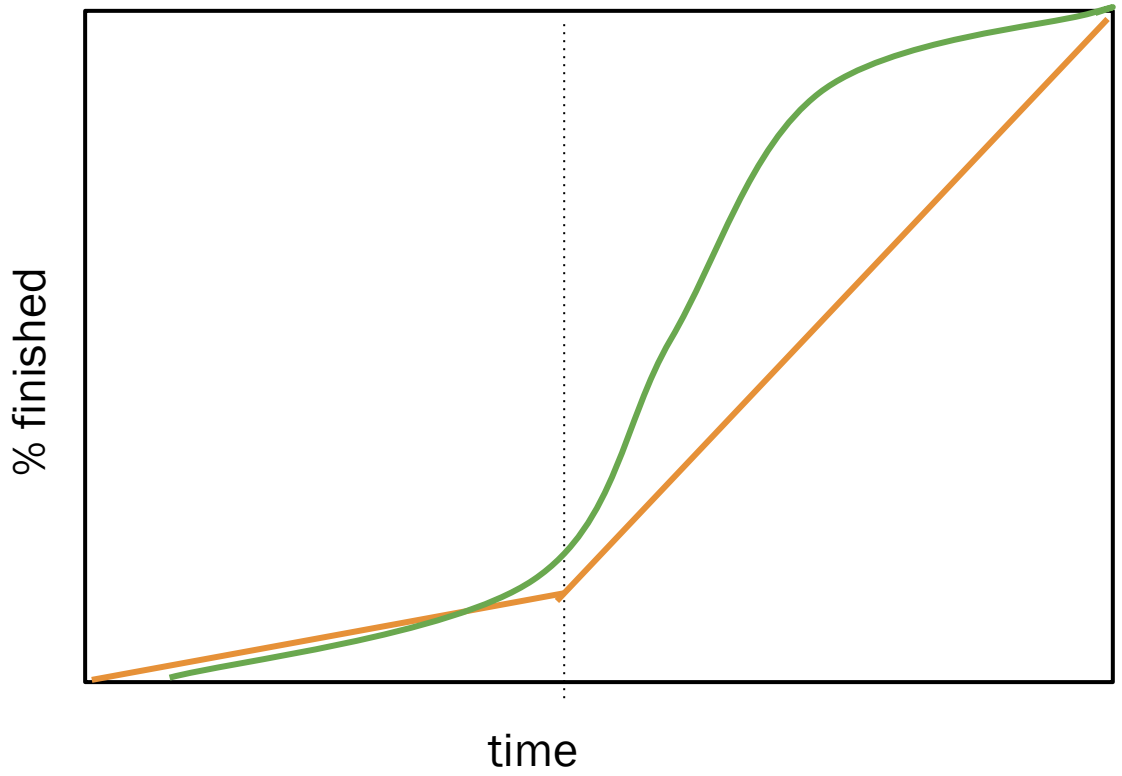


Fig. 1. Coverage Closure of CRV against DV

Linear Progress Planning

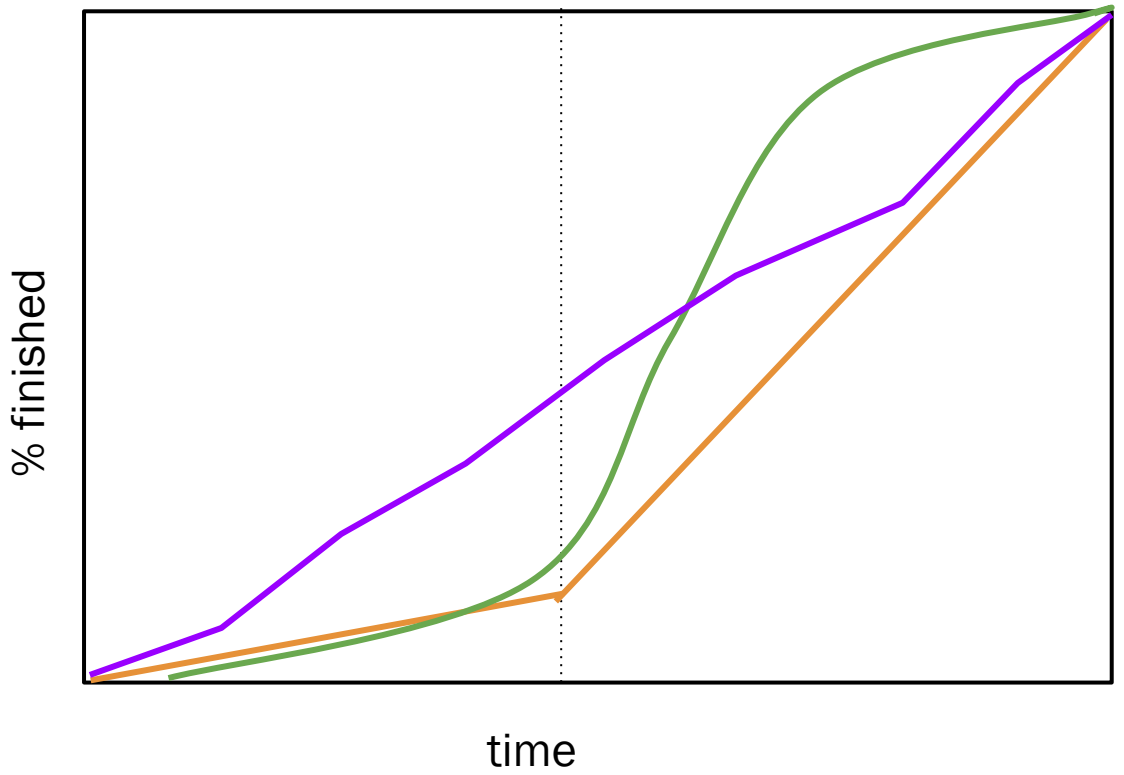
Hard for outsiders to see the difference between

- ideal coverage
- We'll suddenly get more productive next week



Linear Progress Planning & Status

- Clear communication
- Regular deliverables
- Predictable timeline
- Simple status tracking



Linear Progress Planning: How to

One deliverable at a time	• Clear beginning and ending
Consistent sizes	• May need to divide or combine some deliverables
Order to build on previous work	• Depth first v breadth first

A Note on Ordering

Depth first development

- Finish up major features one at a time
- Design team may focus on a few things first
- Some features may be completely coded or brought in as IP

Breadth first development

- Simple implementation first, then go back and add more features
- Many designers working in parallel

Status Updates

Simple to determine which deliverables are done	• Running estimate vs actual gives idea of ahead/behind
Simple to determine what isn't done	• Higher abstraction level reduces status reporting effort
Simple to know what is being developed	• Easy to determine if we're ahead or behind on current deliverable

Flexibility

We'll inevitably have changes

- Want to be able to modify schedule easily
- Communicate impact clearly

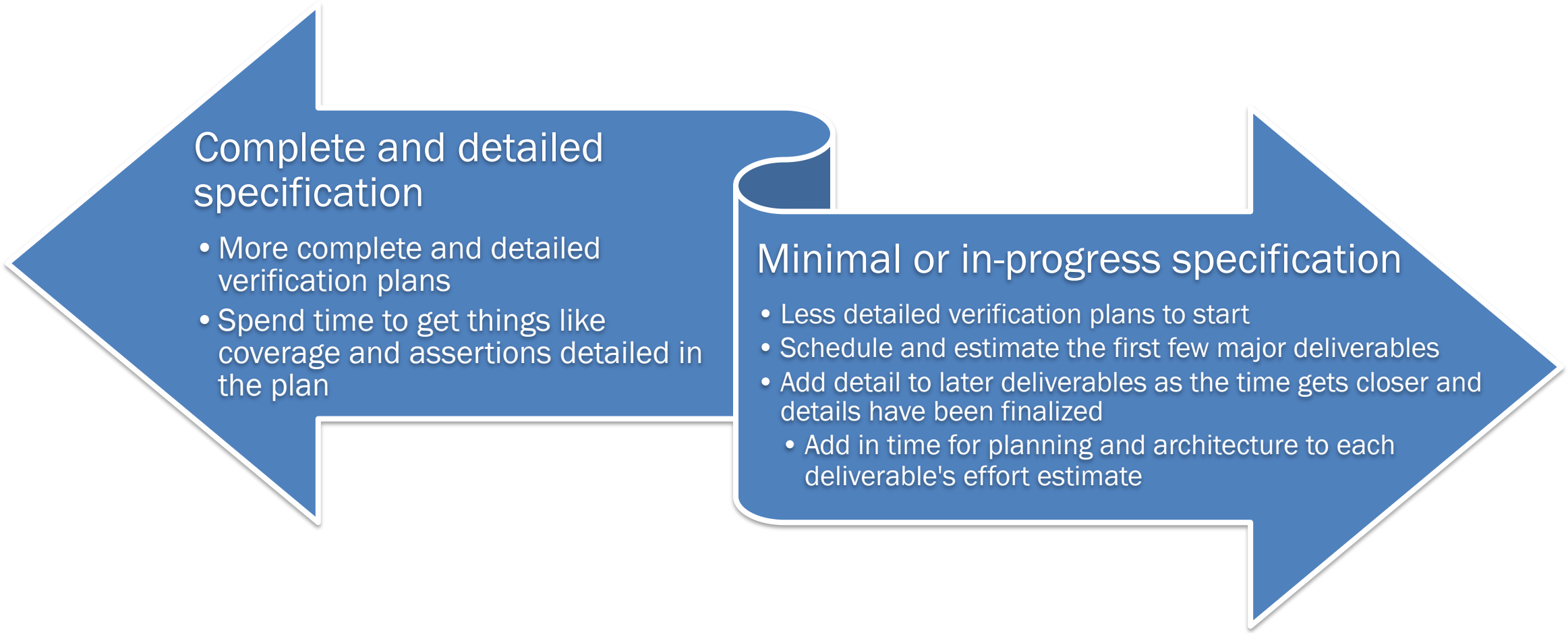
Refine or absorb small changes in future work

Add new deliverables for significant changes

- New features / modes
- Changes to finished work

Reviewing the changes to the plan with the team will communicate the impact in a way that will be easily understood

Detail levels



Does our process produce these results?

Low effort for fast results

- More detail easily added later if necessary
- Independent of larger team

Clear communication of verification status

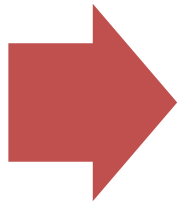
- Clear to all teams what is done and what is left to do

Flexibility

- Reacts well to changes
- Adapts to differing degrees of documentation completeness

Question and Answer Session

Verilab has deep experience in Verification Planning garnered through working with many clients over our 22 year history



Contact Jason Sprott (Jason@verilab.com) to schedule a consultant to help **you** create a Verification Plan for your project