CHANGE AGNOSTIC TESTBENCH

STRATEGY TO MANAGE UNFORESEEN REQUIREMENTS

ANSHUL GOEL

PRINCIPAL ENGINEER, NXP

AUTOMOTIVE GROUP

Agenda

- IP development lifecycle challenges – Unforeseen changes
- Case studies (with MIPI-CSI2 Protocol/TB).
- Strategies to manage the changes
  - VIP overdependency
    - Simple innovations reduced 100+ files.
  - Guidelines and key areas.
  - UNR flow for dead code.
- Result - IP Verification Life Cycle.
- Conclusion
- Q&A

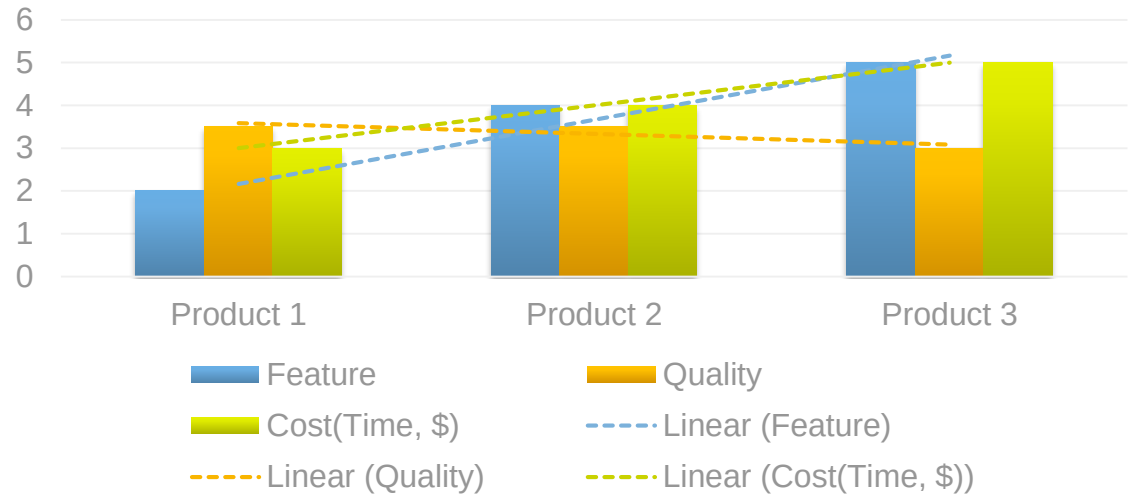
Typical IP Life Cycle

Introduction
- High Differentiation
- Innovation Key
- Targeted product
- High on investment

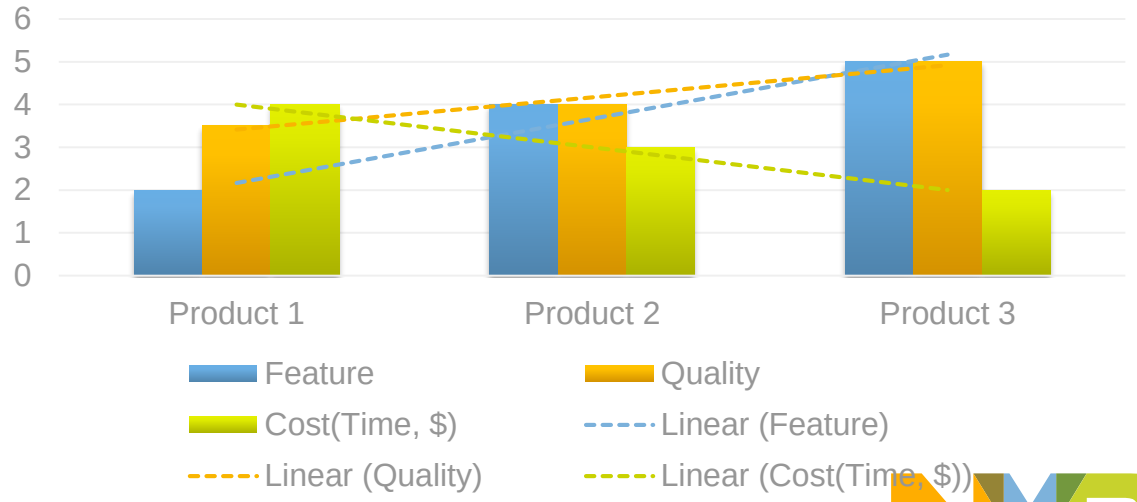
Growth
- New feature, New market
- High Quality
- Needs quickly

Maturity/Decline
- Price competition
- IP phase out

Unplanned Development Cycle



Planned Development Cycle



IP Development Lifecycle – Unforeseen Issues

Typical development cycle sees many changes, both planned and unplanned

Some type of changes / challenges are :-

- **Implementations driven** :
  - Protocol updates, new transactions,
  - programming model, ports changes,
  - architecture / instance changes,
  - Lastminute changes to spec.
  - Generic IP - Code coverage issues.
- **Project Management driven** :
  - EDA tools change, Collaterals change,
  - Late availability of input collaterals(RTL, models etc.) leading to decreasing schedule buffers,
  - Human Resource :
    - Proficiency mismatch,
    - Inadequate Bandwidth.

Case #1 - Implementation Driven Changes

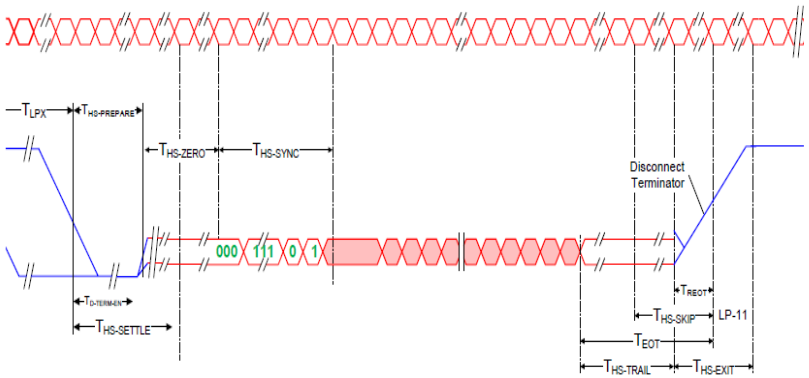
- **Feature creeps** : While in planned execution, you suddenly realize that there is more implementations needed to support a feature. For example…
  - MIPI-Dphy V1.2 support 2.5G link, added extra requirement for initial and period De-skew.

Q: Can we run existing test suite with no change or change only in environment ?

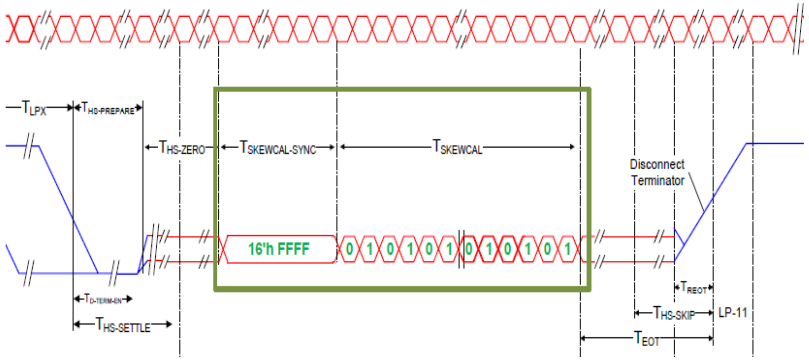
- **Supporting multiple configurations**:
  - Example : MIPI used in radar and vision applications may support different features like DT, VC, interleaving etc.

Q: Can we make this testcase independent ? Control using command line, inline constraint based on product ?

High-Speed Data Transmission in Normal Mode



High-Speed Skew Calibration



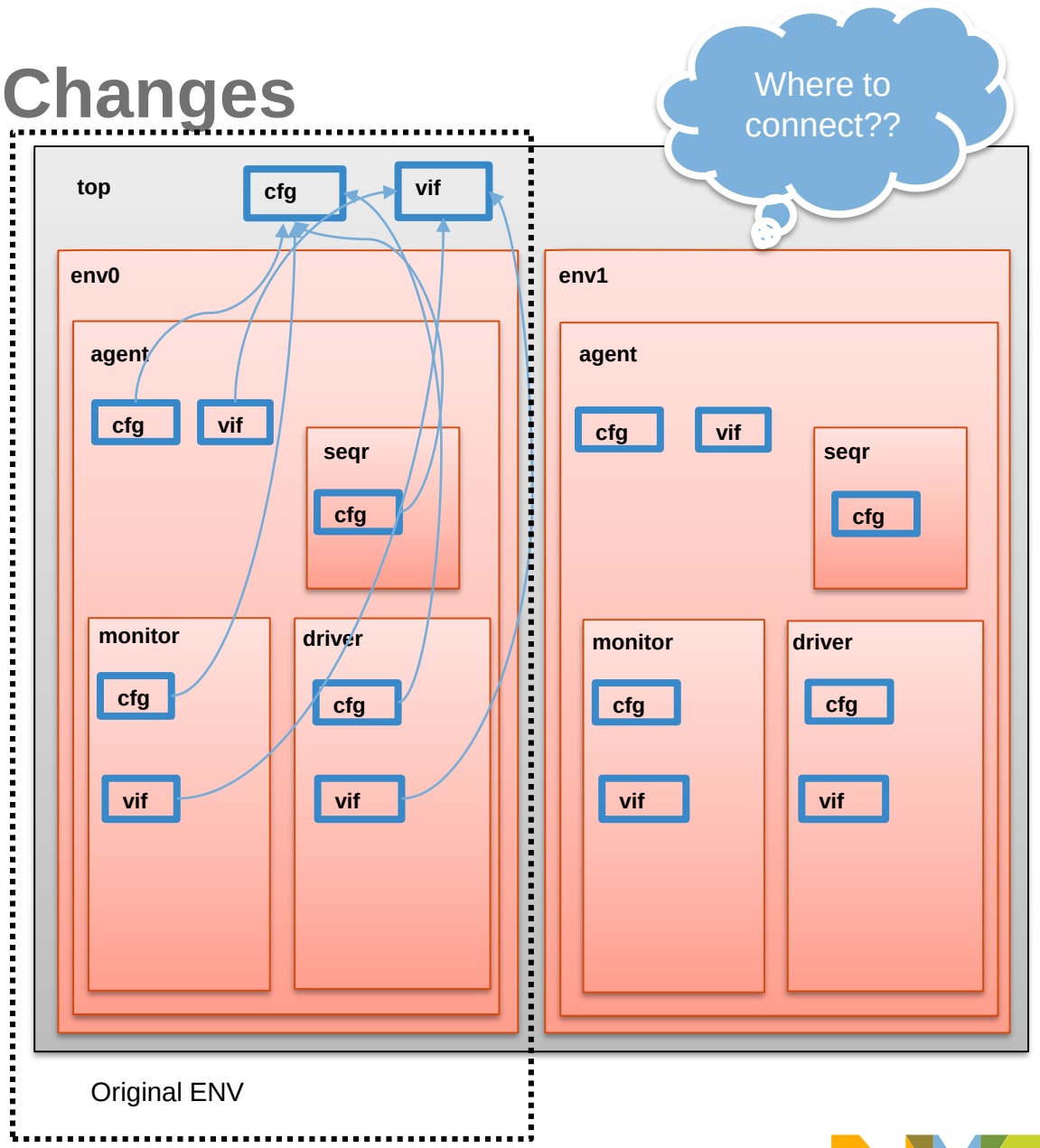
Case #2 - Implementation Driven Changes

Sales – *our competitor has X port solution! we need more!!*

Architect - *hmmm… solution add one more MIPI instance in SoC*.

Designer - *no problem, only changes in glue.*

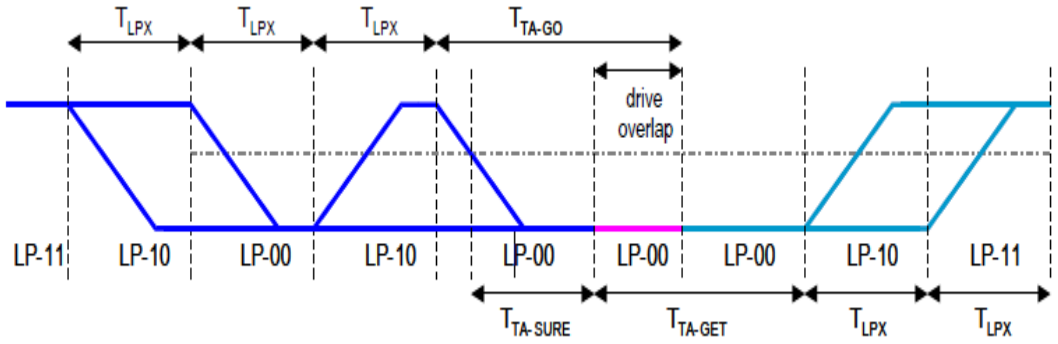
Verification - *two instance!!.. Ok but now I need two env0/1 handle, and how do I **get** cfg, vif ?*



Case #3 – Planning issues

- Last-minute new market requirements :
  - Support Bus turnaround feature?

Q: can we run existing test cases with bus turn around enabled?



- Configuration/register definition:
  - Specification not available, Register name/address changed.

Q: Can we start building environment (ref model)?

Q: Changes needed in all test cases?

Case #4 – Project Management Driven Changes

- EDA tools / IT infrastructure changes :
  - For example -Tools transition overlaps with project execution window.
  - Transitions unavoidable say due to cost efficiency or better comparative tool.
  - Organizational direction changes

Q: Can we change VIP, tools, reduce resource and signoff X month early? As you know time to market is the key!

VIP Overdependency!

- Working with VIP configuration class across TB components and testcase development helps in fast development, but in long run creates high dependency.
- High user intervention in designing sequences and executing testcases.

  Sequence flow:

  //randomize dt, vc, mode

  //uvm_do_with(FS)

  //uvm_do_with(HS)

  …..

  //uvm_do_with(FE)

  …..
- Sequences are hard coded to sent specific patterns – like VC/DT event, interleaving.
- Randomization are inbuilt in sequence instead in global configuration.
- Sequences need lot of effort to adhere to protocol enchantments. Like >1.5G we need to insert initial/periodic skew transaction in every sequence.
- Good understanding of VIP and methodology like UVM is a must for someone writing the test sequences.

Simple innovation reduced 100+ files

Pseudocode :

Queue :

```
struct {
        virtual_channel_id
        data_type
        transmission_mode
        number_of_pixel
        seq_len
        bta_enable
    } queue;
```

Base Sequence:

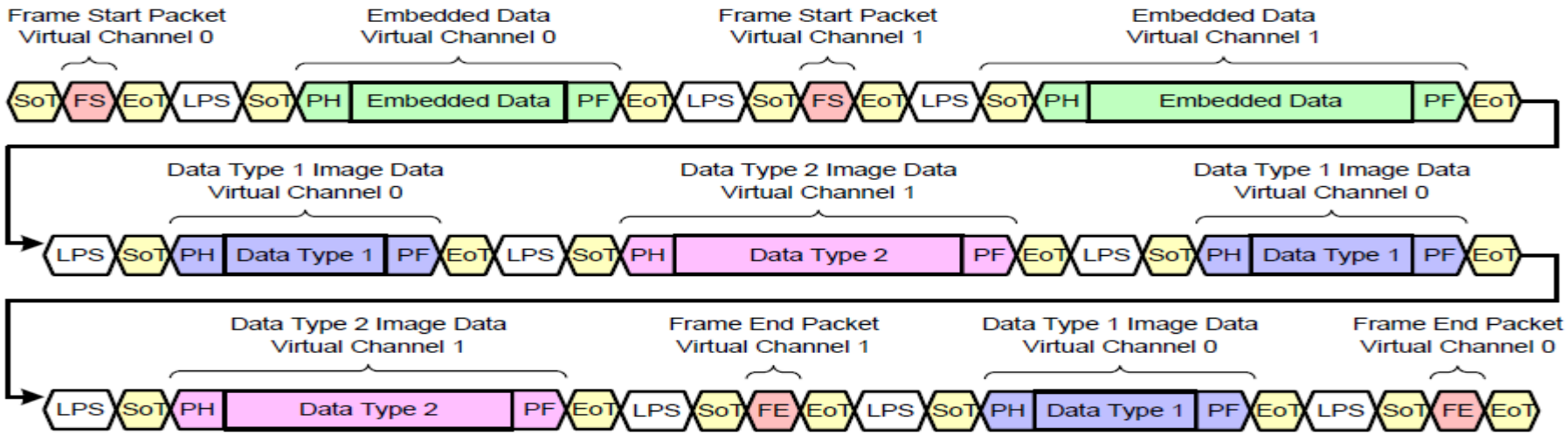
```
foreach(queue[idx]) begin
  repeat(queue[idx].seq_len) begin
    `uvm_do_with(req,{
      req.vc_id                == queue[idx].virtual_channel_id;
      req.data_type            == queue[idx].data_type;
      req.transmission_mode    == queue[idx].transmission_mode;
      req.number_of_pixels     == queue[idx].number_of_pixel;
      req.bta_enable           == queue[idx].bta_enable
    });

    get_response(rsp);
  end
```

- User can fill queue randomly or directed.
  - Number VC
  - Data type
  - VC interleaving/no-interleaving
  - DT interleaving/no-interleaving
  - And many more..
- Single sequence used for all datapath feature test.
- Achieve ~80% of functionality by providing knobs to control data queue filling.
- Single test with run time arguments can replace which was getting covered by 100+ classes/files (test+sequences).
- Protocol changes can be addressed by manipulating queue. E.g. filling queue based on speed, just add de-skew pattern in top of queue.
- Features like bus turnarounds require changes only at sequence controlling VIP

Constraint Randomization - CH[].FLOW[] concept

```
if (ucp.get_arg_value($psprintf("+set_dt_%0d_%0d=", ch, flow), value)) begin
  cmd_line_ovrrd.ch[ch].fl[flow].dt = cmd_line_ovrrd.ch[ch].fl[flow].dt.first();
  for(int ii=0; ii<cmd_line_ovrrd.ch[ch].fl[flow].dt.num(); ii++) begin
    if(value == cmd_line_ovrrd.ch[ch].fl[flow].dt.name()) break;
    cmd_line_ovrrd.ch[ch].fl[flow].dt = cmd_line_ovrrd.ch[ch].fl[flow].dt.next();
  end
  cmd_line_ovrrd.ch[ch].fl[flow].dt_vld = 1;
end
else if (ucp.get_arg_value($psprintf("+set_dt_%0d_all=", ch), value)) begin
  ----
  ----
end
```



Managing Changes – Guidelines and Key Areas

- Don’t work with VIP configuration/knobs in your TB.
  - Create own configuration class with all required feature, protocol etc.
  - Create API’s to map custom configuration to VIP configuration.

- Don’t use RAL model classes/fields or hard coded address directly in your TB.
  - Inside TB components work with custom TB configurations
  - Create API to map TB configuration to RTL configuration.

- It’s always good idea to create TB randomizer manageable through command line arguments.
  - Help in creating fast directed cases, achieving quick functional and can be left full random.
    - cfg_cmd_line_ovrride();
    - cfg_inline_constraints();
    - map_to_vip();
    - map_to_rtl_cfg();

Managing Changes – Guidelines and Key Areas

- Configuration object or virtual interface should set in top-down fashion, this helps in creating reusable verification environment.
  - Each component in hierarchy responsible for passing the cfg/vif to its sub-component rather than getting or passing directly from top-level.
  - In case of multiple instance can we associate core/instance number into interface name?

```
uvm_config_db#(XMTR_IF)::set(uvm_root::get(), `"*env_``CN`","xmtr_if", xmtr_if_``CN);
uvm_config_db#(XMTR_IF)::get(this,"",psprintf("xmtr_vif_%s",core_number), xmtr_vif)
```

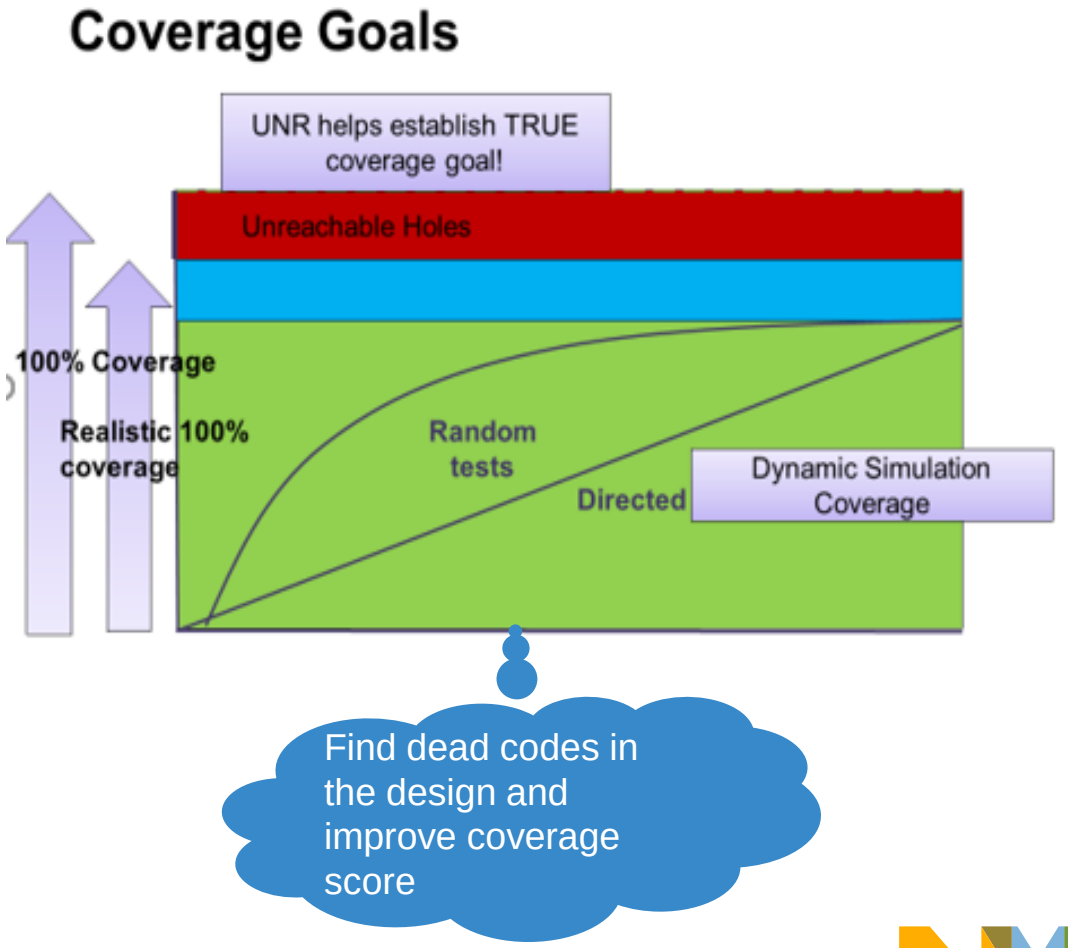
- Interface binding having capability to connect according to instance number. Below example show macro based implementation to connect any number of instances.

```
`define MIPI_DPHY_IF(CN)     \
   assign `rtl_top.mipi_``CN.cp = xmtr_if_``CN.serial_if.serial_tx_clk_if.dn;     \
```

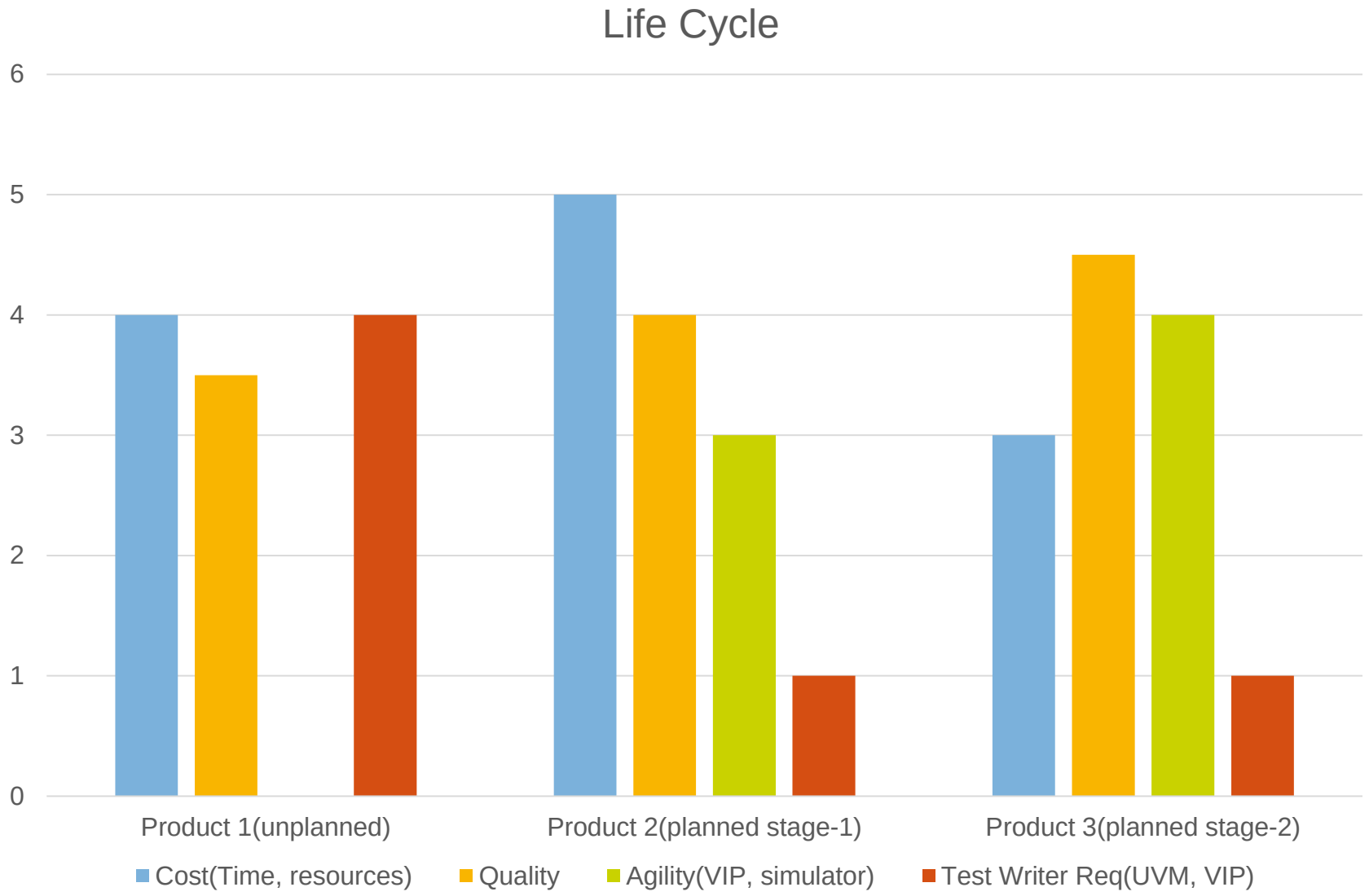
- Verification testbench linting tools can help to enforce rule sets customized for these guidelines.

Generic IP – Dead Code ? Unreachability analysis

- Simulation has hit a plateau and reached 80-90% coverage
  - Not possible to close the remaining gap using directed tests alone
  - Manual review of uncovered points is tedious and time consuming. Can take several weeks.
- Import simulation coverage database and target uncovered points using formal
  - Automated push button technology.
  - No test bench is required. Saves time & effort writing tests
  - Use formal algorithms to find dead codes in the design
  - Waive unreachable points to get improved coverage score



Result - IP Verification Life Cycle



Conclusion

- IP life cycle has various stages with its own challenges.
- Important to plan introduction phase keeping in mind longer term rewards, right question and unforeseen issues. Design Verification closure is key to many issues – agility, quality and time to market.
- Try to avoid VIP dependency as much as possible in testbench infrastructure.
- Simplification of testcases development help in accelerating bug hunting process.
- UNR can help in faster code coverage closure and avoid manual review of uncovered points, as it’s tedious and time consuming. Can take several weeks.
- Testbench linting tools can help to enforce rule sets customized for these guidelines.

Q&A

SECURE CONNECTIONS
FOR A SMARTER WORLD

Backup

Example Testbench