How to Succeed Against Increasing Pressure: Automated Techniques for Unburdening Verification Engineers

James Pascoe (STMicroelectronics)

Steve Hobbs (Cadence)

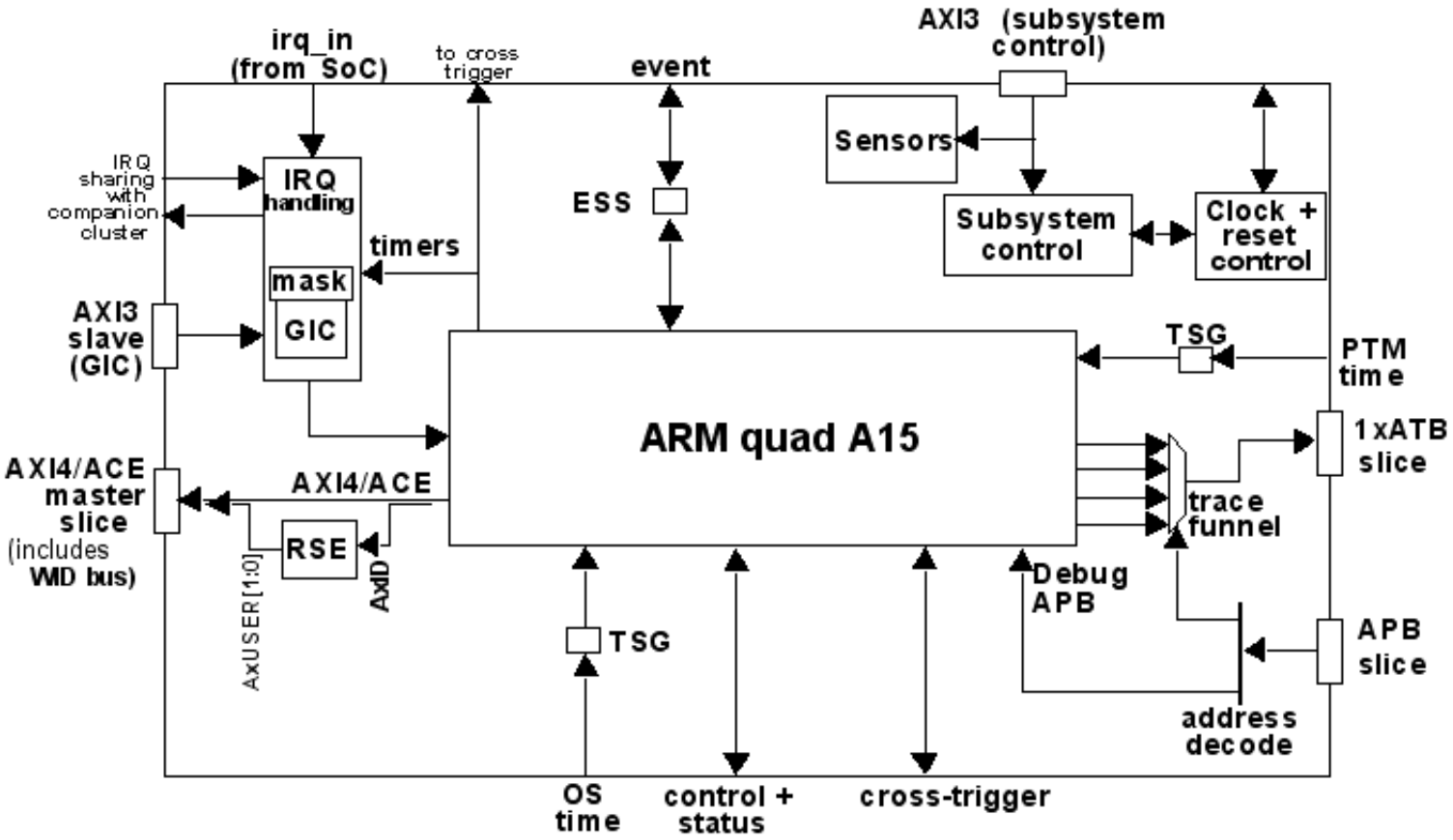
26 February 2013

life.augmented

Overview

- Who are we:
  - The ‘CPU’ part of ‘CPU/GPU’ in TR&D (ST Bristol)
  - Steve is the local Cadence FAE ☺
  - We develop ARM based sub-systems for a range of SoCs
- Organisation:
  - System-level functional verification (Noida)
  - Block-level activities (Bristol)
  - Low-power and DFT verification (Grenoble)
- Automation techniques:
  - Release Management System (RMS)
  - Gatekeeper flow
  - Reachability flow

Subsystem



ESS = Event signal synchronizer

TSG = Time-stamp generator

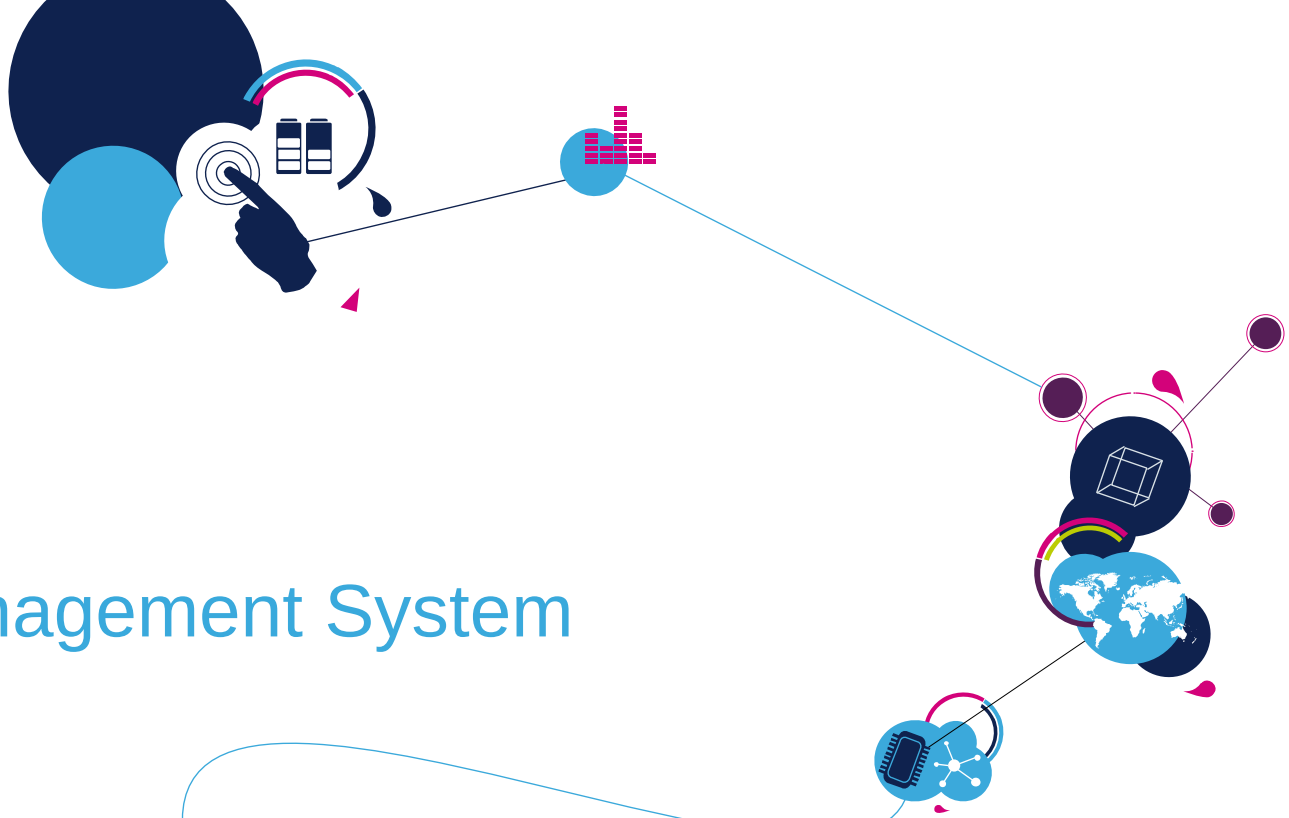
RSE = Request source encoder

Context

- The scope of verification is increasing:
  - Verification engineers in Bristol act as integrators for the rest of the team
    - Merging developer commits onto the main-line
    - Lots of time spent debugging faulty commits
    - Verification engineers were spending > 35% on non-verification activities!
  - Grenoble team provides a customer interface
    - Developing IP-XACT descriptions of components
    - Providing integration support to SoC teams
    - Fielding questions on technologies not related to verification
  - Noida team is frequently required to produce commodity data
    - Coverage reports, qualification runs, regression data …
    - Requests are ad-hoc and disrupt the day-to-day workflow
    - High potential for automation

Approach

- Developed a Release Management Server (RMS)
  - Extensible infrastructure for driving bespoke flows
  - Automated merging of developer commits
  - Commodity data on request
- Gatekeeper flow
  - Developers do not want to run full regressions for each modification
  - Smoke tests do not always exercise the actual modification
  - Gatekeeper flow provides a meaningful list of tests to run before a commit
- Unreachability analysis
  - Not all states are reachable in functional mode
  - Unreachability analysis excludes unreachable states from coverage data
  - Indicates areas of dead code and increases accuracy in coverage data



Release Management System

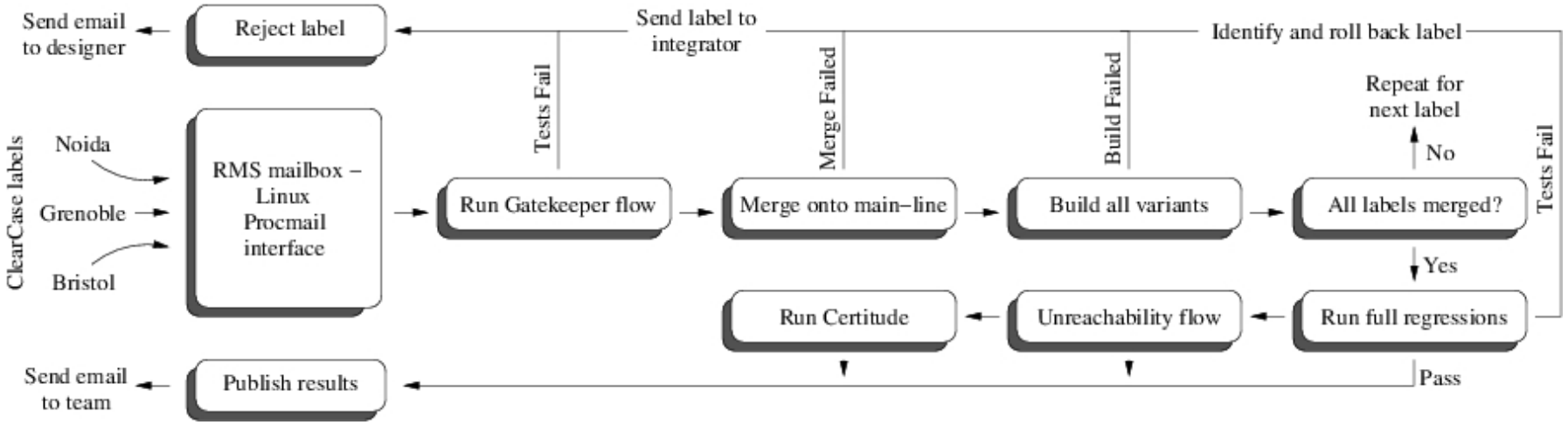
life.augmented

The Problem

- Bristol team are required to produce global releases
  - Designers email ClearCase labels to integrators (verification engineers)
  - Tight timescales lead to poorly tested labels
  - Integrators merge labels into ClearCase and run regressions
  - Verification engineers spend lots of time debugging commits
- Noida team is increasingly asked for commodity data
  - Requests for coverage reports, qualification runs and regression data
  - Frequency of requests has increased during the project
  - Highly disruptive to day-to-day work flow
- Observations
  - At peak times the number of labels and requests is high
  - Majority of labels and requests can be fulfilled with no human involvement
  - Over time, code quality remained flat

Release Management Server

- Similar principle to a Continuous Integration server (e.g. Jenkins)
  - Developers email commands to the RMS:
    - `MERGE GNB_EAGLE_SS_V14.15.3_pascoej_incisiv_update`
    - `COVERAGE –unreachability YES`
    - `CERTITUDE –run NOW`
  - Server executes following algorithm:



Release Management Server

- Implementation details
  - Built around widely available tools (Fedora Core 17)
  - Procmail polls a well defined mailbox
  - Perl scripts perform actions
  - RMS integrates with existing project build infrastructure
  - Provides a platform for running custom flows
- Results
  - Developers were more willing to fix problems when given counter examples
  - Encouraged frequent smaller merges rather than big monolithic merges
  - On-demand access to commodity data unburdened the Noida team
  - Engineers enjoyed building the RMS
  - Allowed three key verification engineers to work on verification ☺

The Gatekeeper Flow

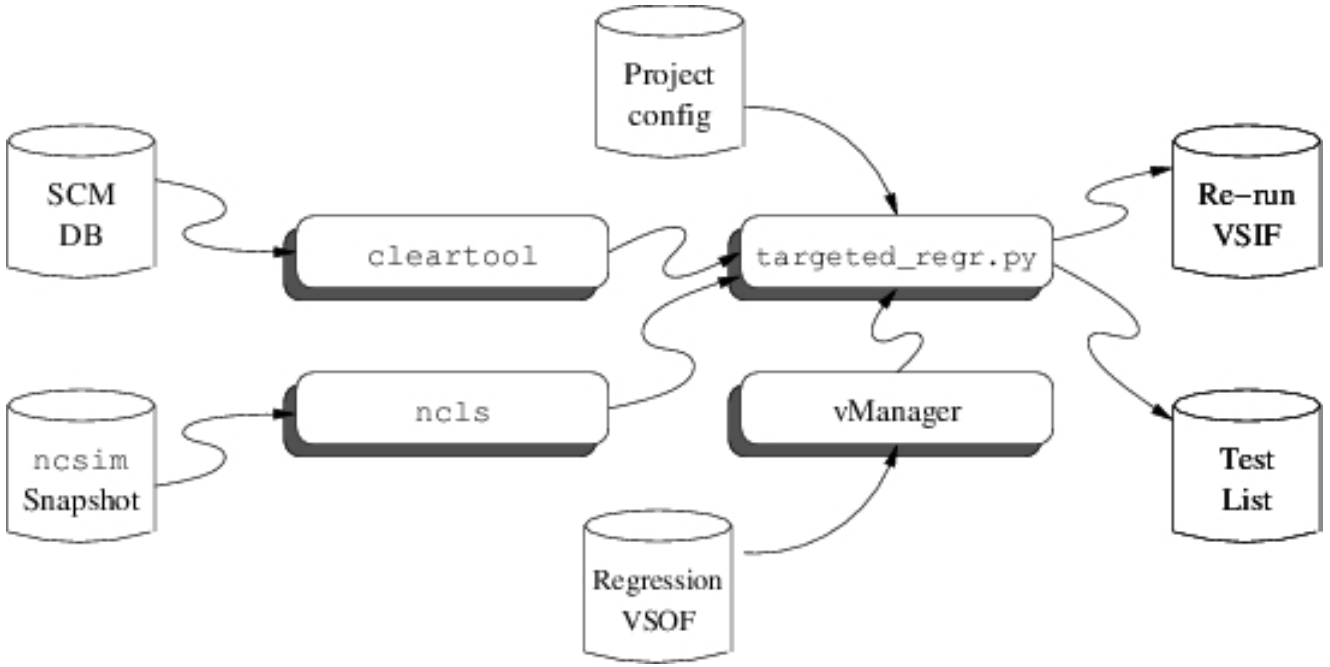
life.augmented

Gatekeeping

- How can we improve the quality of labels?
  - Poorly tested developer commits waste verification effort. However …
  - Running full regressions on each label takes too long
  - Smoke tests do not always exercise the modifications
- Gatekeeper flow
  - Provides a set of smoke tests that are meaningful for each label
  - Uses ClearCase to determine which modules have been affected
  - Analyses simulation snapshot to determine which tests are suitable
- Observations
  - Developers like and are happy to use the flow
  - Test failures are detected more quickly and are more relevant
  - Python script is included in the paper

Example Deployment

- Flow is designed to be portable:
  - Script leverages helper classes to abstract away from specific toolsets etc.
  - Project specific details are contained in a project config file
  - Current deployment at ST is as follows:



Reachability Analysis

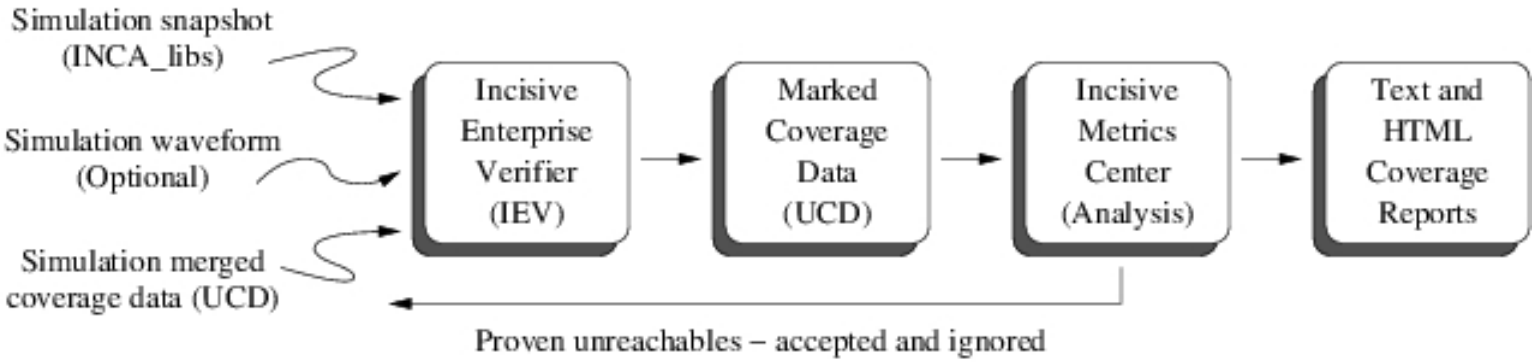
life.augmented

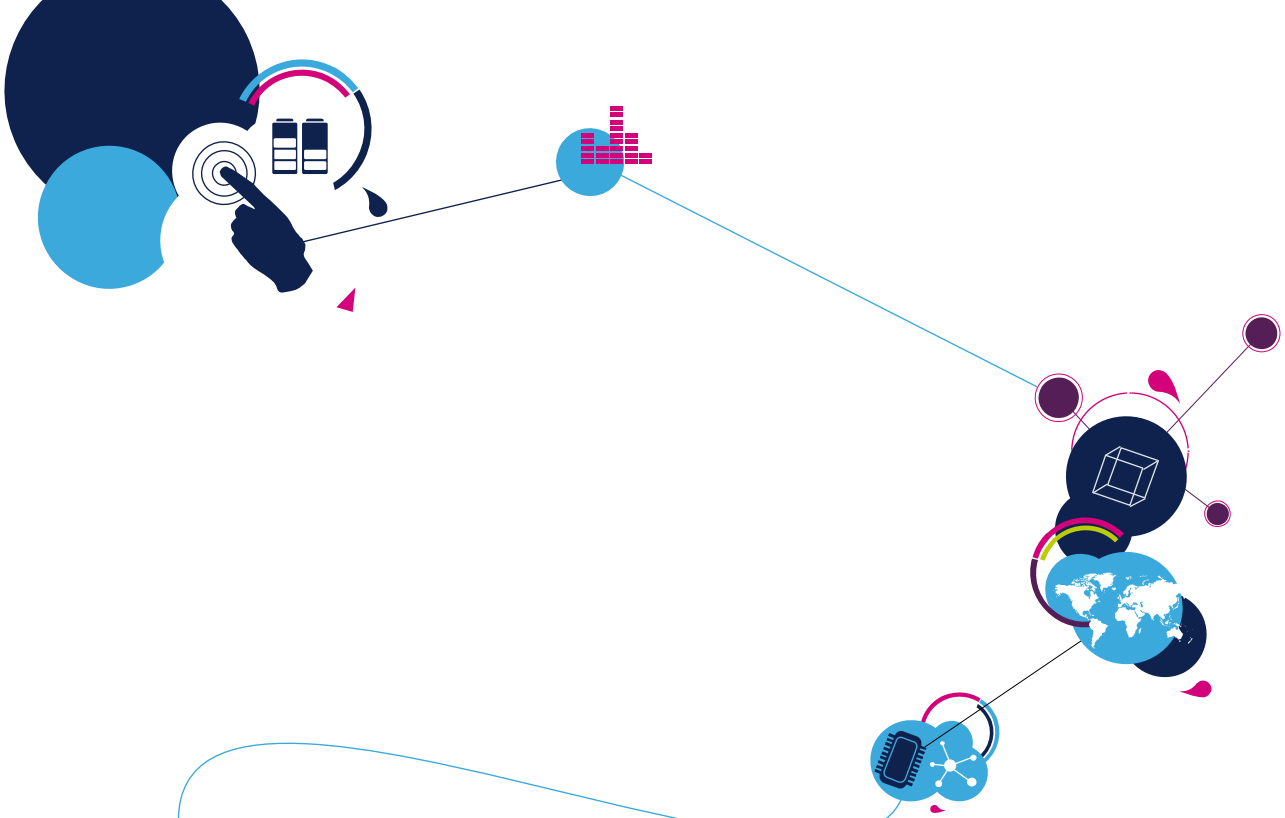
Reachability Flow

- Coverage data is useful to monitor progress
  - However, not all states are reachable through functional testing (e.g. DFT)
  - The RMS includes an automated flow to exclude these states
  - Useful for highlighting areas of `dead-code'
- Reachability flow
  - Implemented using formal tools (i.e. `formalverifier`)
  - Uncovered items in the coverage database are translated into 'cover' assertions
  - Assertions are 'proved' by `formalverifier`
  - Generates a list of coverage marks
  - Marks are passed to coverage tool which excludes unreachable states
- Observations
  - ARM IP contains almost no dead code
  - Reachability flow meant that redundant code did not accumulate

Operation

- Reachability flow:
  - Based on the Cadence tool-chain
  - Uncovered items in the coverage database are translated into ‘cover’ assertions
  - Assertions are ‘proved’ by `formalverifier`
  - Results in a list of coverage marks
  - Marks are passed to a coverage tool
  - Coverage tool excludes unreachable states from published results
  - Script is in the paper



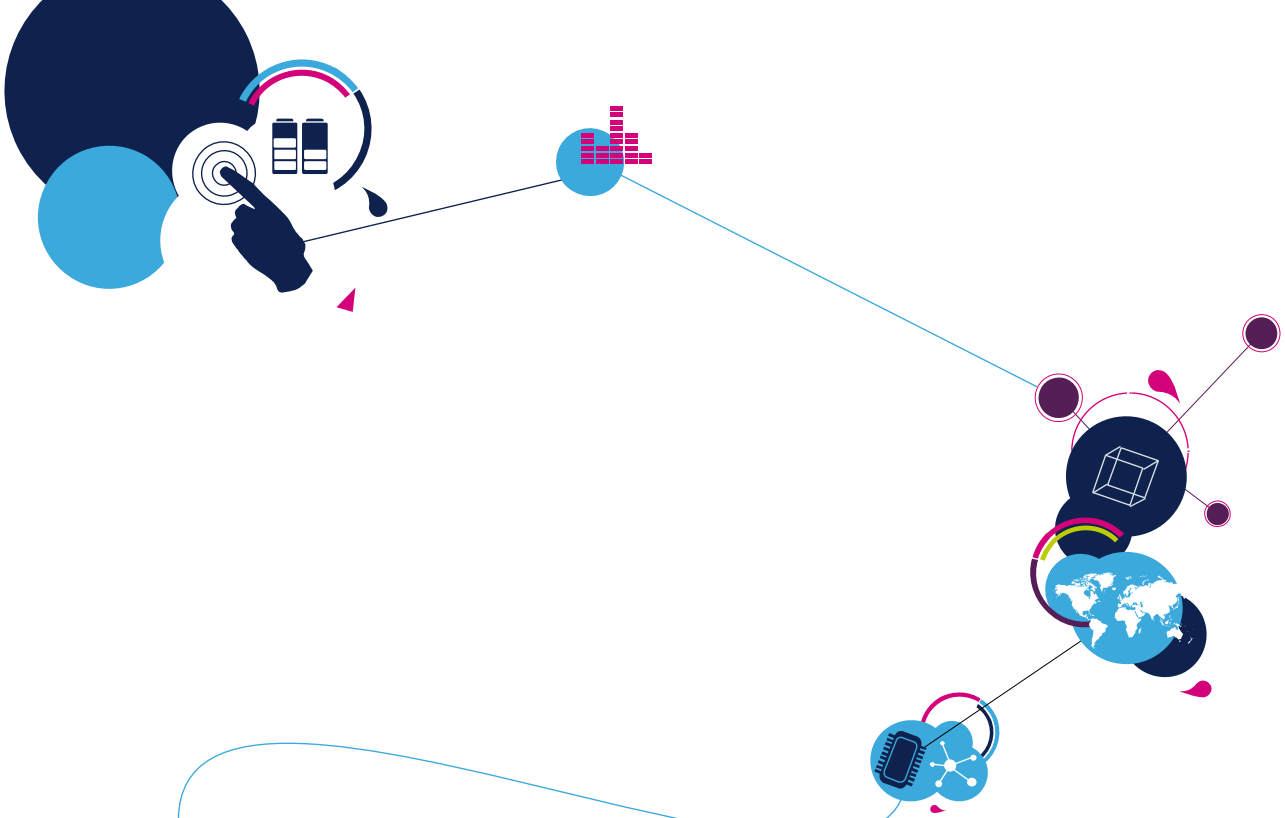


Conclusions

life.augmented

Conclusions

- Overall, the project has been successful
  - Allowed three verification engineers to focus on verification
  - Gatekeeper flow has improved code quality
  - Reachability analysis has improved accuracy and eliminated dead-code
  - Commodity data now available on request
- Interesting cultural benefits
  - Engineers enjoyed developing the automated solutions
    - More interesting than `handle turning'
  - Designers will address bugs in labels when provided with counter examples
    - RMS provides good feedback
  - Encourages better working patterns
    - Engineers are making more frequent smaller commits rather than big merges
  - Provided a sense of engineering the way that we work
    - Not just what we deliver! ☺



Questions

life.augmented