Presentation to Verification Futures Europe 2017

T&V S

Agile Software Testing

Test and Verification Solutions

Delivering Tailored Solutions for
Hardware Verification and Software Testing

T&V S
Test and Verification Solutions

‘agile’ or ‘AGILE’?

- **agile is a philosophy not a process**
  - It can apply to almost any business
    - not just software development
  - It should change the structure of the company
    - especially the roles of staff and management
  - Continuous improvement
    - OK to make mistakes if you learn from them

- **There are lots of AGILE software methodologies!**
  - Scrum, Kanban, Lean Startup, …
  - They all need:
    - An agile company culture
    - An agile DevOps organisation for projects
    - Support for continuous improvement (PULL not PUSH)

Agile Software Development

- **Need clarity of requirements & priorities**
  - A well defined customer who will be accepting deliverables
- **Implement from a task list [of features] in priority order**
  - You can still commit to product releases at fixed times → **BUT DON'T OVER-COMMIT**
- **Developers 'own' the code they write (Need accountability!)**
  - 'Done' means [regression/release/user] tested
- **Continuous Integration (CI)**
  - Always 'green'/deliverable [after each sprint]
- **Manage technical debt [especially testing debt]**
- **Agile software development IS scalable**
  - There are yet more methodologies: SAFe, DAD, LeSS, …
  - Avoid formal handoffs and synchronised releases

    eg: Use feature toggles and decoupled architecture

Continuous Integration and Shift Left Testing

- **Developer testing**
  - Typically Unit Testing & API testing (Can combine with TDD & BDD)
  - Can improve testability of legacy code by refactoring or finding ‘seams’ & APIs
- **Continuous Integration and ‘shift left’ for testing**
  - Test runs must be painless for the developer = easy, adequate, fast
  - Use a hierarchy: Pre-commit testing → Regression Testing → Release Testing
- **Automate all repetitive testing**
  - Need a strong investment in infrastructure
  - DevOps: but diversity is the rule (no single ‘best practice’)
    → solutions should be PULLED by Dev, NOT PUSHED by Ops
    …… *except bug tracking?*
  - Only keep what works (and this keeps changing) = a ‘waste repellent culture’

Integration Testing and System Verification

- **Typically have a separate Integration Testing Team (ITT)**
  - To overcome the complexity of system integration
  - To test performance and QoS

    … make sure not regressing
  - The ITT may serve multiple development teams

    …. In which case their iterations may need to be offset

- **Sometimes MUST have some independent testing**
  - Regulatory compliance

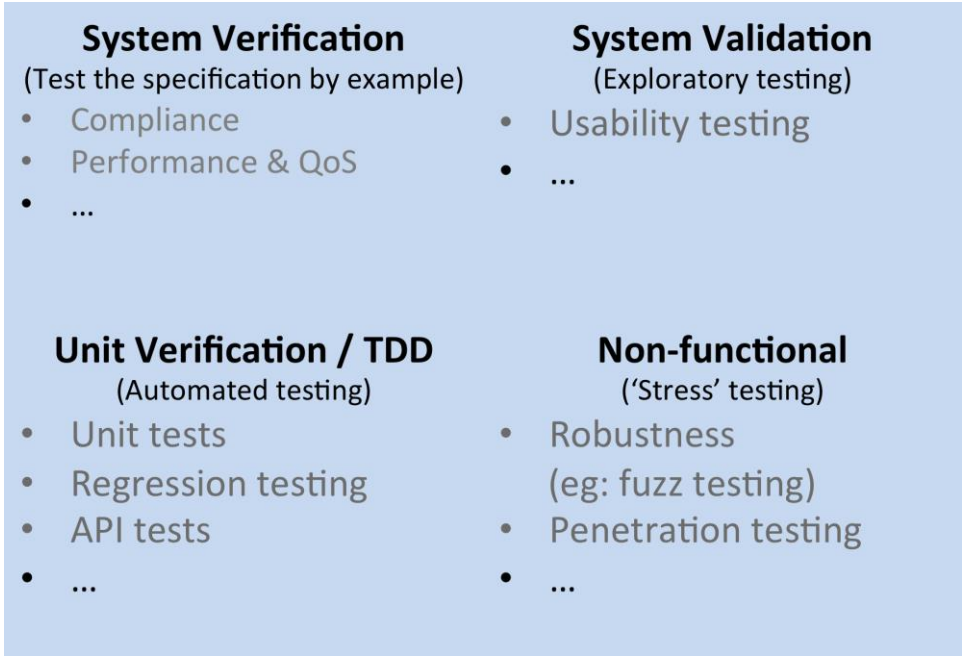
Exploratory Testing

- **Validation**
  - Is the product fit for purpose? → Try diverse use cases
  - User ‘stories’ should include acceptance criteria
- **End user testing**
  - Beta programs or ‘canary release process’
  - Feature toggles allow dark launching without planning
- **Fuzz testing to achieve robustness**
  - Want to catch illegal use (eg: warn end user)
  - Provide invalid, unexpected, or random data as inputs
  - Monitor program for exceptions
    → crashes, failing assertions, memory leaks …
- **Testing the ‘-ilities’ (security, reliability …)**

Testing Quadrants

BUSINESS

CHECKING EXPECTED RESULTS



ANALYSING OUTCOMES

- Undefined
- Unknown
- Unexpected

TECHNOLOGY

Test Selection

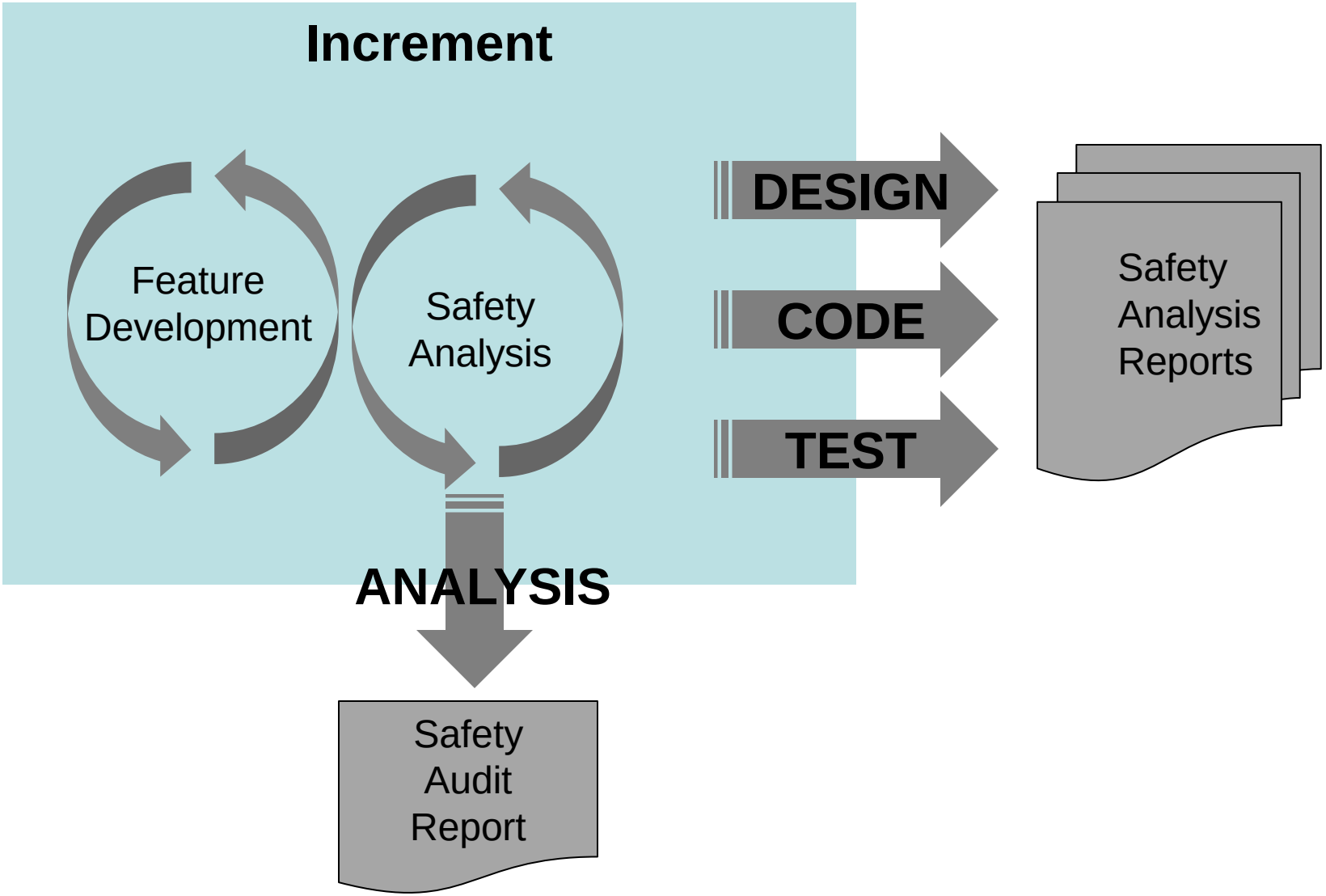
- **Good test selection essential to Continuous Integration**
- **Unit tests committed with patches or identified with good naming conventions**
- **Can use build dependencies to subset the regression tests**
- **Can collect data on tests to help prioritize test selection**
  eg: run time?, last failed?, testing bug fix?, …
- **Triage failing legacy tests:**
  don't be afraid to throw away tests!

  … but can use poorly understood legacy tests to ensure no regression in test passes

The Role of Metrics

- **Metrics need a clear purpose**
  - **Progress**
    - Number of features ‘done’
  - **Bug rates**
    - Defect cycle time (ensure issues are not being allowed to accumulate)
    - Number of bugs that regress
    - Percentage of [critical] bugs found by customers
  - **Identify ‘risky’ code and potential technical debt**
    - Code churn (visualised by heat maps)
    - Code complexity eg: cyclomatic complexity
    - Implicit architectural dependencies
      … can look at change waves to identify interconnected software modules
    - Test coverage to warn of holes in testing (line/statement coverage, path coverage …)
- **Who benefits?**
  - Measure OUTCOME rather than OUTPUT
  - Reporting should focus on TRENDS rather than NUMBERS
  - The generation of metrics should be automated + visual dashboard

Incremental Safety Analysis



Conclusion

- **‘agile’ v ‘AGILE’**
- **Agile software development**
- **Continuous Integration and Shift Left Testing**
- **Integration Testing and System Verification**
- **Exploratory Testing and System Validation**
  - Stress Testing: Fuzz Testing and ‘-ility’ Testing (*eg: reliability*)
- **Agile Testing Quadrants**
- **Test Selection**
- **Metrics**
- **You can do [incremental] Safety Analysis**