VOLANT

HISC

HIGH INTEGRITY SOFTWARE CONFERENCE

OCT 17, 2023

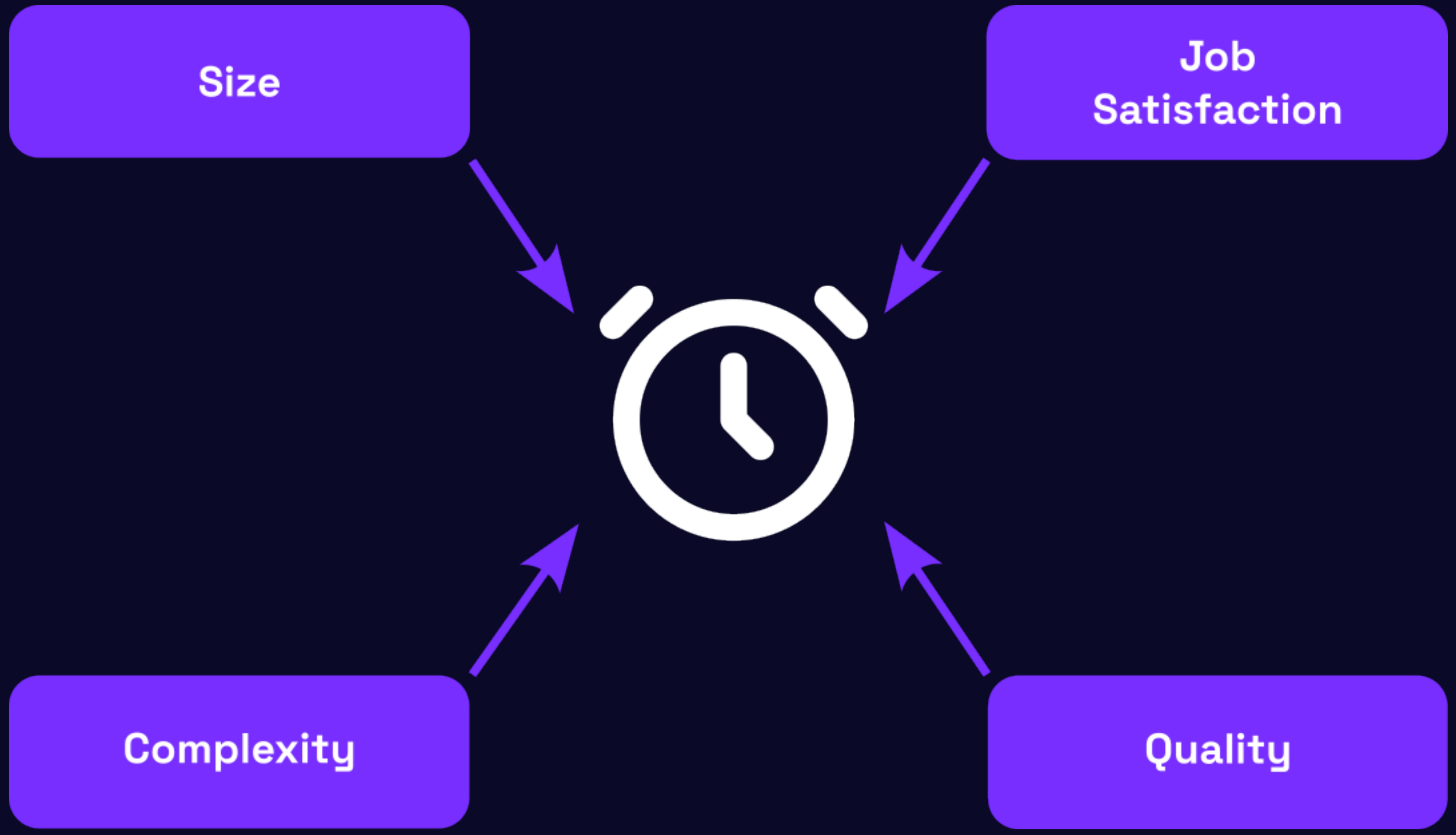
Optimizing the human: Safety critical on a budget

- Why do we have to optimise the human now?
- How do we achieve rapid certification?
- What is the human's role in future work?

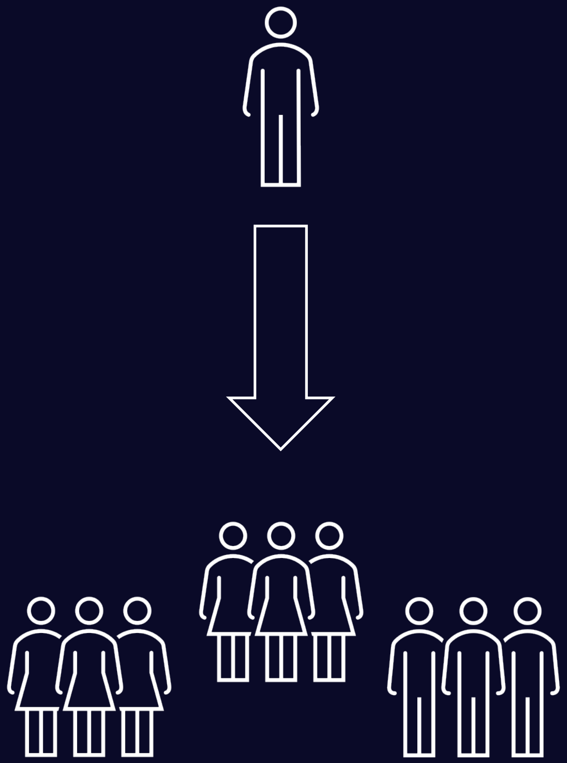
What is coming to our industry?

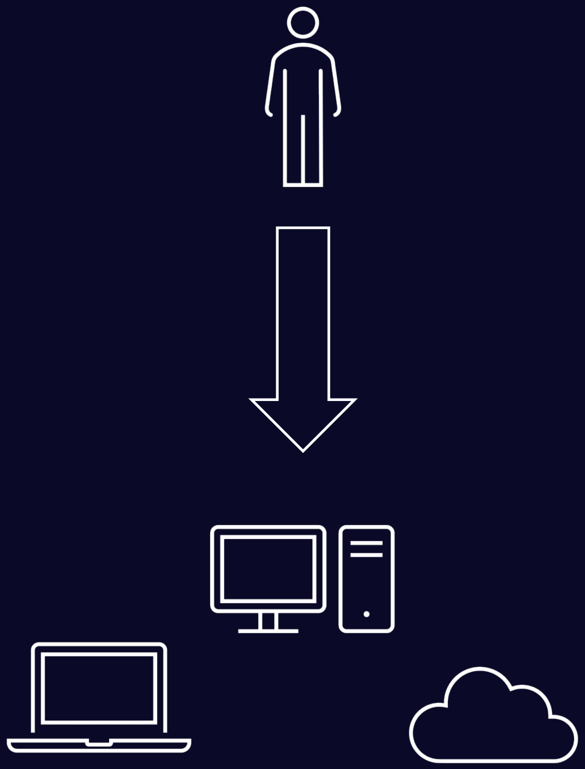
- Rapid expansion of platform capability
- Added complexity of interconnected systems
- Increasing demand

What drives timelines?

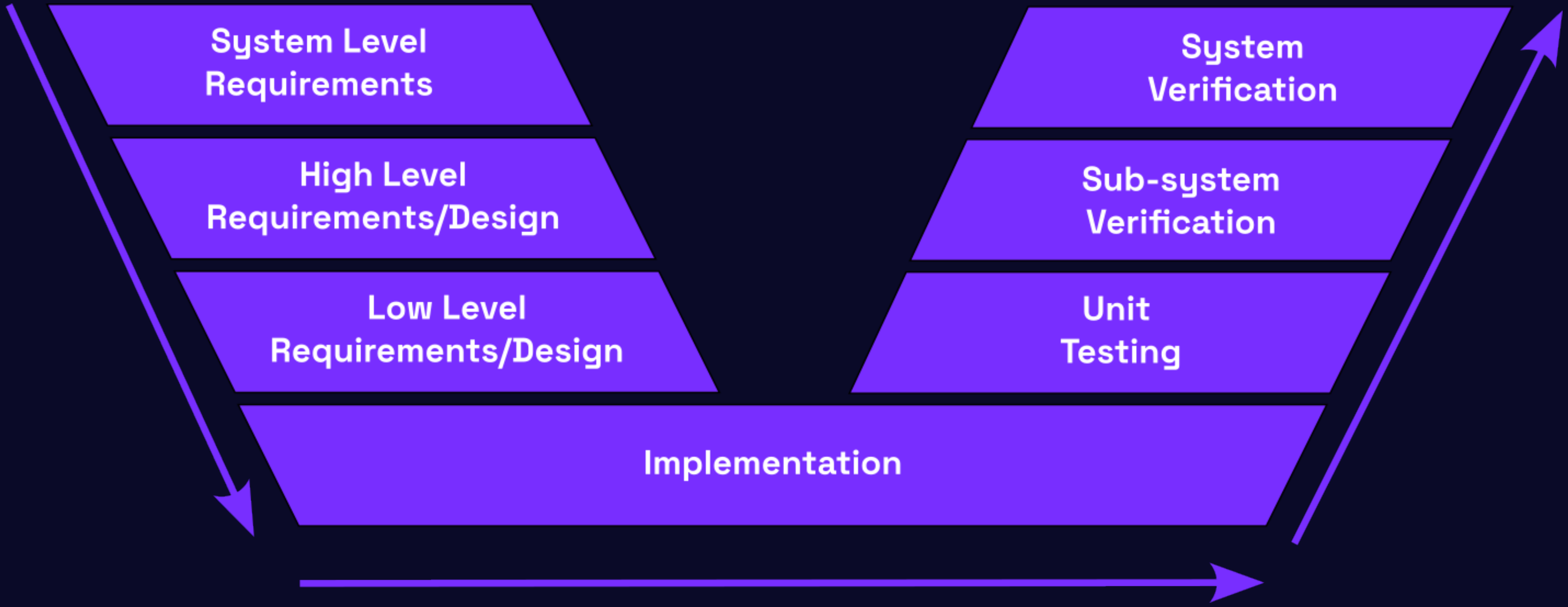


How do we scale?

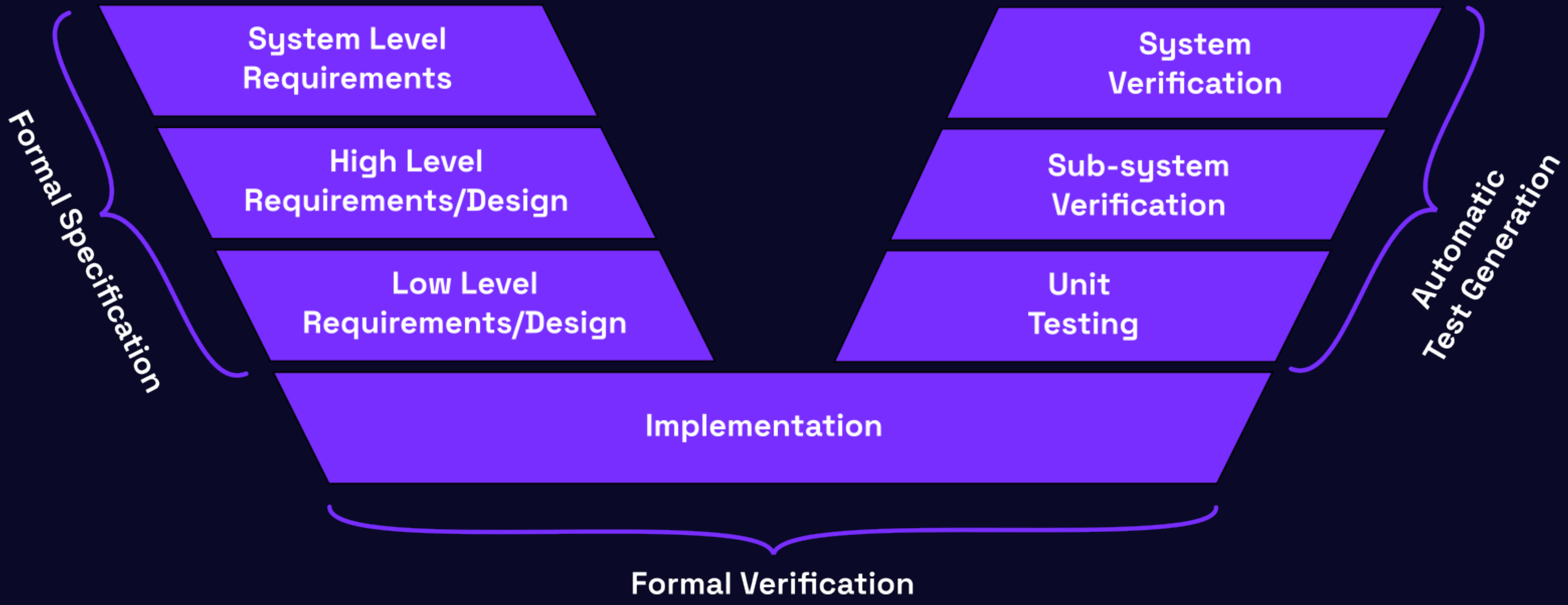




Where are we spending time?


System Level Requirements
High Level Requirements/Design
Low Level Requirements/Design
Implementation
Unit Testing
Sub-system Verification
System Verification

Where does automation come in?

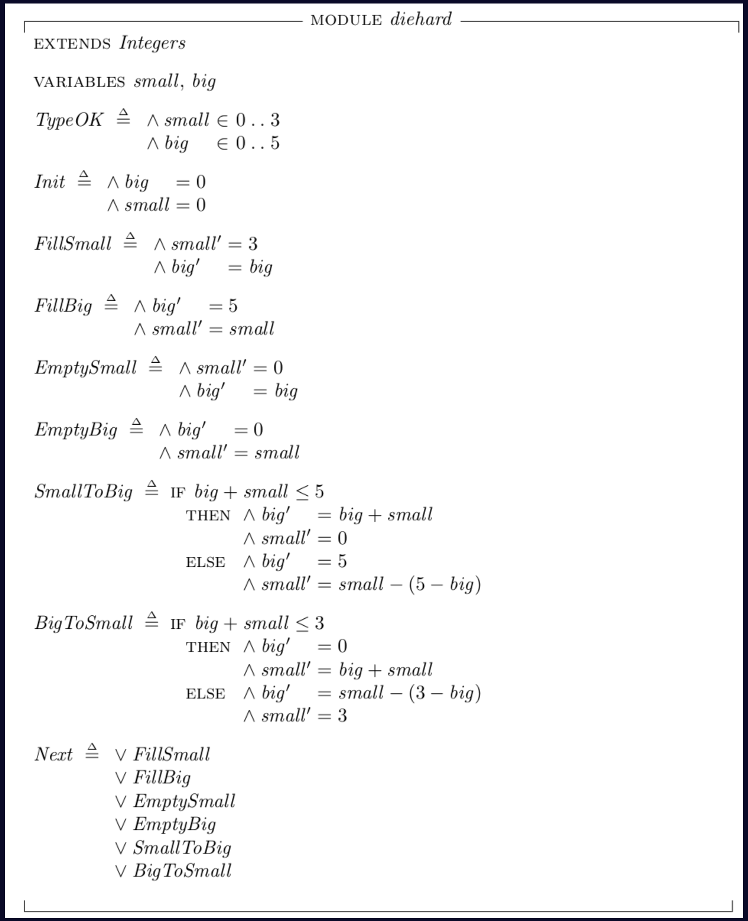

System Level Requirements
High Level Requirements/Design
Low Level Requirements/Design
Implementation
Unit Testing
Sub-system Verification
System Verification
Formal Specification
Automatic Test Generation
Formal Verification

Requirements - Formal specification

- Mathematical proving of models of our system and requirements

- Reduced ambiguity leads to fewer errors

- Examples include TLA$^+$, Alloy, Z...

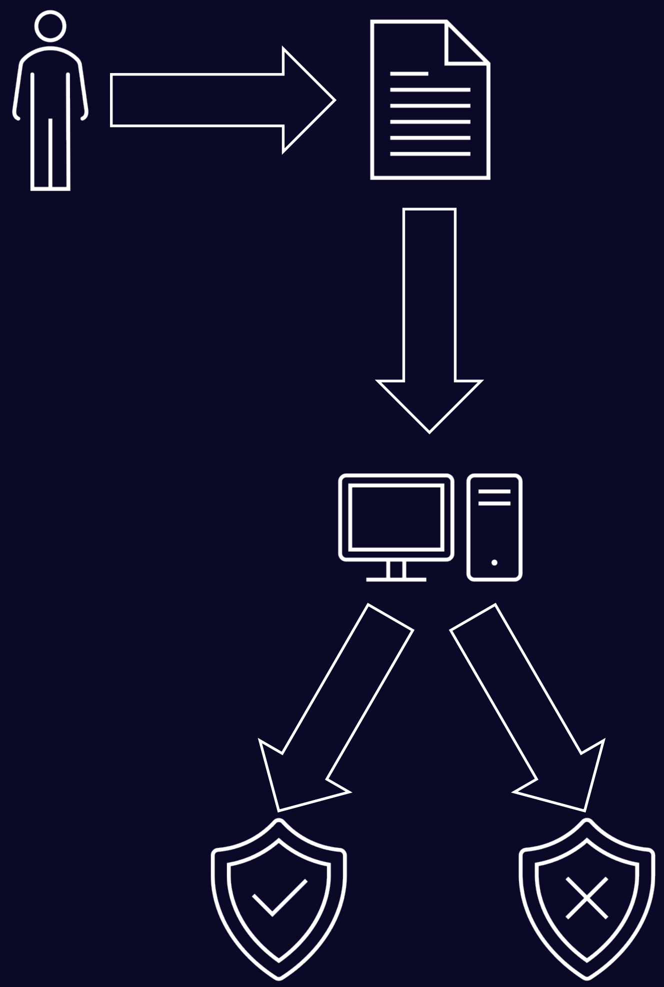


```
MODULE diehard
EXTENDS Integers

VARIABLES small, big

TypeOK ≜ ∧ small ∈ 0 . . 3
         ∧ big   ∈ 0 . . 5

Init ≜ ∧ big   = 0
       ∧ small = 0

FillSmall ≜ ∧ small′ = 3
            ∧ big′   = big

FillBig ≜ ∧ big′   = 5
          ∧ small′ = small

EmptySmall ≜ ∧ small′ = 0
             ∧ big′   = big

EmptyBig ≜ ∧ big′   = 0
           ∧ small′ = small

SmallToBig ≜ IF big + small ≤ 5
                THEN ∧ big′   = big + small
                     ∧ small′ = 0
                ELSE ∧ big′   = 5
                     ∧ small′ = small − (5 − big)

BigToSmall ≜ IF big + small ≤ 3
                THEN ∧ big′   = 0
                     ∧ small′ = big + small
                ELSE ∧ big′   = small − (3 − big)
                     ∧ small′ = 3

Next ≜ ∨ FillSmall
       ∨ FillBig
       ∨ EmptySmall
       ∨ EmptyBig
       ∨ SmallToBig
       ∨ BigToSmall
```

Requirements - Formal specification

Automated:

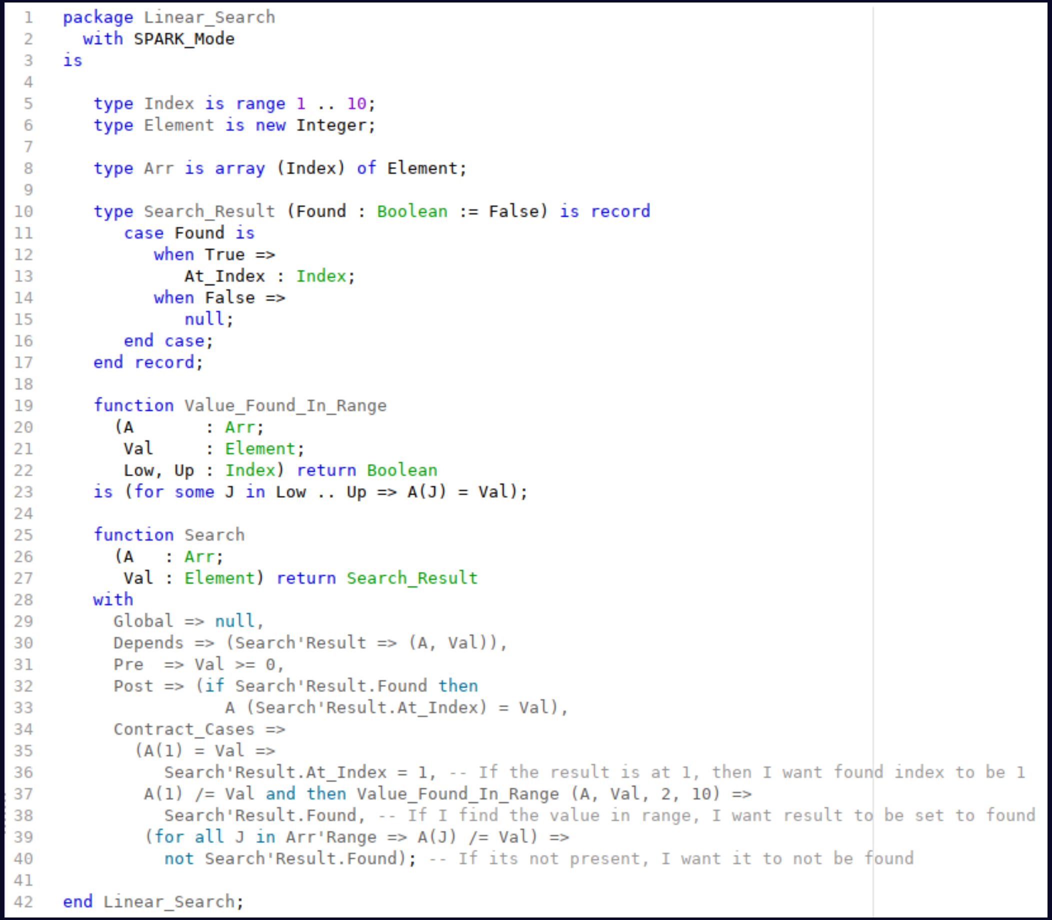
- Type checking and proof
- Code generation
- Test case generation



1. User writes requirements
2. Computer checks logic
3. Automated answer

Implementation - Formal Verification

- Mathematical proving of source code
- Early identification of implementation errors
- Reduces regulatory demand for costly unit testing
- Examples include SPARK Ada, FRAMA C...

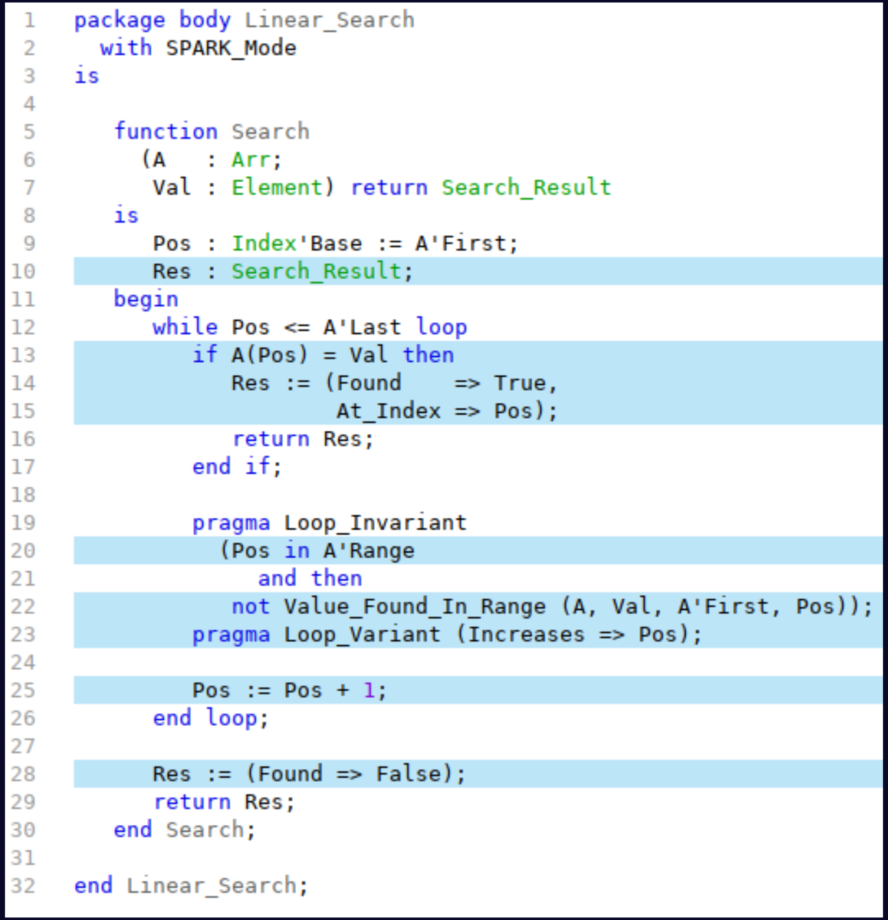


```
package Linear_Search
  with SPARK_Mode
is

   type Index is range 1 .. 10;
   type Element is new Integer;

   type Arr is array (Index) of Element;

   type Search_Result (Found : Boolean := False) is record
      case Found is
         when True =>
            At_Index : Index;
         when False =>
            null;
      end case;
   end record;

   function Value_Found_In_Range
     (A       : Arr;
      Val     : Element;
      Low, Up : Index) return Boolean
   is (for some J in Low .. Up => A(J) = Val);

   function Search
     (A   : Arr;
      Val : Element) return Search_Result
   with
     Global => null,
     Depends => (Search'Result => (A, Val)),
     Pre  => Val >= 0,
     Post => (if Search'Result.Found then
                 A (Search'Result.At_Index) = Val),
     Contract_Cases =>
       (A(1) = Val =>
          Search'Result.At_Index = 1, -- If the result is at 1, then I want found index to be 1
        A(1) /= Val and then Value_Found_In_Range (A, Val, 2, 10) =>
          Search'Result.Found, -- If I find the value in range, I want result to be set to found
        (for all J in Arr'Range => A(J) /= Val) =>
          not Search'Result.Found); -- If its not present, I want it to not be found

end Linear_Search;
```

Implementation - Formal Verification

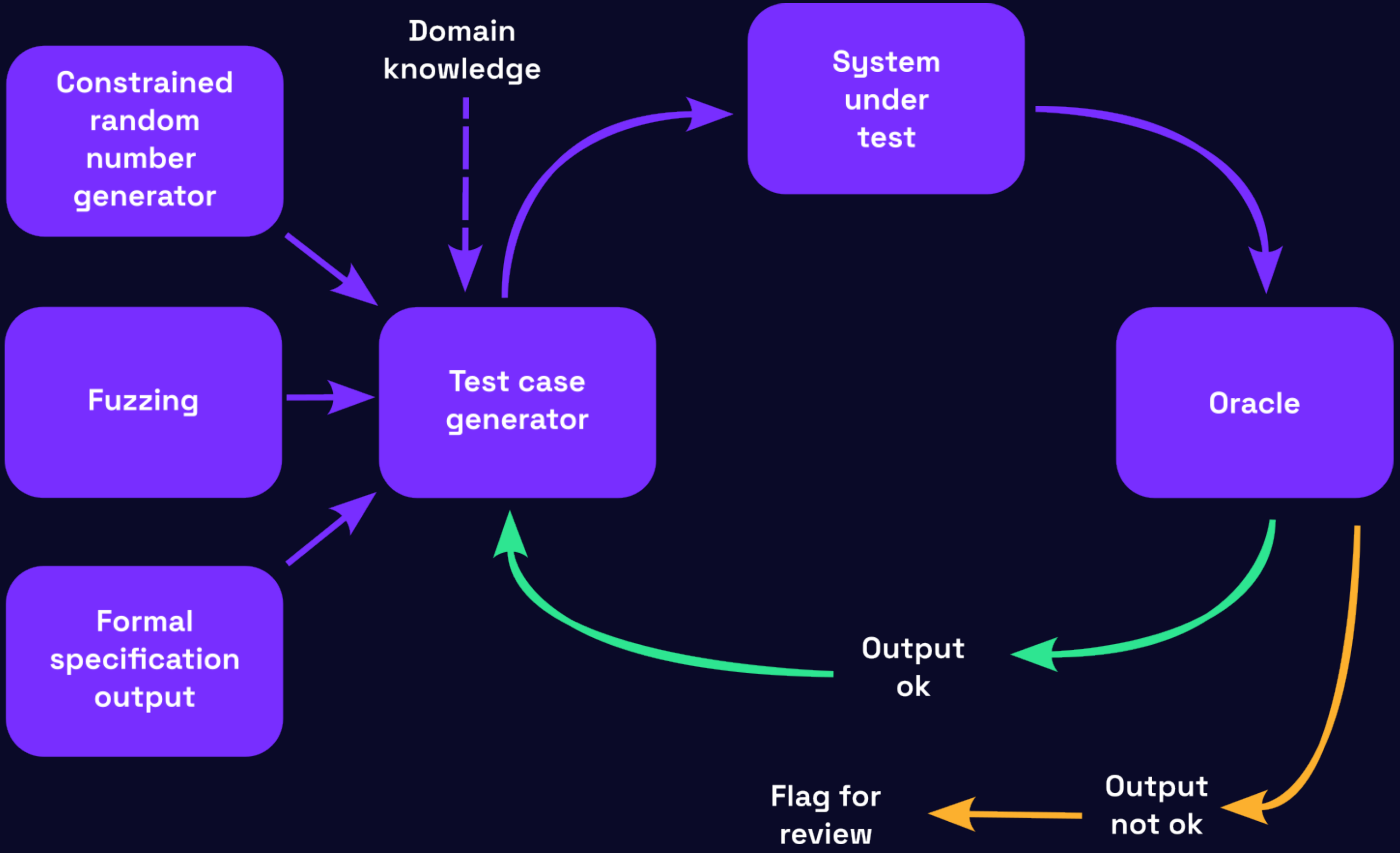
Automated:

- Data & control flow analysis
- Proof of the absence of run-time exceptions
- Dependency contract checking

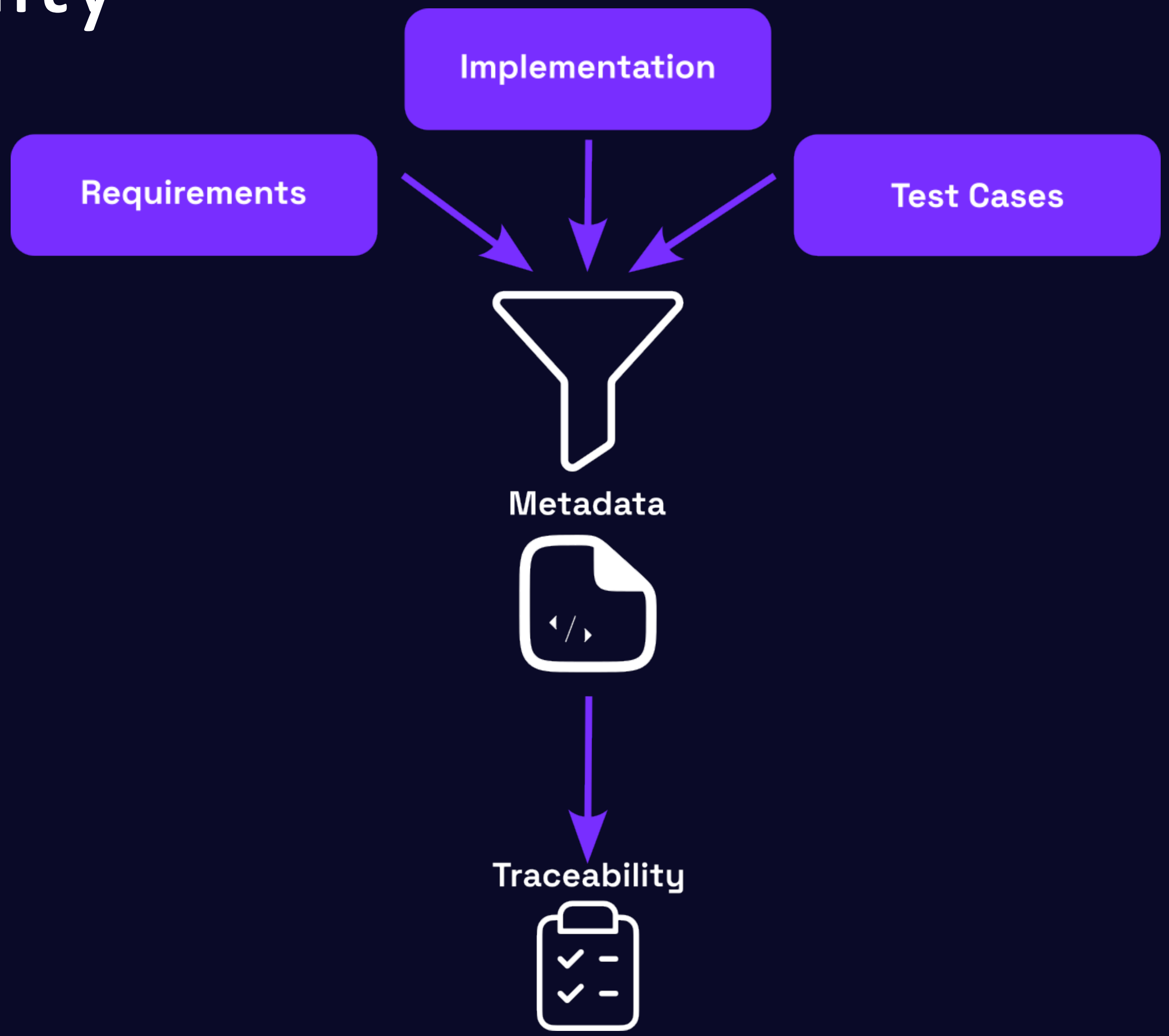


```
package body Linear_Search
  with SPARK_Mode
is

   function Search
     (A   : Arr;
      Val : Element) return Search_Result
   is
      Pos : Index'Base := A'First;
      Res : Search_Result;
   begin
      while Pos <= A'Last loop
         if A(Pos) = Val then
            Res := (Found    => True,
                    At_Index => Pos);
            return Res;
         end if;

         pragma Loop_Invariant
           (Pos in A'Range
              and then
            not Value_Found_In_Range (A, Val, A'First, Pos));
         pragma Loop_Variant (Increases => Pos);

         Pos := Pos + 1;
      end loop;

      Res := (Found => False);
      return Res;
   end Search;

end Linear_Search;
```

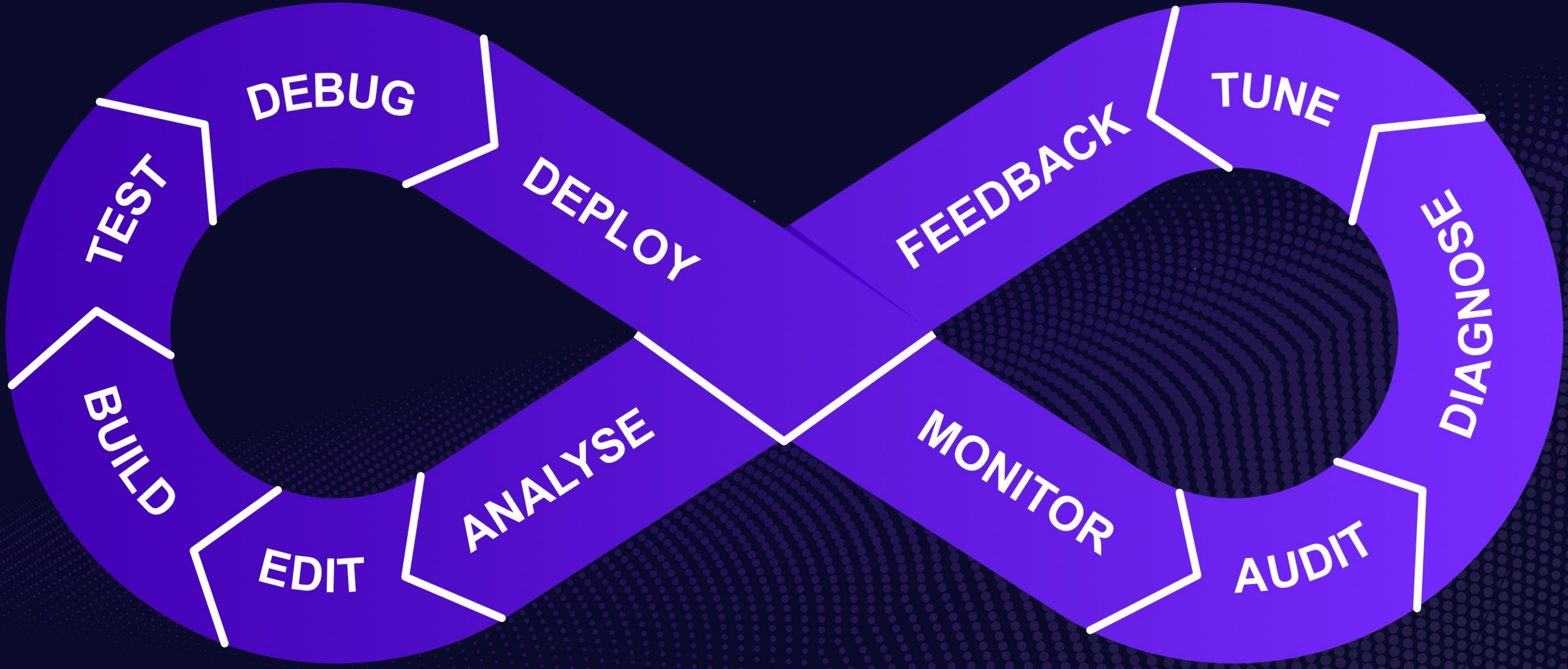
Testing – Automated test cases



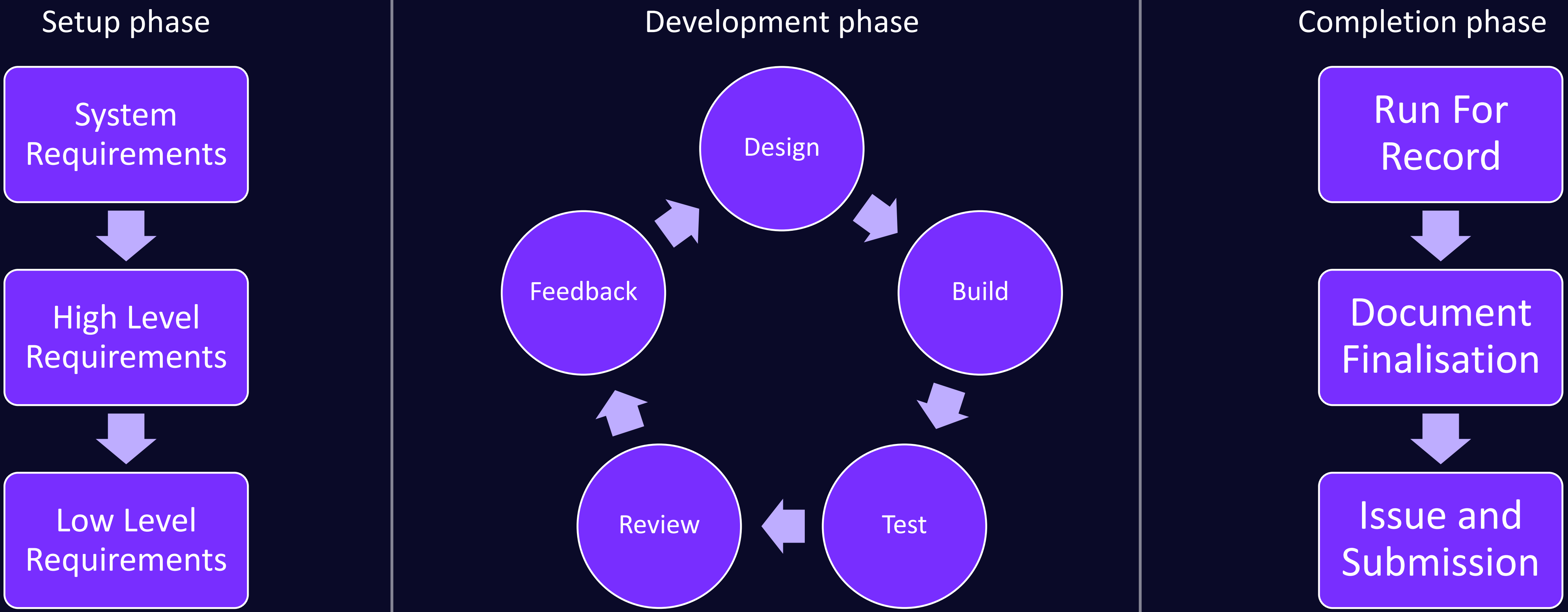
Traceability



DevOps


DEBUG
TEST
BUILD
EDIT
ANALYSE
DEPLOY
FEEDBACK
TUNE
DIAGNOSE
AUDIT
MONITOR

How do we manage projects?



Optimising the human

- Change is coming to the industry and safety needs to be at the forefront

- Many methods to solve the problem, but there is no silver bullet!

- Offload repetitive tasks to automation so we can focus on the complex problems!

Questions?

Follow us on LinkedIn