ELISA
Enabling Linux in Safety Applications

WORKSHOP

Eclipse Trustable Software Framework (TSF)

Paul Albertella, *Codethink*
Daniel Krippner, *ETAS GmbH*

9th May 2025

Introduction

Paul Albertella (@reiterative)

- Consultant at Codethink since 2019
- Certified Functional Safety Practitioner (ISO 26262)
- Developing Codethink's safety approach since 2020
- Currently applying TSF to internal and customer projects

- Providing technical leadership for TSF project
- Working in public since February 2025
- Approved as an Eclipse Foundation project April 2025
- Currently migrating project work into Eclipse

- Contributor to ELISA project since 2019
- Chair of Open Source Engineering Process (OSEP) group

What is TSF and why is it needed?

What?

- A theoretical model for reasoning about software and trust
- A methodology for managing evidence to support claims about this
- A framework for evaluating risk in continuous delivery of critical software

Why?

- Software in critical products is increasingly complex and rapidly changing
- Open source software is ubiquitous and deeply-established in most domains
- Existing safety standards were not developed with either of these in mind
- Safety and security are not the only risk factors for a software project

For more details read: Building Open Safety Standards with the Eclipse Trustable Software Project[1]

Is it suitable for safety?

- TSF is used by Codethink to manage the safety case for **CTRL OS**
  - A Linux-based operating system for use in safety-critical and mixed-criticality systems up to SIL 3 / ASIL D, developed using TSF and RAFIA[1]
- Codethink published a baseline safety case assessment by **exida** this week:
  - https://www.codethink.co.uk/news/trustable-software.html

“The assessment of the process framework as applied to CTRL OS has shown that the relevant safety requirements of IEC 61508 at SIL 3 are met and a process compliance argument is complete with this baseline safety case assessment.”

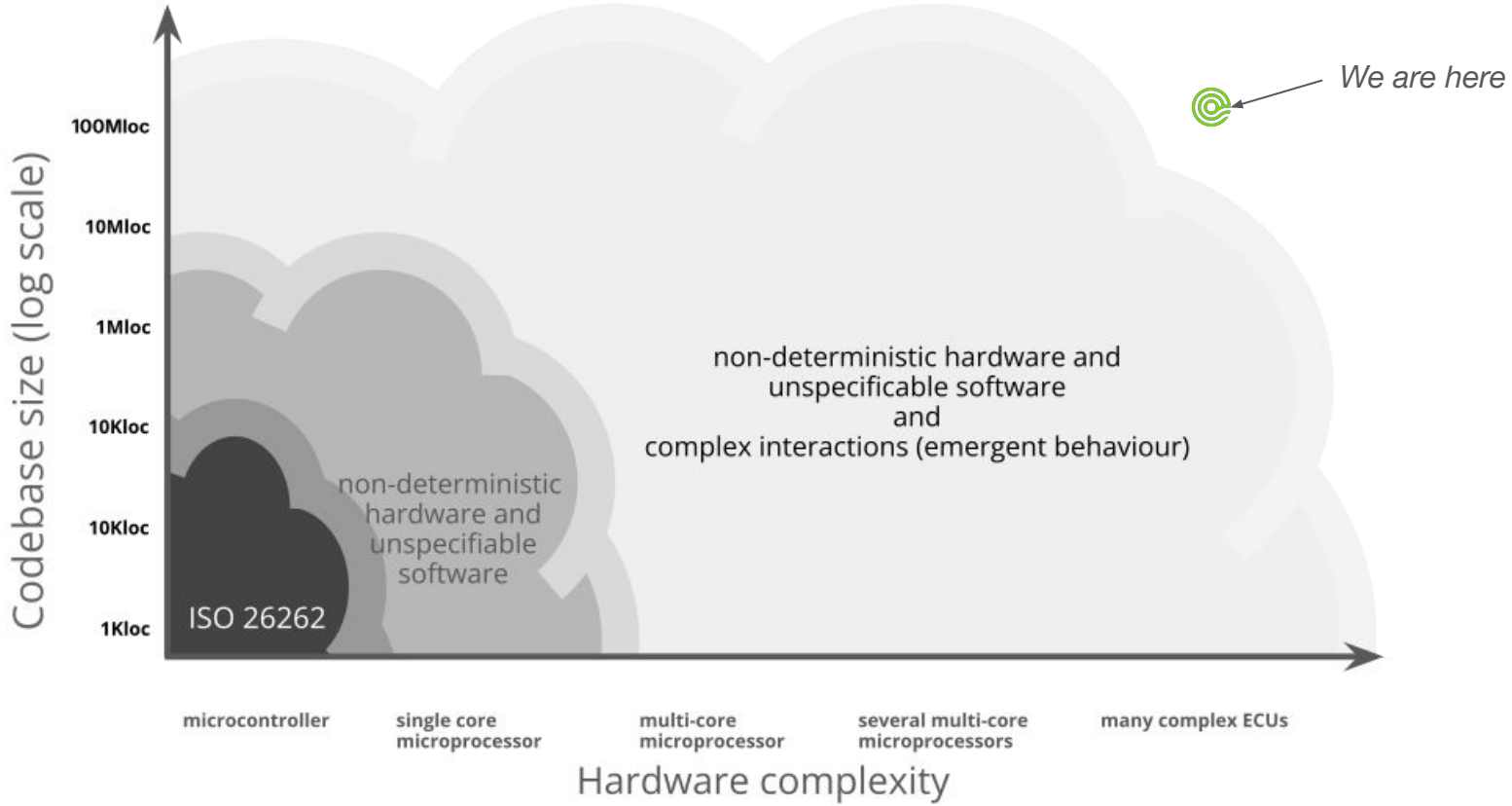
[1] RAFIA: *https://codethinklabs.gitlab.io/trustable/trustable/applications/rafia/index.html*

Why Trustable?

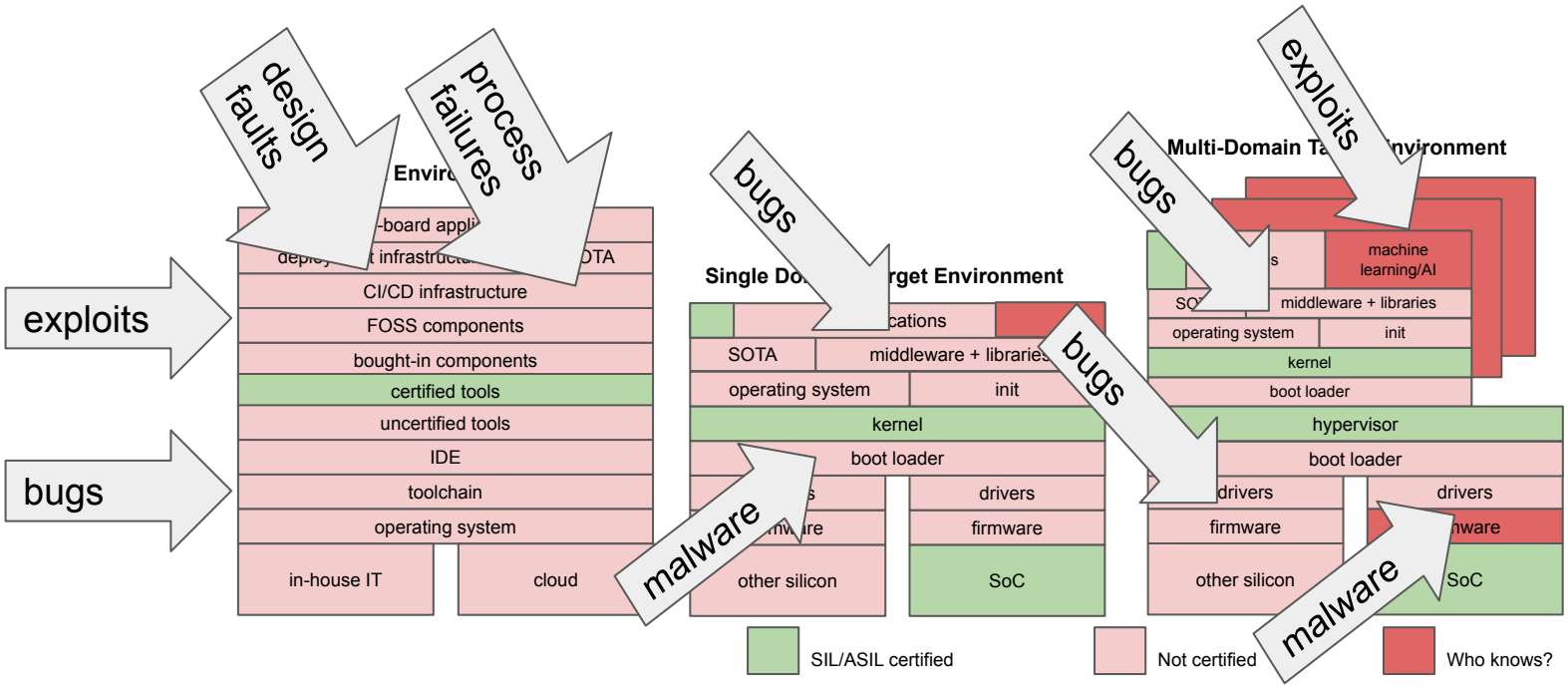
- Safety and security are the key risk factors, but others exist
- Often interconnected, and/or balanced against each other
- Consumers, contributors and stakeholders have different risk factors and priorities
- Need evidence to make informed decisions about **risk**

Trustable, rather than ***trusted*** or ***trustworthy***

A changing risk landscape

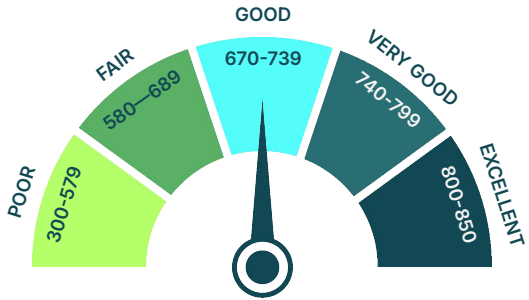


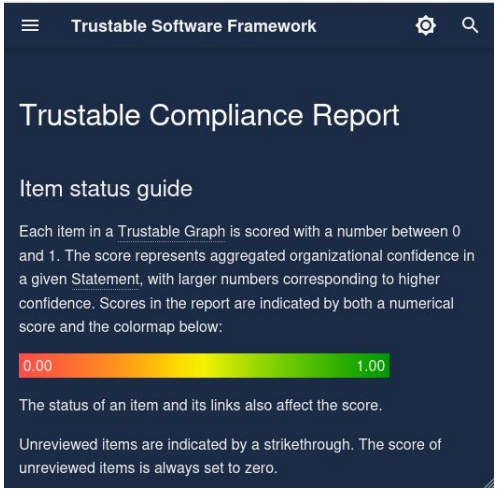
A complex risk landscape



A common frame of reference

- Need consensus about the factors to consider when evaluating risk for critical software
- Use this to drive a **Trustable Score** — like a ‘credit score’ for software
- Enable software projects to organise and evaluate evidence relating to these factors
- Use alongside existing standards to show that the measures and objectives are equivalent
- Develop as a basis for cross-project comparison and improvement





What is the TSF?

What do we mean by a framework?

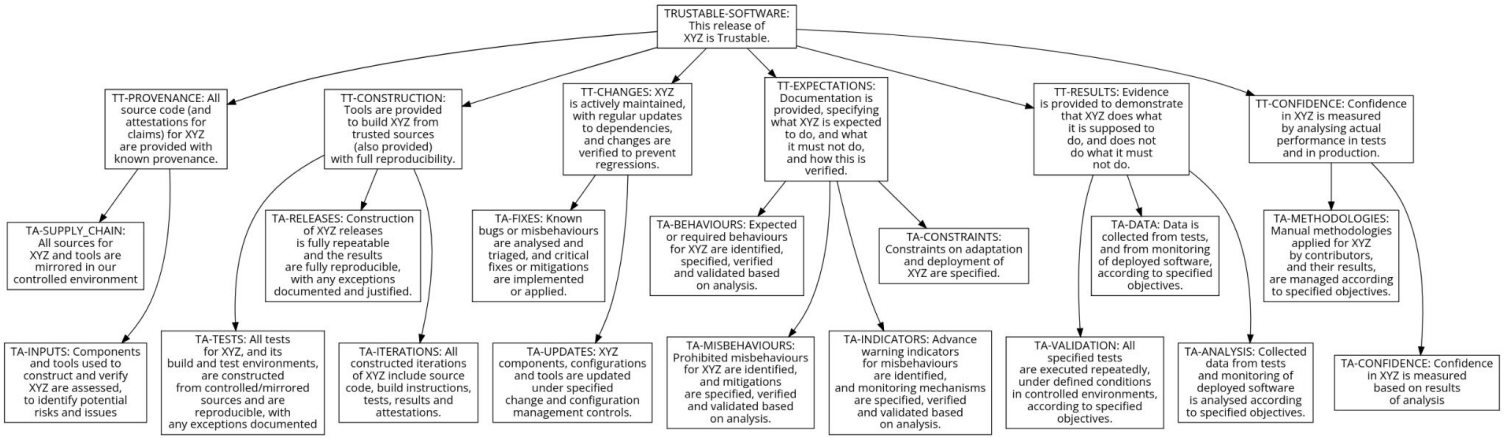
TSF is a **framework**, providing **objectives,** a **model** and a **methodology**.

- **objectives** define what is important, or what we are trying to accomplish
- a **model** is a simplified description of a more complex system or idea, focusing on specific elements or relationships
- a **methodology** is a system of methods used for a particular activity, which may use models
- a **framework** provides practical structures and tools to help apply these methods, while allowing flexibility about how objectives are achieved

Trustable objectives

What evidence is needed for software to be considered ‘trustable’?

- Common set of ‘baseline’ objectives, to be extended with project-specific ones
- Based on established best practices and past experience
- Intended to be extended and refined over time - input very welcome!



Trustable model

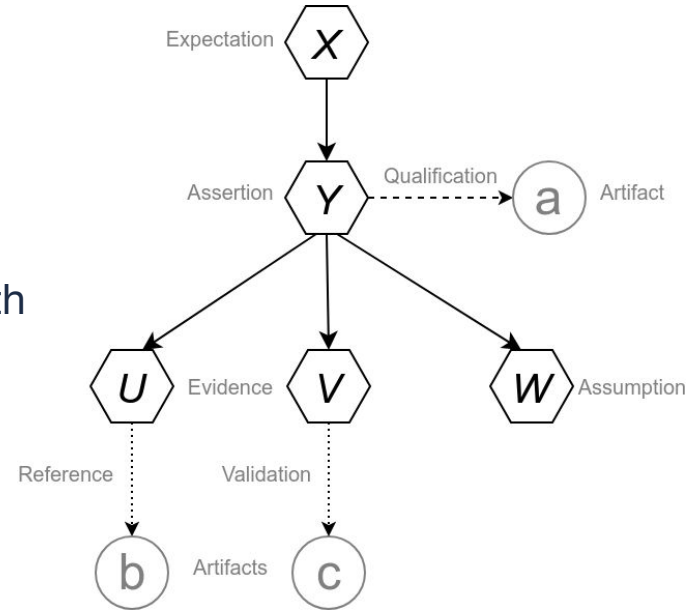
Theoretical model for reasoning about software, based on:

- the **behaviours** or **properties** we expect from it
- the **claims** we make about it
- the **evidence** we provide to support these claims

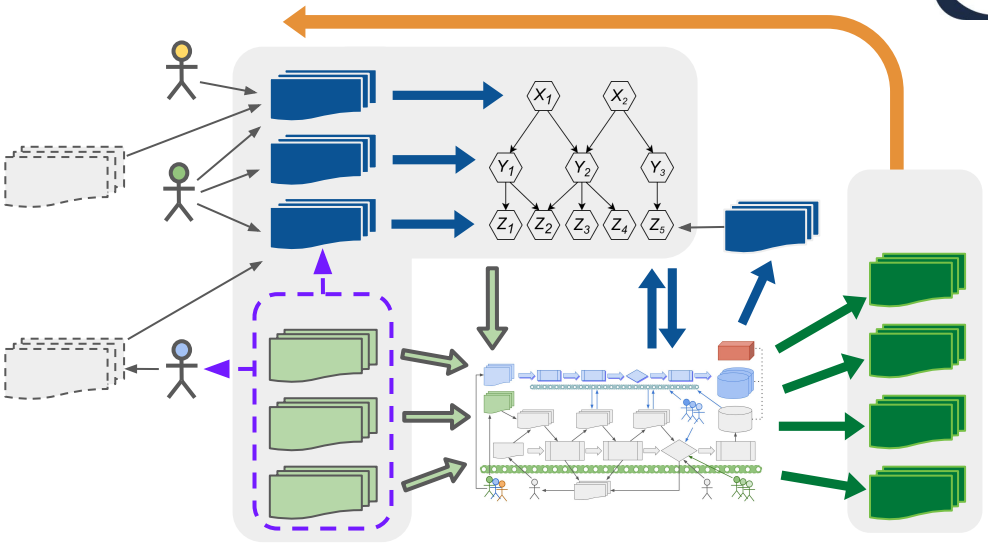
Composed of **Statements** and **Artifacts**.

- Statements express a **Request**, or a **Claim**, or both
- Artifacts **support** a Claim or **qualify** a Request
- **Evidence** is a Claim supported by an Artifact

Linked Statements form a **Trustable Graph**, which stores and organise **project metadata**.



TSF methodology



- **Apply in-context** - as much as appropriate for the project, extending for components
- Map your claims and evidence to the Trustable Objectives
- Document project-specific objectives and Expectations for your software
- Link to requirements or evidence managed in other systems or contexts
- Map Trustable and project-specific objectives and evidence to the corresponding requirements defined by standards

Trustable Objectives

Trustable Objectives

We can offer software as Trustable if we can provide **evidence** to support all of these claims…

1. Provenance

We know where its inputs come from, who is responsible, and our confidence in them

2. Construction

We can build it - **reproducibly** - from source

3. Changes

We can upgrade it and it will not break or regress

4. Expectations

We know what it must do, and what it must not do

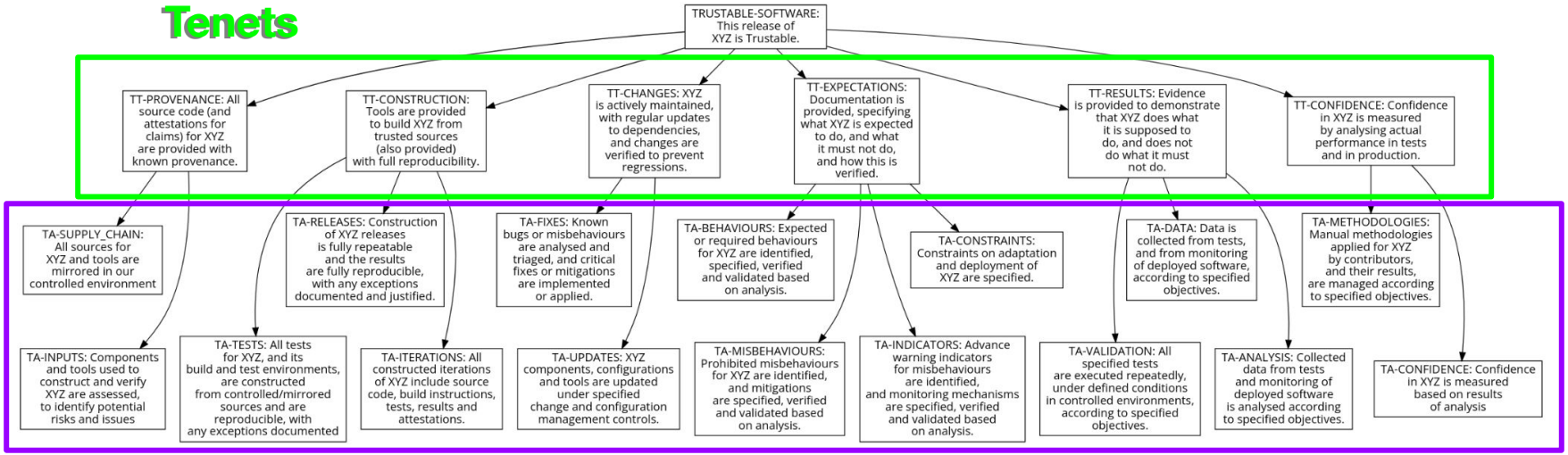
5. Results

We show that it does what it must do, and does not do what it must not do

6. Confidence

We measure and declare our confidence that it satisfies its other claims

Tenets and Assertions



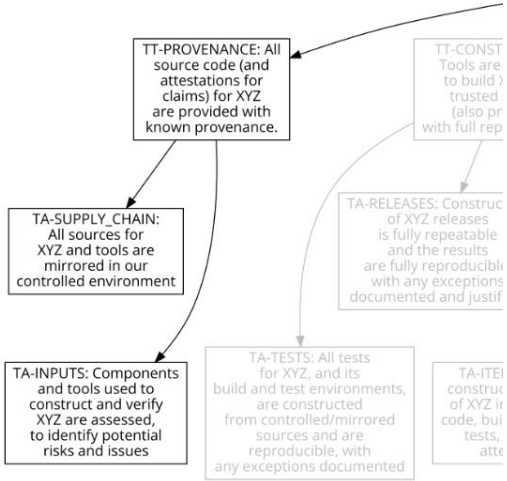
A common set of Statements maintained by the TSF, describing the evidence needed to determine whether a given iteration of a software project ("XYZ") is **Trustable**

- The **Tenets** (TT-*xxx*) describe a set of high level goals for trustability
- The **Assertions** (TA-*xxx*) break these Tenets down into more specific objectives

Provenance

Understand all of your external dependencies, including tools and toolchain components, and why — or to what extent — you can trust them.

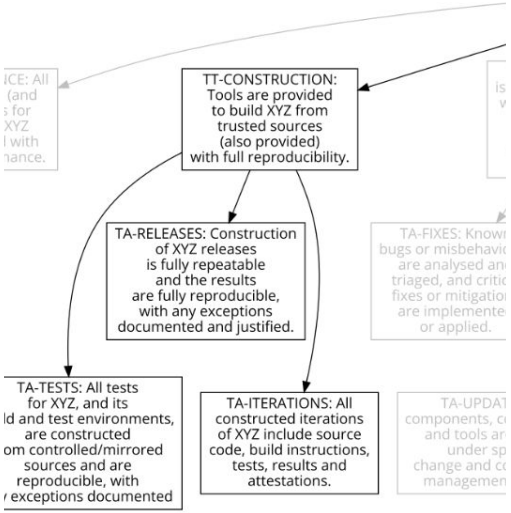
- **Supply Chain** - Mirror all your external dependencies using infrastructure that you control, to avoid them changing or disappearing unexpectedly.
- **Inputs** - Assess (and regularly reassess) all of your dependencies, to identify potential risks and issues, including those identified by their providers.



Construction

Understand and control how your software is constructed, and the tools and dependencies that are used.

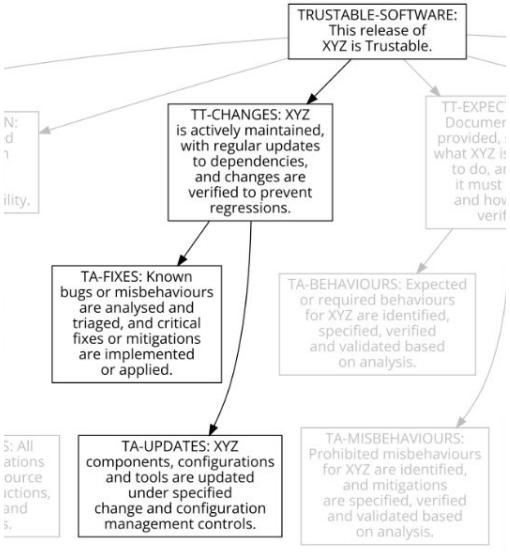
- **Releases** - Releases of your software should be both repeatable and reproducible, to confirm that you have control over *all* of the inputs.
- **Tests** - Apply the same principles when constructing tests and the environments in which you run them.
- **Iterations** - Confirm this for *every* iteration of your software, to avoid surprises on release day!



Changes

Control and verify every change to your software, its dependencies and its toolchain(s), to prevent regressions — but also update tools and dependencies regularly!

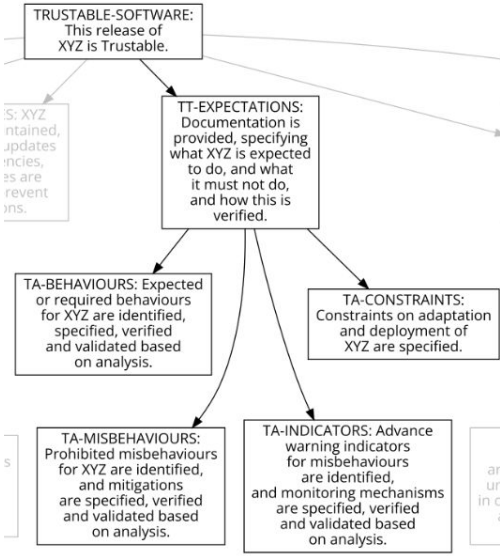
- **Fixes** - Analyse and triage bugs identified by your project, or by external providers, and apply fixes.
- **Updates** - Apply the same controls to all updates, and coordinate changes to tools or shared dependencies to avoid integration problems later.



Expectations

Document what your software is expected to do, how this is verified and how issues are detected and mitigated.

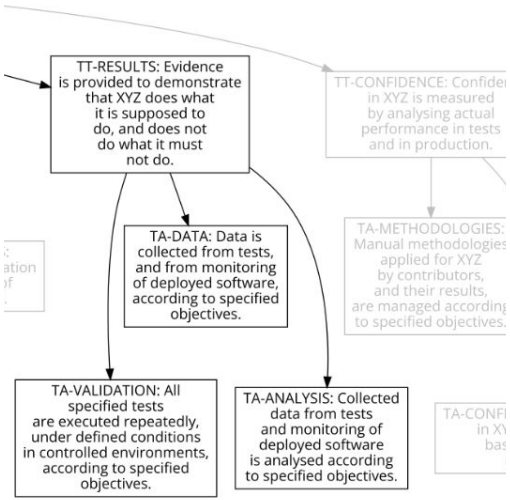
- **Behaviours** - What it is supposed to do (and not do).
- **Misbehaviours** - How this can go wrong, and how to prevent this or deal with the consequences.
- **Indicators** - What is monitored to detect and proactively respond to potential misbehaviours.
- **Constraints** - Limitations, restrictions or assumptions about how the software is to be used.



Results

Evidence that your software satisfies its expectations, and how you ensure that this continues to be the case.

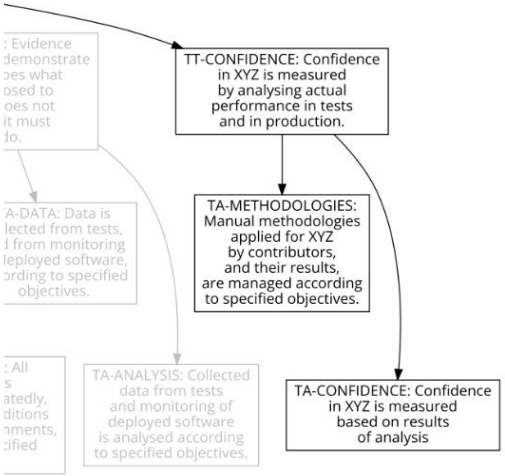
- **Data** - What and how data is collected during tests, *and* from deployed software, to verify its Behaviour and detect or identify Misbehaviours.
- **Validation** - Confirming that tests and mitigations detect and respond to Misbehaviours as intended.
- **Analysis** - Examine data to identify patterns or anomalies, which may indicate Misbehaviours.



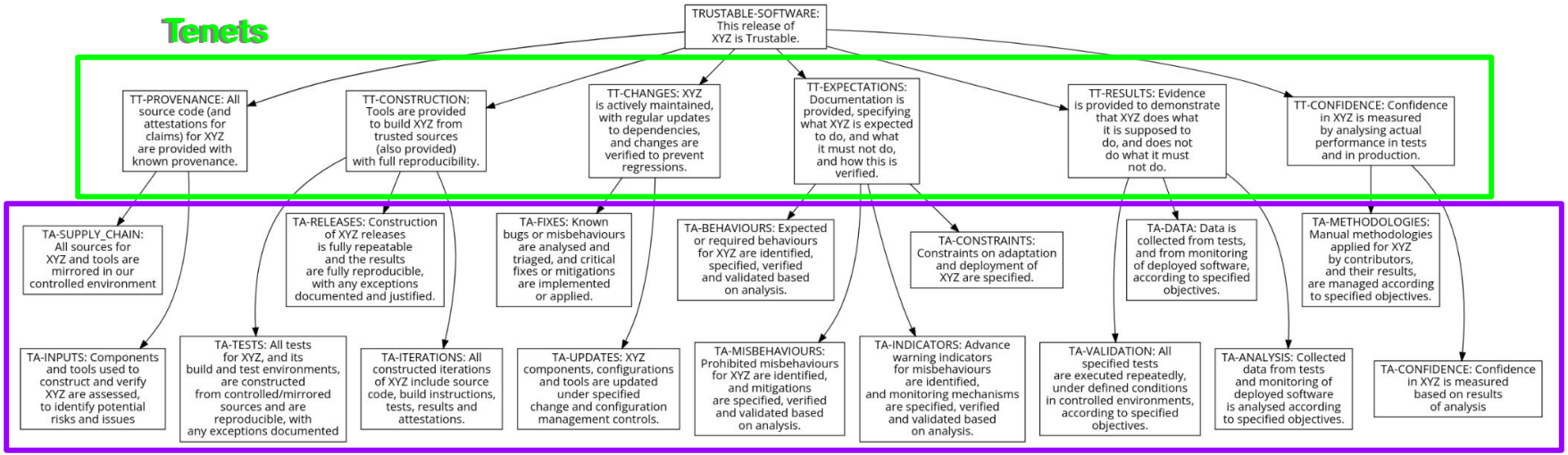
Confidence

How you measure your confidence in your software, and the processes that you use to construct and verify it.

- **Methodologies** - Techniques or strategies used by contributors for other objectives, and how you verify that these have been applied correctly.
- **Confidence** - How you measure and record confidence in your software, and how this data is used to inform activities and priorities.

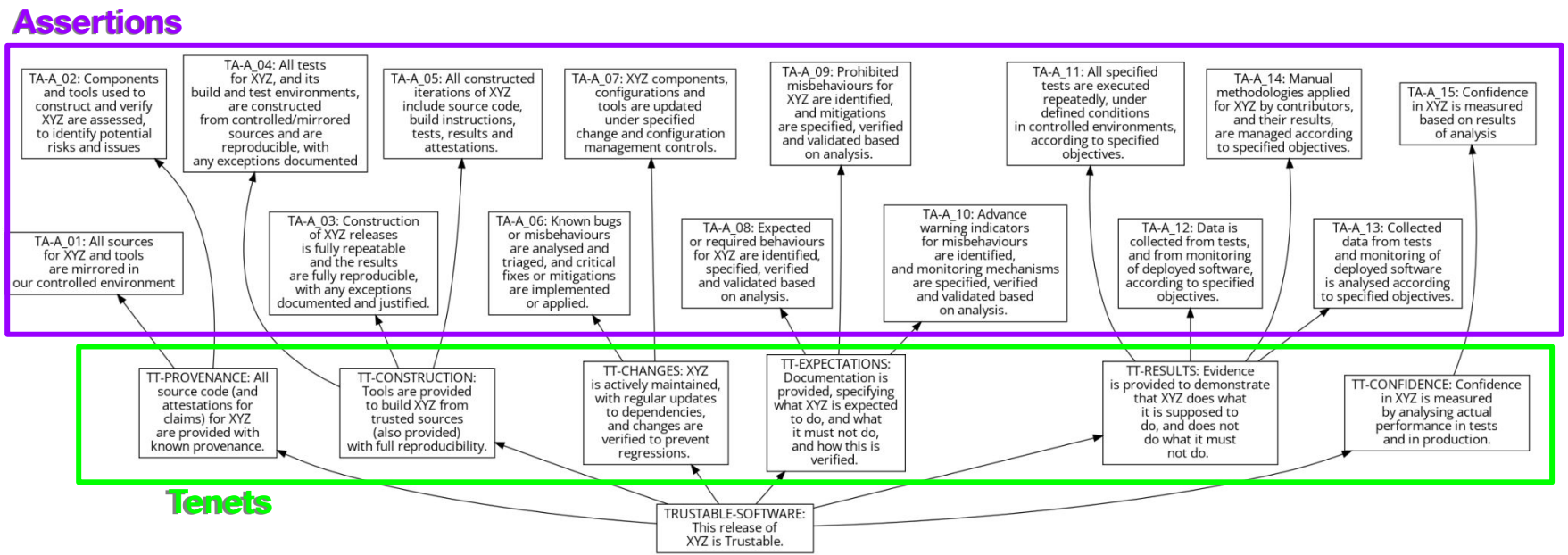


Building out from the objectives



Building ~~out~~ **up** from the objectives

Your Statements go here!



Trustable Model

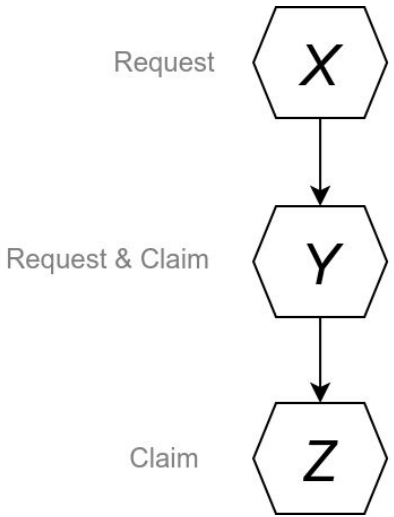
Why do we need a model?

- TSF is domain-agnostic and evidence-based
  - Use generic terminology to establish fundamental concepts
  - Enable all users make their own judgement about evidence
- Express complex ideas using simple elements
  - Small set of ‘building block’ elements and rules
  - Language rules simple enough to enforce
  - Structure rules that can be verified mathematically
- Structure for recording, collecting and deriving metrics
  - Confidence scores recorded by contributors
  - Data-driven scores from collected test results and monitoring data
  - Metrics derived from scores to feed into risk evaluation and project management

Statements and Artifacts

Fundamental elements of the TSF model

- **Statements** define some aspect of the software
  - A single sentence that can be True or False
  - Used to express a **Request**, or a **Claim**, or both
  - **Linked** to other Statements to show dependencies
- **Artifacts** support a Claim or qualify a Request
  - **Qualifying artifacts** provide more detailed information about a Request
  - **Evidence artifacts** provide support for a Claim



Making a Statement (example)

SMA-03

Project tracks known security advisories for dependencies.

Supported Requests:

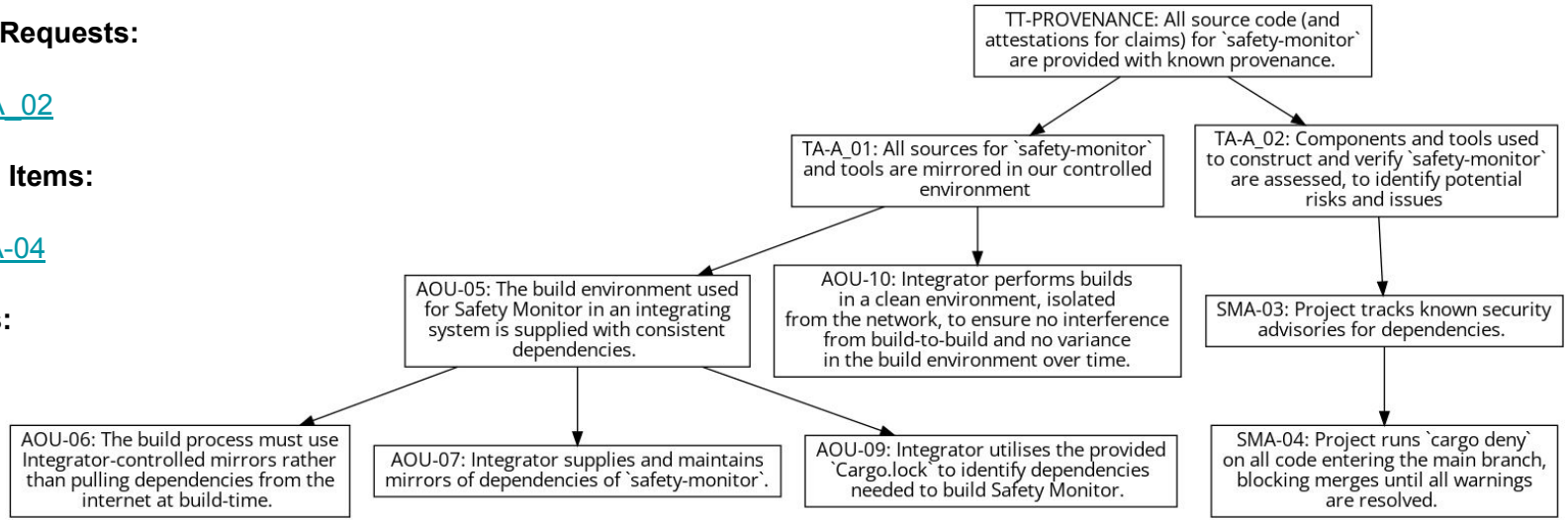
- TA-A_02

Supporting Items:

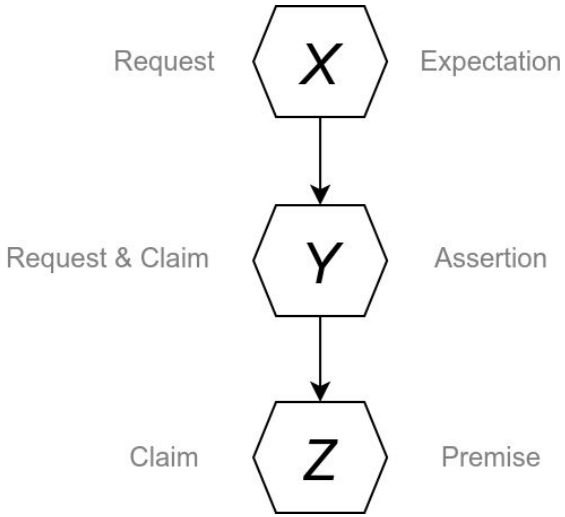
- SMA-04

References:

None



Classifying Statements



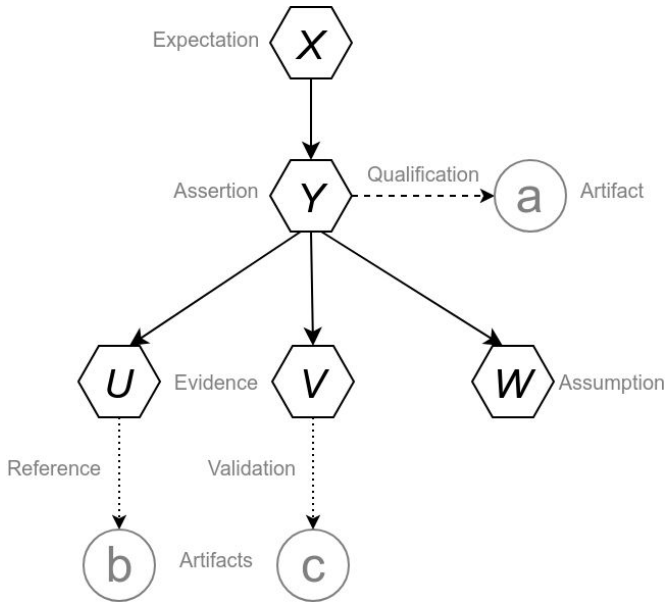
Classifications characterise the role of Statements in a given **context**:

- Request only: **Expectation**
- Claim and Request: **Assertion**
- Claim only: **Premise**

Contexts reflects **boundaries** or **abstraction levels**:

- A Premise in one context may be an Expectation in another (e.g. the AOU Statements in the example)
- An Expectation for a subsystem may be treated as an Assertion at the system level

Statements and Artifacts



- Assertions may be qualified by an Artifact
- A Premise with an Artifact is **Evidence**
  - The Statement describes the Claim
  - The Artifact must support this Claim
- A Premise without an Artifact is an **Assumption**
  - **Gap**: evidence not yet provided by the project
  - **Dependency**: evidence to be provided in the context of a system using the software

Linking to Evidence (example)

SMA-01

The `safety-monitor` project CI periodically executes the integration test suite, and failures in these runs are investigated by contributors; resolution of the identified causes of these failures is tracked by GitLab issues.

Supported Requests:

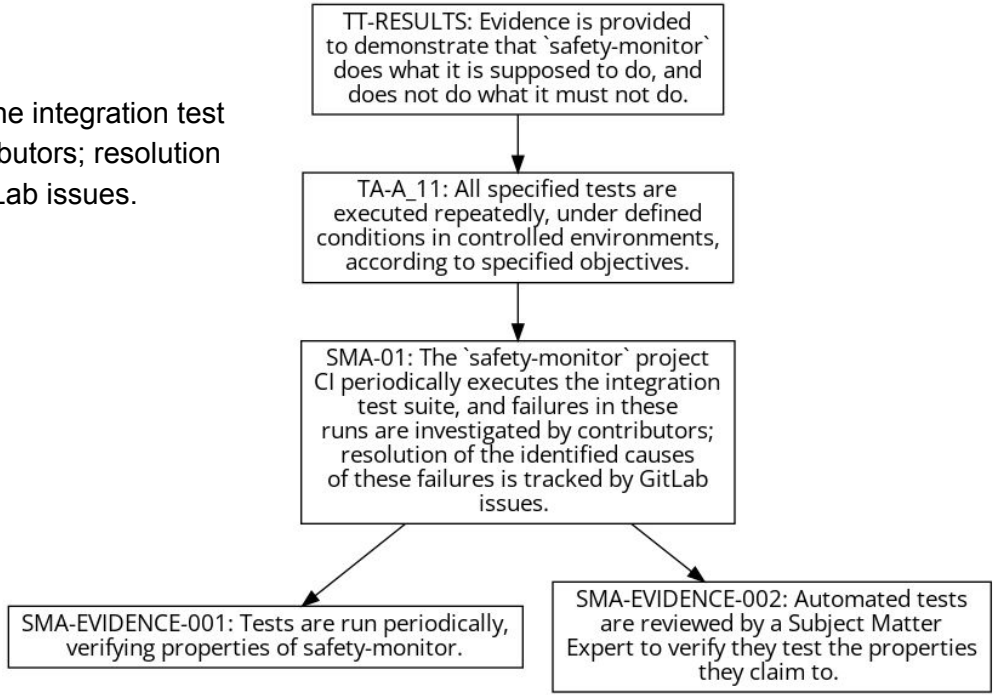
- TA-A_11

Supporting Items:

- SMA-EVIDENCE-001
- SMA-EVIDENCE-002

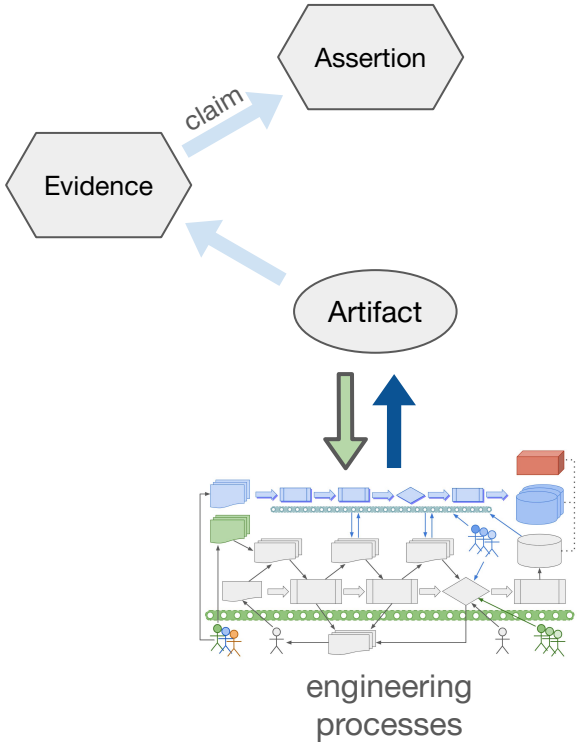
References:

None



Evidence means Artifacts!

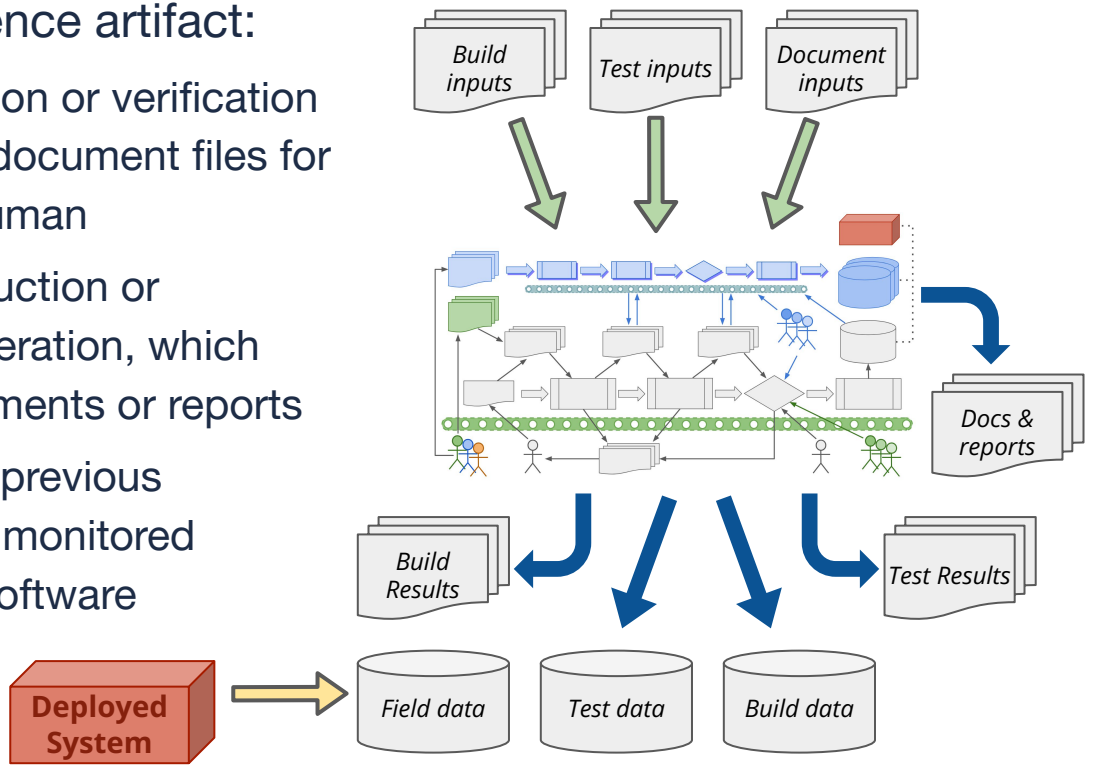
- An Evidence Statement makes a Claim about an Artifact with respect to a Request made by another Statement
- Artifacts must *always* relate to the software itself, or to the results of software engineering processes applied as part of its development



Types of Evidence

There are broad types of evidence artifact:

- **Inputs**: Inputs to a construction or verification process, which may include document files for processes performed by a human
- **Results**: Outputs of a construction or verification process for this iteration, which may include generated documents or reports
- **Data**: Test data collected for previous iterations and field data from monitored system deployments of the software



Evaluating Evidence

- Designed to support **scoring** of the Claims captured in Statements
  - Scores are **only** assigned to Evidence!
  - Scores come in two categories
- **Confidence scores** are committed in the graph by a human
  - Result of an assessment of the evidence by a **Subject Matter Expert**
- **Validator scores** are calculated by an automated process, based on:
  - Result artifacts produced during construction and verification for this iteration
  - Data artifacts collected for previous iterations or from deployments
- This part of TSF is still being developed
  - Planned features include **weights**, to define the relative importance of contributing Assertions and Evidence in the graph

TSF Methodology

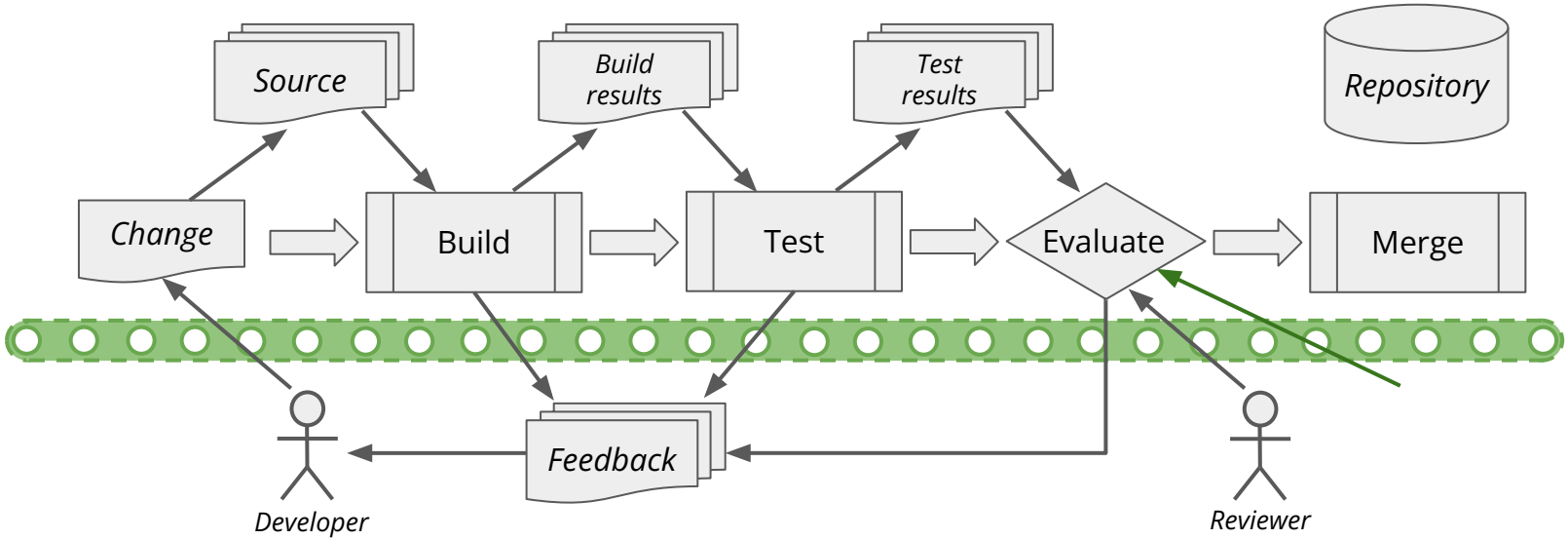
Foundations

- *Everything*-as-code
  - Store **inputs** to construction and verification processes rather than their results
  - Store as plain text where possible and manage **everything** under version control
- Coordinated change and configuration management using git
  - Store inputs in git repositories, managed by a single ‘forge’ (e.g. GitLab, GitHub)
  - Maintain a **mainline** branch as the “source of truth” for each repository
  - Apply controls at the point of **merge** (incorporation of changes from a branch)
  - Manage the versions of inputs from other repositories using SHA[1] or tag
- Pre-merge verification and approval
  - A set of automated tests **must** succeed for the branch before it can be merged
  - The set of automated tests is configured and managed as part of the repository
  - Merges may also require review or approval by designated individuals or groups

[1] *Value produced by a Secure Hash Algorithm*

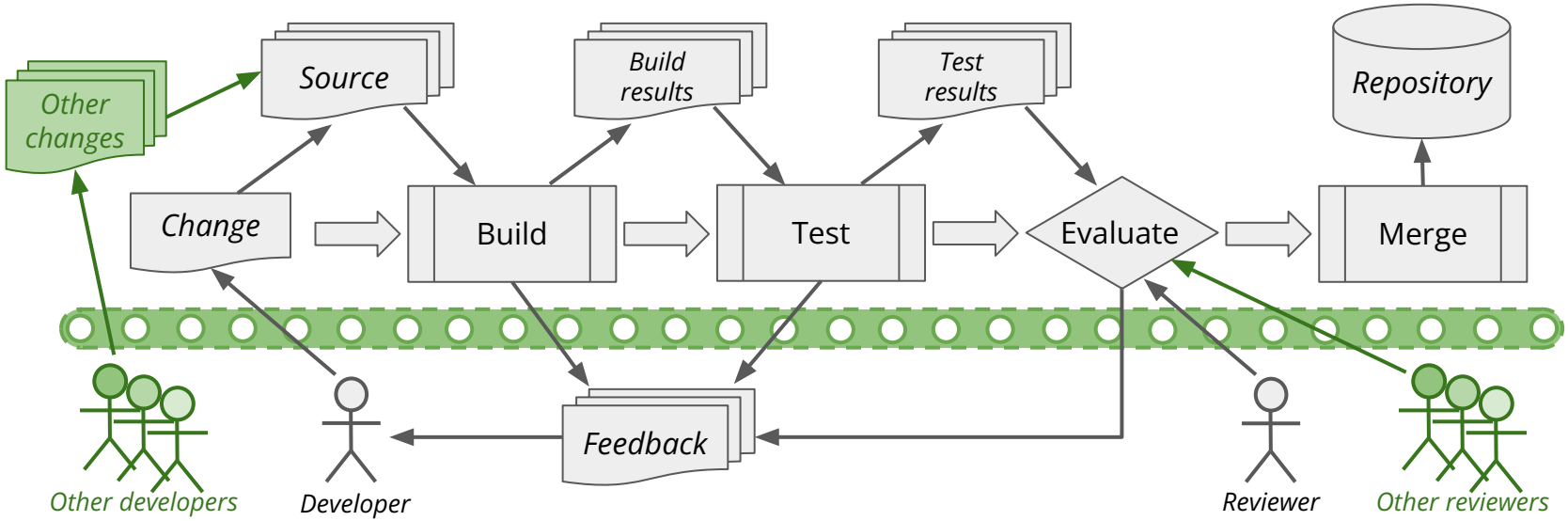
Foundations: Software as a production line

- Pre-merge verification
  - Changes must be built, tested and reviewed before merge is allowed
- Landing changes in a shared repository
  - Specifically: the mainline branch for that repository

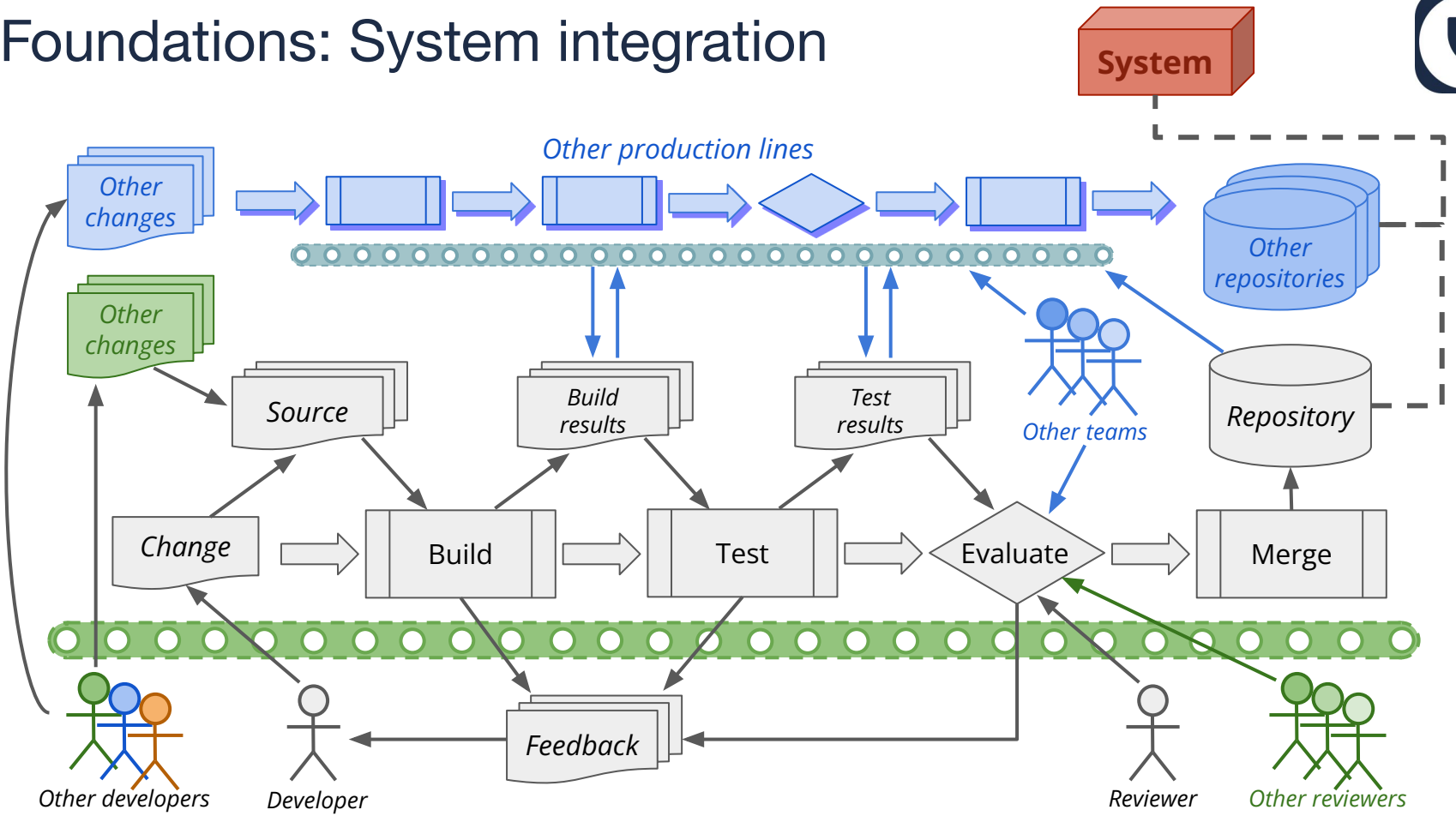


Foundations: Interacting changes

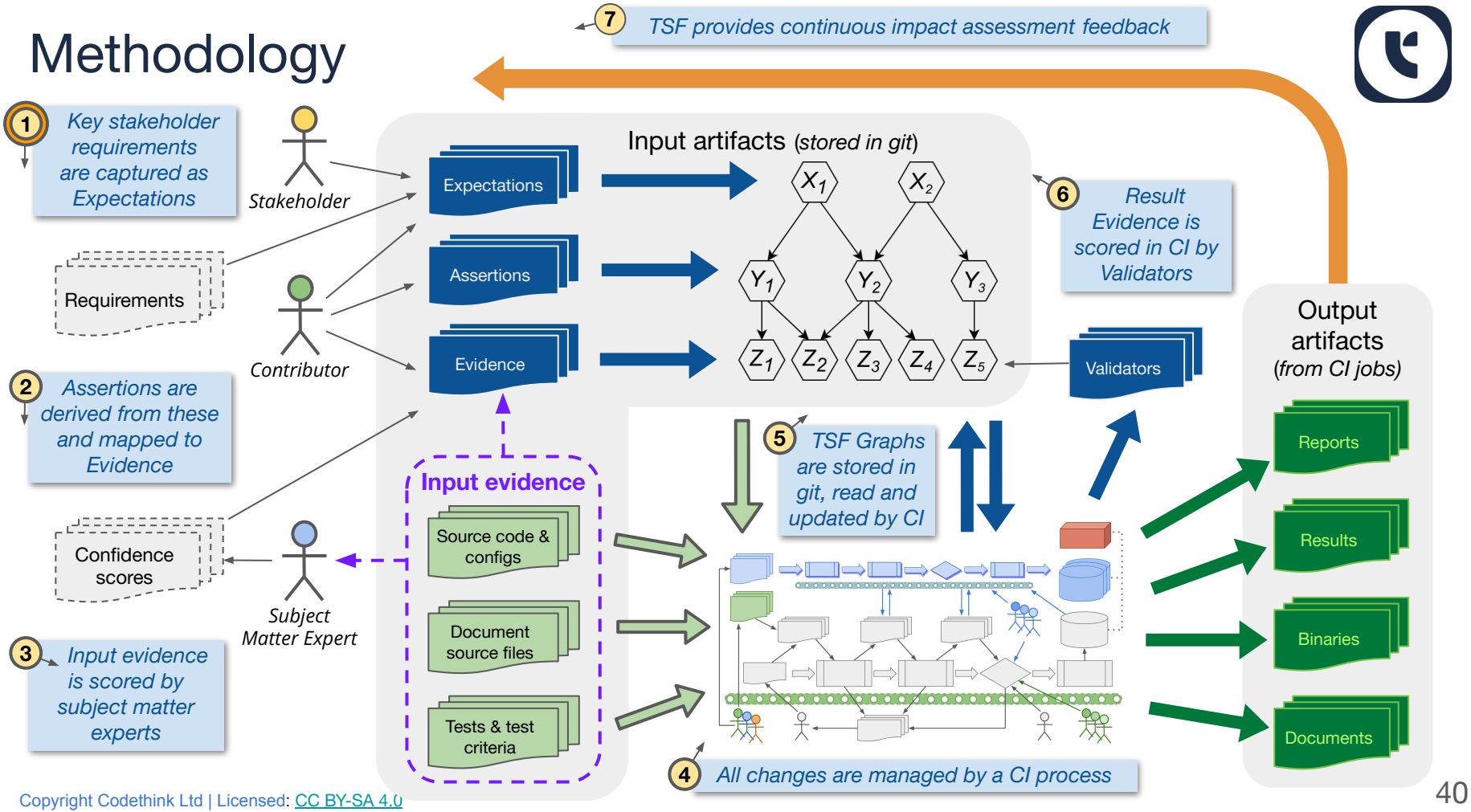
- Changes are not processed in isolation
  - Other developers are working with the same source
  - Changes may depend upon or conflict with each other



Foundations: System integration

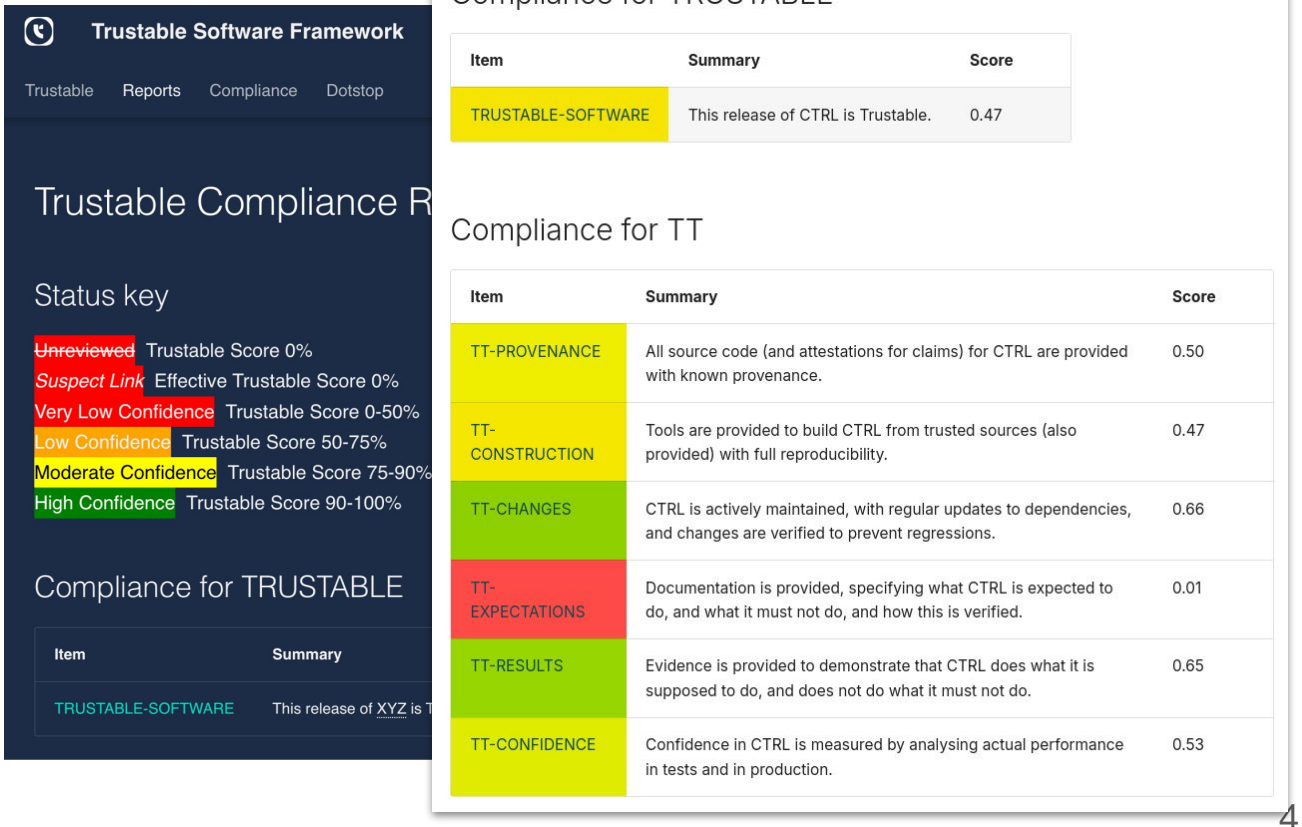


Methodology



Using scoring to guide activities and priorities

- Produce Trustable report for main and development branches
- Track progress towards objectives and assess impacts of a change
- Integrate with automated testing using validators to see and link to ‘live’ results
- Use confidence scores to give feedback on gaps or work required



TSF Tooling

- Command line tools and libraries written in Python to:
  - Manage a stored representation of a TSF Graph in a git repository
  - Publish documentation and reports, and plot visualisations of a TSF graph
  - Define a plug-in ‘validator’ interface for automated evidence scoring
  - Calculate metrics based on evidence scores and weights
- Under very active development!
  - Was originally based on Doorstop[1], but now a standalone tool (**trudag**)
  - Retains legacy support for Doorstop as a data format
  - Included as part of the main TSF project
  - Currently extending to add support for remote graphs and evidence

[1] *https://doorstop.readthedocs.io*

Feedback on using TSF

Using TSF for uProtocol

Summary and next steps

Summary

- A new approach is needed to manage risk in critical systems using software that is complex or non-deterministic, whether proprietary or open source
- The Trustable Objectives define a common set of factors that should be considered when evaluating risk for any software project
- The Trustable Software Framework enables projects to:
  - Document their approach to satisfying the Trustable Objectives
  - Define project-specific objectives alongside these
  - Collect, organise and evaluate evidence to support their objectives
- The Eclipse Trustable Software Framework project has been established to continue development of this approach in the open

Future plans

- Complete migration of documentation and tooling into Eclipse Foundation
- Provide more examples of how TSF can be applied
- Extend tooling to support references to remote graphs and evidence
- Extend the scoring approach to support weights
- Start building a community to shape and contribute to the project
- Support other projects applying TSF in the open and use their feedback to drive improvements and add new use cases

Where to find more information

Introductory talks and article

- FOSDEM: https://www.youtube.com/watch?v=2TS5EENC6Ms
- SDV Community Day: https://www.youtube.com/watch?v=lyp3b2e35iY
- Building Open Safety Standards with the Eclipse Trustable Software Project

TSF project home in the Eclipse Foundation

- https://projects.eclipse.org/projects/technology.tsf

TSF project documentation (*temporary home on gitlab.com*)

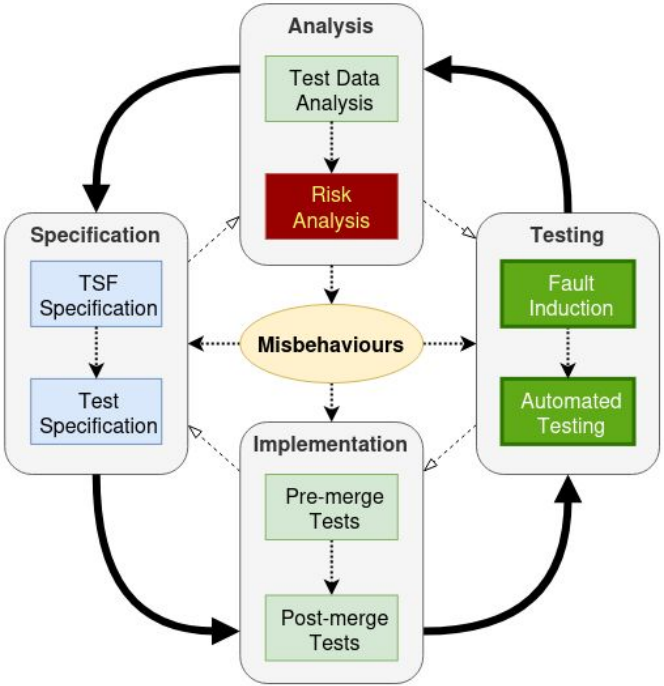
- https://codethinklabs.gitlab.io/trustable/trustable/

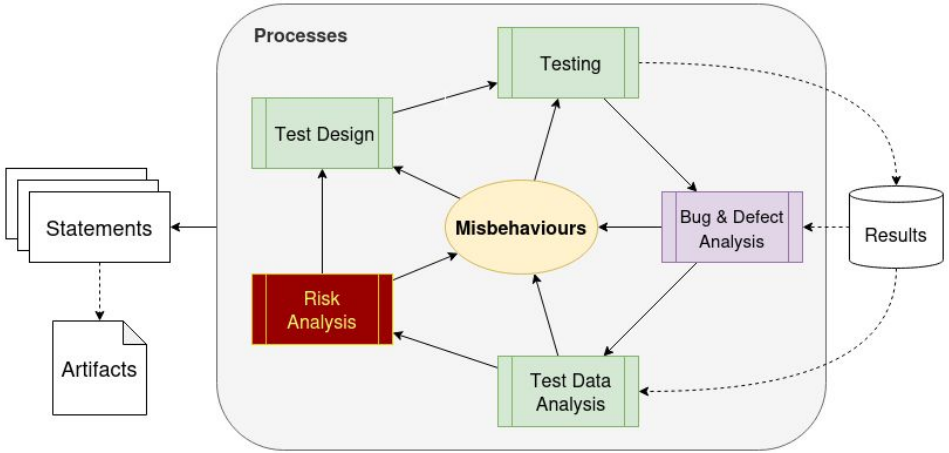
Backup slides

TSF and S-CORE

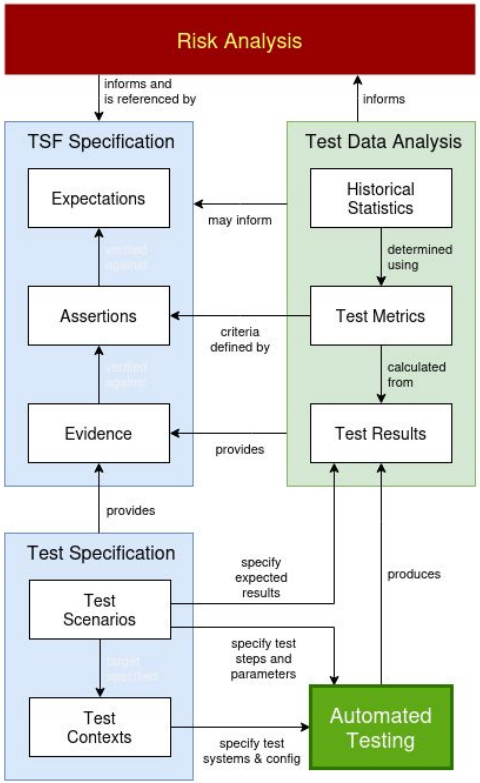
Trustable Software Framework	S-CORE process
Mainly for existing projects, including FLOSS	Mainly for new development projects
For adopters of FLOSS for safety-relevant systems	For S-CORE stack developers + integrators
An argument with measurements, not a process	A standards-compliant safety process
Aiming to be a new fully open standard	Aiming to develop standards-compliant FLOSS
Ongoing safety assessments by exida	Ongoing safety assessments by exida
Tooling is doorstop + mkdocs	Tooling is sphinx-needs + sphinx
May affect EFFSP + badge programme?	

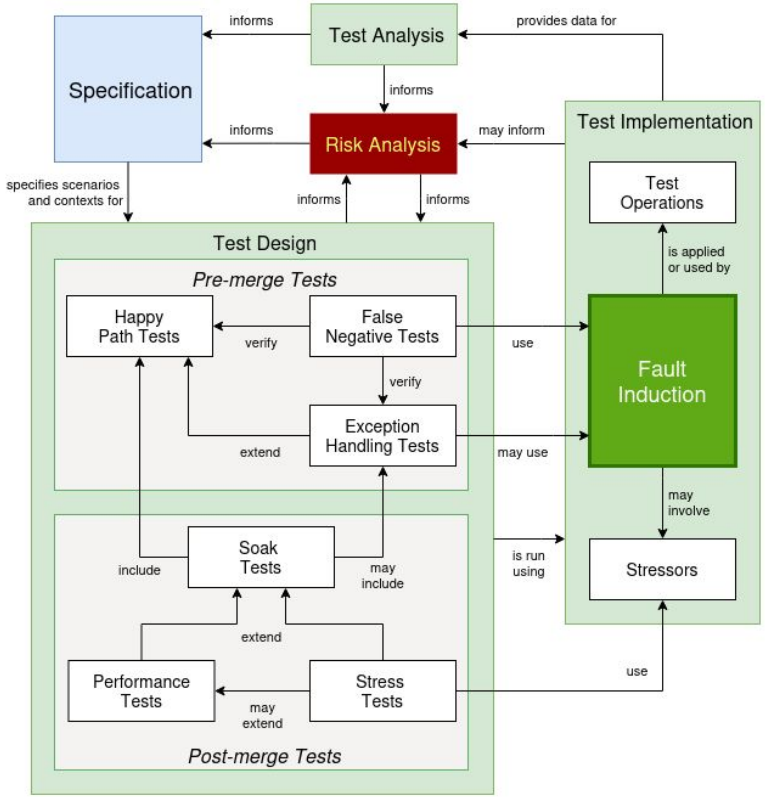
RAFIA: Risk Analysis, Fault Induction and Automation



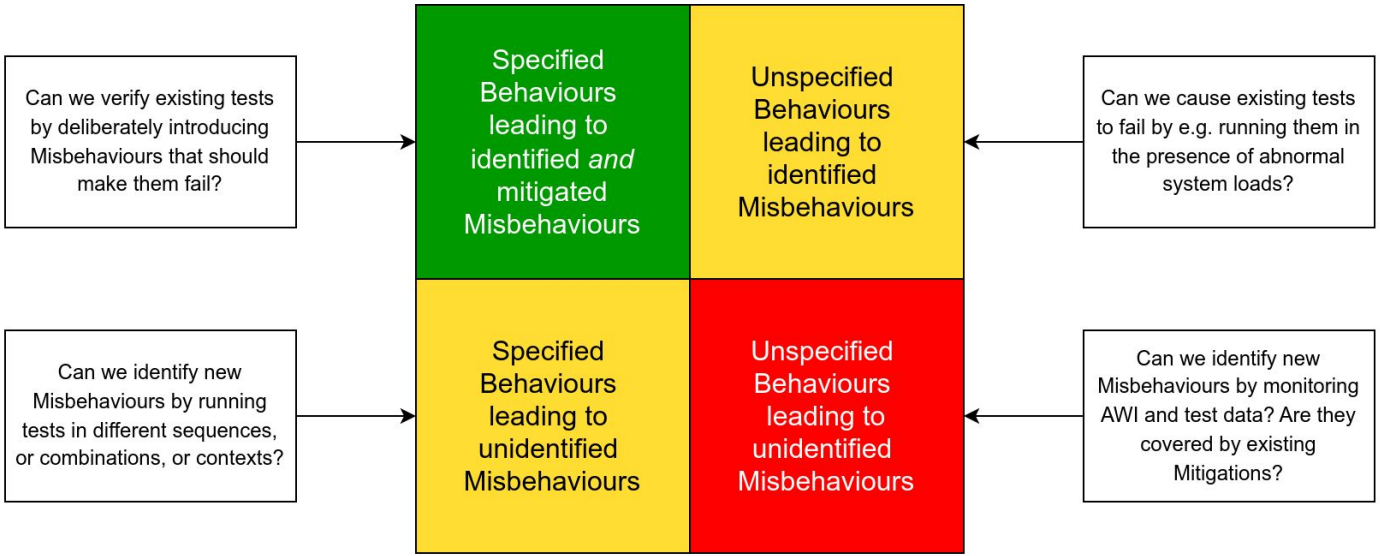


RAFIA: Automated testing





RAFIA: Testing quadrant



Misbehaviours describe ways in which the software may deviate from its its expected **Behaviours**

Identified means that Misbehaviours are predicted by Risk Analysis or observed in test or production

Unidentified Misbehavours may be caused by interference we've not considered or tests we've not specified

Identified Misbehaviours may point to scenarios we've not specified, or inadequate test implementations

Unspecified Behaviours may mean that **Expectations**, **Assertions** or **specification artifacts** need improving

Statistics for each quadrant are used to measure confidence in testing, detection and **Mitigations**