TRUSTWORTHY SOFTWARE

GUIDANCE DOCUMENT

TRUSTWORTHY SOFTWARE ESSENTIALS (TSE)

February 2016
ISSUE 1.2 - TLP WHITE

TS502-1

DOCUMENT CONTROL

CHANGE RECORD

VERSION	DATE	SUMMARY
0.A	24/03/2015	First draft of document, for internal TSI comment
0.B	03/06/2015	Updated with internal and stakeholder comments
0.C	16/06/2015	Updated with internal comments
0.D	25/06/2015	Updated with internal comments
0.E	22/07/2015	Updated with additional Annex & changes to format and naming conventions
0.F	10/11/2015	Updated with internal/stakeholder comments
1.0	02/12/2015	Updated with final corrections
1.1	07/12/2015	Updated with stakeholder comments
1.2	16/02/2016	Updated to include reference to fundamental requirements

DOCUMENT INFORMATION

DOCUMENT ID	TSI/2014/206
PRODUCTION ID	N/A
TSK-ID	TS502-1
PUBLISHER	UK-TSI, IMC, Westwood Heath, CV4 7AL
TRAFFIC LIGHT PROTOCOL (TLP)	WHITE: Open Published Material with no restrictions on distribution. *NB. Please see Glossary for further information*
RIGHTS.COPYRIGHT:	
RIGHTS.CUSTODIAN:	TSI Programme Support Officer (+44 300 030 1928 / enquiries@uk-tsi.org)

FOREWORD

TRUSTWORTHY SOFTWARE INITIATIVE

The Trustworthy Software Initiative (TSI) is part of the UK Government's National Cyber Security Programme to improve the UK's ability to combat cyber risks and ensure that the UK leads the way in trustworthy software systems and expertise.

The objective of TSI is to provide the knowledge, skills and capability for supply, demand and "corpus" (education and research) communities such that trustworthy software can be designed, implemented, sustainably maintained and assured in a risk-based, whole-life process.

TSI works with organisations and individuals in the UK, and international partners, including government, academia, private/public companies, software developers and users, to achieve a recognised level of trust of software by providing targeted education, skills, standards and guidance.

LICENCE CONDITIONS

DISCLAIMER

FURTHER INFORMATION

For further information relating to other aspects of Trustworthy Software, including the more comprehensive Trustworthy Software Framework, please visit: www.uk-tsi.org.

For all associated and/or supporting documents, please see "Annex B: References" of this document.

CONTENTS

1 INTRODUCTION

1.1. PURPOSE AND APPLICABILITY OF DOCUMENT

This document provides guidance on the Trustworthy Software Essential (TSE) controls that should be adopted by organisations in order to ensure the trustworthiness of the software they produce, procure or use.

It is intended as essential reading for those in the organisation responsible for implementing the decision to adopt trustworthy software, and is applicable to organisations of all sizes who either:

- have a basic requirement for trustworthy software, or
- require a baseline Trustworthiness Level (TL) of 1 or 2.

1.2. USING THIS DOCUMENT

This document sets out the baseline requirements for managing trustworthiness during the software life-cycle (including Governance, Responsibilities and the Documentation required), together with the controls comprising Trustworthy Software Essentials.

In addition to these baseline requirements, each section (E1 to E5) further identifies any fundamental control requirements (considered as the normative approach to ensuring software trustworthiness within PAS 754) which are required to be established in addition to the basic controls listed.

The TSE controls are organised under 5 Essential (E) Objectives:

Scope for Use (E1)

Coding Practices (E2)

Use Tools Effectively (E3)

Defect Management (E4)

Artefact Management (E5)

The TSE controls are a subset of the comprehensive Trustworthy Software Framework (TSF) techniques and therefore each have a unique control serial number which cross-references back to the TSF. This is stated at the end of each control in the format of [xx.nn.nn], where xx denotes:

GV Governance controls

RI Risk controls

PE Personnel controls

PH Physical controls

PR Procedural controls

TE Technical controls

CM Compliance controls

It is intended that each control should be implemented in a prescriptive manner, as it is only by full adoption that the majority of the most commonly encountered risks can be mitigated.

If there is a valid business reason as to why a particular control cannot be adopted, then this should be appropriately documented (in accordance with Section 3), and either alternative controls implemented or the risk appropriately treated.

2 BACKGROUND

2.1. WHY TRUSTWORTHY SOFTWARE?

From smart phones to power stations, airliners to e-commerce, our economy and society is increasingly dependent on software in many different guises.

Our daily lives and industrial processes are now heavily reliant on a wide range of underpinning software, whether it be the tools we use to communicate, the methods of transport we use, or the infrastructure that is used to support us in both our professional and personal lives. This makes software trustworthiness an underlying concern for all those who commission, write or use it.

There is, therefore, a pressing need to address the quality and robustness of our software, that is to establish its "trustworthiness" and ultimately to ensure that the software we use performs as it should, when it should and how it should.

This means addressing the trustworthiness of software throughout its life-cycle (see *Figure 1*), from development through to disposal.

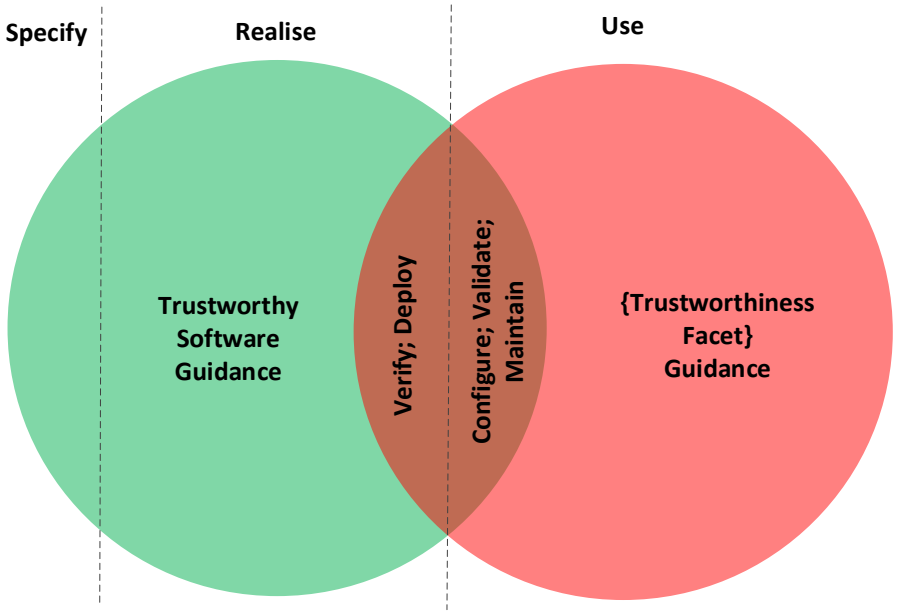

Figure 1

2.2. FACETS OF TRUSTWORTHY SOFTWARE

No software asset can be proven, or even be expected to be completely free of all defects i.e. free from conditions which could cause it to fail or behave in an unexpected manner. However, it should have a level of "trustworthiness" commensurate to the purpose for which it is used.

TSI identifies trustworthiness to predominantly consist of five facets:

Safety, Reliability, Availability, Resilience and Security

2.3. APPROPRIATE LEVEL OF TRUSTWORTHINESS

To ensure that the software achieves an appropriate level of trustworthiness, TSI recommends a risk-based approach to determining the level of trustworthiness required, whereby the reliance on the software to provide trustworthiness (in particular the 5 facets of trustworthiness listed above) is considered together with the purpose of the software and the maximum impact of a defect/deviation in the software.

2.4. TRUSTWORTHY SOFTWARE CONTROLS

TSI has collated two sets of controls which are Trustworthiness Level (TL) dependent and should be applied to software in order to achieve an appropriate level of trustworthiness:

- Baseline Trustworthy Software controls, referred to as the TS Essentials (TSE); and
- Comprehensive Trustworthy Software controls, utilising the full TS Framework (TSF).

The following table (*Table 1*) shows the control set that should be adopted, dependent upon the TL required.

TL	SOFTWARE AUDIENCE	CONTROL SET
TL 0	No requirement for TS	No Requirement
TL 1	Mass Market with Implicit Need (M/I)	TS Essentials (TSE)
TL 2		Baseline TS controls forming a sub-set of the TS Framework (TSF)
TL 3	Mass Market with Explicit Need (M/E)	TS Framework (TSF)
TL 4	Niche with Explicit Need (N/E)	Comprehensive TS controls utilising the full TS Framework (TSF)

Table 1

3 MANAGING TRUSTWORTHINESS

3.1. GOVERNANCE

Before starting the process of producing or procuring software, you should take time to establish an effective method of governance which encompasses the policies and mechanisms required to ensure that an appropriate level of trustworthiness is achieved and continues to be maintained throughout the software life-cycle.

This should be documented as part of a Trustworthy Software Management System (TSMS) and should explicitly cover the following areas:

- Roles, Responsibilities and Accountabilities – this must include the role of Trustworthy Software Release Authority (TSRA, as defined below) together with relevant roles both internally to the organisation and externally such as Stakeholders and Third Parties;
- Communications plans;
- Risk Management – documenting the level of Trustworthiness (TL) required for the software asset;
- Trustworthy Software Defect & Deviation List (TSDDL) Management process – documenting the process to be taken for the notification, recording and remediation of defects and deviations.
- Trustworthy Software Defect & Deviation List (TSDDL) Deferral process – documenting the process to be taken for defects and deviations that are not resolved and this should include the way in which the resultant level of risk and any subsequent impacts on the organisation or system are to be managed.
- Trustworthy Software Controls – documenting those controls relevant to the organisation in accordance with the level of Trustworthiness required, together with the state of implementation of each control;
- Monitoring and Compliance regime – to ensure that risk and control decisions are current, have been implemented and are continually maintained (for example using the Plan-Do-Check-Act (PDCA) cycle) and should include the regular checking of the TSDDL for software issues, mitigation progress and efficacy of remediation;
- A formal assurance process to assess the trustworthiness of the software, together with appropriate sign-off (Trustworthy Software Release Note) permitting software release;
- Documentation required to be produced in order to support the TSMS;
- TSMS maintenance – guidance on the frequency of TSMS reviews to ensure that they are still relevant to the organisation (for example, a yearly review performed for organisations who have a basic requirement for Trustworthy Software or TL1 and six-monthly reviews performed for a TL2 requirement).

Please note that the TSMS does not need to be a separate document provided that the areas listed above are satisfactorily encapsulated with existing management systems (for example an ISMS or QAMS) and/or certification schemes (for example, TickITplus).

3.2. ROLES & RESPONSIBILITIES

The roles identified below may be encompassed into an existing role insofar as their responsibilities specifically include those indicated below (as appropriate).

THE BOARD

The organisation's Board should be responsible for the overall governance of the provision of Trustworthy Software and, in particular, should be fully cognisant of the organisation's key information assets and the organisational impact in the event of compromise, loss or unavailability.

TRUSTWORTHY SOFTWARE RELEASE AUTHORITY (TSRA)

The TSRA should have day-to-day responsibility for the management of Trustworthy Software and should be empowered by the Board with sufficient authority to fulfil this role. In particular, the TSRA should be responsible for:

- Ensuring that all elements of the software are subject to Configuration Management (CM) during specification, development and release;
- Ensuring that key information assets are specifically taken into consideration, with appropriate design and technical controls implemented;
- Ensuring that sufficient consideration has been given to all relevant Trustworthy Software concepts, principles and techniques;
- The maintenance of the Trustworthy Software Defect and Deviation List (TSDDL);
- Production of the Trustworthy Software Constraint and Dependency model (TSCDM) document, detailing all external constraints and dependencies involved in software deployment, configuration and operation;
- Establishing a product release and acceptance/commissioning process, which the TSRA is responsible for undergoing before completing and signing-off the Trustworthy Software Release Note (TSRN).

3.3. DOCUMENTATION

TRUSTWORTHY SOFTWARE DEFECT AND DEVIATION LIST (TSDDL)

The TSDDL should be used to record all defects and deviations both during Realisation (R-TSDDL) and In-service (I-TSDDL), to the following level of detail:

- Date and Time the Defect/Deviation was found.
- Title or Name of Defect, such that it can be easily referenced (this is to ensure that the same defect is not raised more than once).
- Unique identifier in order that it can be tracked through to remediation.
- Communication, this should list the name and contact details of the person who discovered the defect/deviation.
- Description of Defect/Deviation, including the location and effect of the defect/deviation and screenshots where appropriate.

- Configuration, which describes the environment in which it occurred (e.g. platform, hardware and software).
- Steps to Reproduce Defect/Deviation, this should clearly describe how the defect/deviation can be reproduced.
- Root Cause Analysis - if possible (or appropriate) a reason for the occurrence of the defect/deviation should be stated.
- Classification of Defect/Deviation.
- Risk Evaluation and Severity rating of Defect/Deviation.
- Priority for remediation.
- Action to be taken (including short-term mitigation measures if necessary).
- Resource allocated for remediation.
- Status of Defect/Deviation.
- Closed, where applicable, this should give the date that the defect/deviation was fixed together with the name of person responsible.

Please note that the TSDDL does not need to be a separate document, provided that the areas listed above are satisfactorily included within other documents.

TRUSTWORTHY SOFTWARE RELEASE NOTICE (TSRN)

The Trustworthy Software Release Notice (TSRN) is intended for those individuals responsible for deploying, configuring and operating the software item, and should be completed and signed off by the TSRA before deployment.

The TSRN should explicitly detail (or reference) the following:

- The full list of controls as specified within TS Essentials, together with any other relevant controls that also support the facets of trustworthiness. For each identified control, the following should be documented:
  - Relevance of control;
  - Action taken to implement the control;
  - If the control has not been implemented, the reasons for this should be given together with any other relevant information.
- A full list of Constraints detailing all mandatory requirements that the system has to satisfy, for example regulatory requirements or communications / hardware / software interfaces.
- A full list of Assumptions detailing any assumptions/predications that have been used as a basis for defining software requirements, for example future capacity requirements.
- A full list of Dependencies detailing any systems or external functionality that the software is dependent upon, for example the use of third-party libraries or dependencies.
- A full list of all unmitigated defects and deviations (taken from the R-TSDDL), with the following documented for each:
  - Reasons as to why the defect/deviation has not been fully mitigated;
  - Any short-term measures that have been put in place;
  - Whether it is intended that the defect/deviation be mitigated together with a timescale where appropriate.

Please note that the TSRN does not need to be a separate document, provided that the areas listed above are satisfactorily included within other documents.

TRUSTWORTHY SOFTWARE CONSTRAINT AND DEPENDENCY MODEL (TSCDM)

The Trustworthy Software Constraint and Dependency Model (TSCDM) should document the way in which all elements of the software asset are intended to be deployed, configured and operated.

In particular, the following should be recorded:

- external constraints, such as communications/software/hardware interfaces;
- external dependencies, such as customers and supply chain; and
- security considerations, such as cryptographic requirements for assurance and privacy.

Please note that the TSCDM does not need to be a separate document, provided that the areas listed above are satisfactorily included within other documents.

4 SCOPE FOR USE (E1)

4.1. OBJECTIVES

The objectives of 'Scope for Use' are to fully understand the requirements of the software you are producing or procuring, in particular: the environment in which it will operate, the functionality it will have and the requirements it needs to fulfil.

4.2. INTRODUCTION

Software, either standalone or for integration into legacy systems and/or external systems can be inherently complex, and therefore difficult to establish trustworthiness.

However, by taking the time to understand and document what you need from the software you are producing/procuring, you can ensure that an informed evaluation is made against your requirements and, therefore, an appropriate level of trustworthiness achieved.

By understanding what you need from the software, you can help to:

- reduce the occurrence of project failure by ensuring that the software will meet the requirements of your users, stakeholders, and customers/suppliers;
- determine a realistic cost of the project – estimates can be provided against the identified software requirements, thereby reducing the occurrence of costs associated with misaligned requirements, such as change requests;
- reduce the amount of development effort required – by ensuring that the software requirements have been unambiguously defined with no omissions or inconsistencies, this decreases the possibility that time will need to be spent on the redesign, recoding and/or retesting of the software;
- determine a realistic time-scale for the project, thereby ensuring that adequate time is allocated for software testing; and
- provide a baseline against which the software can be tested (both for verification and validation), such that compliance against regulatory, legal and business constraints can be achieved.

4.3. FUNDAMENTAL CONTROLS

The following control measures are required to be established (or demonstrated) to meet the TSE objective 'Scope for Use' (E1):

- TSMS (section 3.1)
- TSRA (section 3.2)
- TSCDM (section 3.3)

4.4. BASIC CONTROLS

The controls listed below meet the TSE objective 'Scope for Use' (E1) and each should be implemented in a prescriptive manner as required by the TSMS. Where this is not practicable, this should be documented (in accordance with Section 3), and either alternative controls implemented or the risk arising from non-adoption appropriately treated.

UNDERSTAND REQUIREMENTS [PR.03]:

- **Specify Explicit / Functional Requirements [PR.03.10]**

  You should define what the software is supposed to do, as in its specific behaviour and/or functions. For example: The system will be subject to full weekly backups and daily incremental backups. All requirements should be recorded in the Software Requirement's Specification (SRS) and/or TSCDM as appropriate.

- **Specify Implicit / Non-functional Requirements [PR.03.20]**

  You should define the way in which the system operates, to include specific requirements relating to Safety, Reliability, Availability, Resilience, Security, Usability and Performance. For example: The system shall be protected from the loss of information assets in the event of accidental deletion, corruption, system failure or natural disaster. All requirements should be recorded in the SRS and/or TSCDM as appropriate.

- **Understand Implicit / Non-objective Requirements [PR.03.30]**

  You should define the business requirements of the software and any constraints imposed on the software, either by the business or by other systems. All requirements should be recorded in the SRS and/or TSCDM as appropriate.

- **Understand Use Cases [PR.03.40]**

  You should understand how the user will interact with the software in order to achieve specific goals and focus on what the software is required to do rather than how it will do it. The user in this instance can be a human and/or an external system and may be time dependent. All requirements should be recorded in the SRS and/or TSCDM as appropriate.

- **Monitor and record Derived Requirements [PR.03.50]**

  You should document all additional requirements that have been identified as a result of realisation activities (and have not already been defined as part of PR.03.10, PR.03.20 and/or PR.03.30) in the TSCDM as by definition, they should not be recorded in the SRS.

SEEK TRUSTWORTHY REALISATION [TE.05]:

- **Implement only the minimum set *of features* of users to meet requirements [TE.05.80]**

  You should ensure that the functionality of the system is sufficient to meet the requirements of the user, without containing extraneous features. The addition of unessential functionality can cause needless over-complication of the software, leading to project delay, extra cost and increased complexity when ensuring the trustworthiness of the software. Consideration should also be given to any derived requirements resulting from:

  - business and stakeholder requirements and constraints; and
  - the proposed design, for example: implementation language, resource limits, operating environment, selected architecture.

5 CODING APPROACH (E2)

5.1. OBJECTIVES

The objectives of 'Coding Approach' are to ensure that the software you produce or procure is structurally sound, such that not only does it perform 'as expected', 'when expected' and 'how expected', it is also facilitates maintenance, debugging and enhancement.

5.2. INTRODUCTION

Billions of lines of code are produced each year, forming the software that underpins our daily lives and that we rely on to be trustworthy, in particular, by maintaining the security (Confidentiality, Integrity, Availability and Accountability) of the information that we process. Unfortunately, the security of our software is not a static field and requires constant monitoring to ensure that it is protected from rapidly evolving threats and vulnerabilities.

One of the ways to help improve the trustworthiness of our software is to build security into the whole software life-cycle, in particular during development where the use of secure coding practices can help to reduce the possibility of vulnerabilities being introduced into the software.

The majority of common vulnerabilities occur as a result of unsafe, unreliable and insecure methods of coding, where simple coding errors could result in weaknesses that an attacker can exploit (for example: Buffer Overflows, Unvalidated inputs, SQL injections, Cross-site scripting, Missing encryption of sensitive data).

It is therefore important that you take time to ensure that all coding is performed in a secure manner, not only to improve the quality of the code being produced and ensure the code is able to withstand unforeseen circumstances but also to improve the trustworthiness of your software. This may be achieved by:

- Ensuring that the code is free from all known errors, warnings and vulnerabilities;
- Following best practice in accordance with the programming language chosen to ensure that technical vulnerabilities are not introduced into the software;
- Improving the readability of code to ensure that it is easy to debug and maintain;
- Improving the predictability of the code, to ensure the consistency of outcome as a result of failure modes, unexpected inputs or user action;
- Considering each functional feature for safety, weaknesses and abuse and coding to mitigate those threats.

5.3. FUNDAMENTAL CONTROLS

The following control measures are required to be established (or demonstrated) to meet the TSE objective 'Coding Approach' (E2):

- TSMS (section 3.1)
- TSDDL (section 3.3)

5.4. BASIC CONTROLS

The controls listed below meet the TSE objective 'Coding Approach' (E2) and each should be implemented in a prescriptive manner as required by the TSMS. Where this is not practicable, this should be documented (in accordance with Section 3), and either alternative controls implemented or the risk arising from non-adoption appropriately treated.

MAKE APPROPRIATE TOOL CHOICES [TE.02]:

- **Produce and maintain Coding Standards [TE.02.20]**

  You should ensure that all code developed for the software is written in accordance with a coding standard. A coding standard consists of a set of a formalised guidelines for a particular programming language that is intended to ensure consistency and improve the structural quality of the software being produced. The standard will generally cover aspects of style, practices and methods, such as indentation, comments, and naming conventions.

  Because the use of a coding standard is not enforced by the production tool, its use will not affect the operation of the executable program, however, it will improve the readability of the code and help to ensure maintainability. This is especially important if, as is often the case, the original author(s) of the code will not be responsible for long-term software maintenance and/or enhancement.

FOLLOW STRUCTURED DESIGN [TE.03]

- **Use only proven components [TE.03.30]**

  When re-using components and libraries, including open source, all externally sourced elements should:

  - have a trustworthy software provenance wherever possible;
  - be proven in use within a similar environment and the provision of similar functionality; and
  - be subject to adequate support and maintenance.
  - be subject to a Vulnerability Review (VR) for known vulnerabilities (CVEs) both before selection and again before deployment. For example, by using an SCAP validated product or a manual process of consideration against a CVE master list, such as the National Vulnerability Database (NVD).

FOLLOW STRUCTURED IMPLEMENTATION [TE.04]

- **Produce bespoke components in accordance with Coding Standards [TE.04.10]**

  If you are procuring bespoke software (i.e. software that is specifically being developed by a third party in accordance with your requirements), you should ensure that it is produced in accordance with a coding standard, either one specifically used by your organisation or an existing mainstream standard.

- **Use appropriate and recognised data formats [TE.04.20]**

  All data is encoded (for storage) in a format that a program or application can recognise, read and use. In order to ensure that your data remains in a readable state and is unaffected by obsolete(d) software, you should consider saving the data in a format that is independent of proprietary software. This will include standard formats, exchangeable formats and/or open formats, for example: ISO 8601 which pertains to the representation and format of dates.

- **Select appropriate algorithms [TE.04.30]**

  Algorithms essentially define the way in which a specific task is achieved by following an explicit set of steps. This means that more than one algorithm may exist that each use different methods (or steps) to produce the same end result. For example, when sorting a list, a number of algorithms may be suitable, including: Bin sort; Merge sort; Bubble sort; Shell sort; Quick sort.

  When choosing an algorithm to use, you should consider their strengths and weaknesses in conjunction with:

  - the speed required;
  - the amount of current (and future) data to be handled;
  - the complexity of data to be handled;
  - the granularity of data to be handled;
  - The efficiency of the algorithm (i.e. the amount of resources that it uses) in light of:
    - the way the algorithm is coded;
    - the programming language used;
    - the production tool(s) and options used; and
    - the Operating System used.

SEEK TRUSTWORTHY REALISATION [TE.05]

- **Review Design for failure modes [TE.05.10]**

  At a minimum (or in addition to design and code reviews) you should conduct a formal review of the design for failure modes, concentrating on high-risk areas. This well help you to identify:

  - single-points of failure;
  - causes of the failure, for example, whether they are data related or event related;
  - Severity of the failure, whether to the system or external factors such as to individuals, property, cost to remediate;
  - any requirements missing from the software specification;
  - any unwritten assumptions; and
  - any areas that specifically require fault-handling.

  All identified failure modes should be recorded in the R-TSDDL against a unique identifier.

- **Source and configure off the shelf components in accordance with Design/Effect Pattern(s) [TE.05.20]**

  A ‘design pattern’ is a formal description (or strategy) depicting how to solve a problem which focuses on the interactions between objects without explicitly defining them. By using design patterns across repeatable problems, you can ensure:

  - uniform design of code;
  - that best-practice is followed; and
  - that development time is reduced.

- **Ensure mitigations are used for all identified failure modes [TE.05.45]**

  For each failure mode identified as a result of [TE.05.10] above, you should:

  - Mitigate against the failure by:
    - changing the design in order to eliminate or reduce the cause of failure;
    - changing the design or applying a countermeasure(s), in order to lower or eliminate the failure mode; and/or

- providing a mechanism such that the fault mode can be detected in a timely manner and manual corrective action can be taken to either prevent or mitigate against the consequences of the failure.

- Accept the failure mode and justify the resulting level of risk.

- **Implement measures to control malicious code [TE.05.65]**

  All externally sourced components, as described in [TE.03.30], will need to be reviewed for the presence of malicious software (such as viruses, worms, trojans, adware, spyware, scareware) prior to use by an up-to-date MalWare scanner that has been publicly reviewed as offering good performance at detecting exploits 'in-the-wild', such as against the WildList

MINIMISE RISK EXPOSURE [TE.06]

- **Only grant minimum Privileges required, with all other actions defaulting to not permitted [TE.06.10]**

  You should ensure that the software is **only** provided with privileges that are essential to its successful function. This will help to:

  - protect data from loss or compromise as a result of a system fault and/or security breach;
  - limit the damage to the system in the event of a system fault and/or security breach;
  - ensure that the code does not have sufficient rights that may adversely affect the rest of the system or interfaced applications;
  - ensure that vulnerabilities in the code (either inherent or maliciously injected) are not able to affect the rest of the system or interfaced applications; and
  - ensure that unintentional, unwanted or improper use of privilege is less likely to occur.

- **Separate program data, executables, and configuration data [TE.06.20]**

  In order to provide defence-in-depth and ensure that the principles of least privilege and need-to-know can be enforced, you should segregate program elements such as program data, executables and configuration files.

PRACTICE HYGIENIC CODING [TE.07]

- **Ensure all variables, pointers and references are properly initialised at first and subsequent uses [TE.07.05]**

  You should ensure that all variables, pointers and references are explicitly initialised before use, that is, they should be assigned an initial value. Although some languages automatically assign values to variables (zero) when declared, this is not true of all languages. Failure to assign known values could result in:

  - the program behaving unpredictably;
  - fault modes being unreproducible;
  - segmentation faults; or
  - run-time errors.

- **Ensure all Input Data is Validated [TE.07.10]**

  In order to help protect applications from vulnerabilities (for example: buffer overflows, code injection, directory traversal etc.), you need to ensure that all data input into the system is subject to validation checks. This is to make certain the data is fit-for-purpose, consistent, meaningful, correct and useful. Validation checks normally check that the data input into the system:

  - Is the correct length

- Only contains permitted characters
- Is in the correct format
- Is within the correct range for numerical data.

- **Ensure all Messages are Validated [TE.07.15]**

  It is good practice to use a common set of messages internally within the software, for example internal results and error messages, which are unambiguous and have been validated prior to use.

- **Ensure implementations of all Algorithms are Validated [TE.07.20]**

  It is good practice to use algorithms of known provenance wherever practicable, as this will ensure the effectiveness of the algorithms before use. For example, for the protection of sensitive data, you should consider using one of the validated algorithm lists maintained by bodies such as the European Telecommunications Standards Institute (ETSI) or National Institute of Standards and Technology (NIST) for each cryptographic standard testing program.

- **Ensure all Output Data is Validated [TE.07.25]**

  In order to help protect against accidental leak of information or malicious attacks such as cross-site scripting (where a recipient is sent malicious code from a legitimate site) or SQL injection, all output from your system should be validated before it is sent to a recipient. Output validation checks should check for:

  - appropriate data encoding;
  - proper formatting;
  - the length of the output, to ensure that it is not longer than expected;
  - potentially harmful content;
  - unauthorised active content;
  - whether the actual output matches the expected output;
  - any extra characters; and
  - whether the output needs to be sanitised.

- **Ensure Error Handling is implemented comprehensively, and "fails safe and secure" [TE.07.30]**

  To ensure that error handling "fails safe", you need to make certain that all failure modes, exceptions or errors do not leave the software in an insecure state, thereby causing harm to either the system itself or endangering lives. This can be achieved by ensuring that all resources are locked down and released, sessions terminated and that automatic re-routing to a backup system occurs.

  Making sure that error handling "fails secure" is vital to preventing accidental information leakage and preventing the inadvertent disclosure of the way in which the software works to end users or potential attackers. This can be achieved by making certain that all code is written to handle all failure modes, exceptions and errors (whether expected or unexpected) in a secure and predictable manner, in particular:

  - Only generic messages are sent to end-users;
  - All connections to internal/external systems are closed;
  - The system reverts to low privilege where possible;
  - All sensitive files are closed; and
  - The error is logged.

- **Apply consistent naming convention [TE.07.35]**

  The use of intelligible naming conventions is important because it helps you to write consistent, clear and well-commented code that is easy to read, understand, and therefore, maintain. It consists of a set of rules that dictate how entities in source code should be denoted, for example:

  - whether variables are capitalised;
  - whether prefixes are used for pointers,
  - how private fields are indicated;
  - the length of names;
  - the use of abbreviations etc.

- **Manage resource access explicitly (buffers, stacks, memory, cache and files) [TE.07.40]**

  All access to resources should be explicitly managed by ensuring that:

  - each resource is explicitly released (at all exit points and for all error conditions) once a process has finished using it;
  - the lowest level of privilege is assigned;
  - resources cannot be used by unauthorised processes/individuals;
  - resources are by default read-only unless specifically required to be modified by accessing process/user.

  This to help prevent resource leaks (which could affect performance or cause instability), the unauthorised access of information, inadvertent privilege escalation, improper error handling and insufficient resource tracking.

- **Explicitly remove detritus (temporary files/logs) [TE.07.50]**

  Temporary files may be used by software in order to perform tasks such as sharing data between processes, storing program data, construct/load classes etc. It is important that you ensure the software explicitly deletes these temporary files on termination of the program, this is not only to enable the re-use of file names and secondary storage, but also to mitigate against the occurrence of:

  - File-based attacks;
  - TOCTOU (Time-of-Check, Time-of-Use) race conditions;
  - Unauthorised elevation of privileges;
  - Unauthorised access to sensitive information; and
  - Denial-of-Service attacks.

  If the removal of detritus cannot be guaranteed (for example through an exception or error) then you should ensure, at the very least, that temporary files are: given unique fully randomised names and that alternate files with the same name as the temporary file cannot be created; and/or stored in a secure area that is not shared or publically accessible.

6 USE TOOLS EFFECTIVELY (E3)

6.1. OBJECTIVES

The objectives of 'Use Tools Effectively' are to use software tools in a consistent and correct manner in order to improve the trustworthiness of the software you use, produce or procure.

6.2. INTRODUCTION

Software tools, when used properly, can help increase the efficacy of the development and management of software throughout its life-cycle from Software Specification (pre-release) to Software Maintenance (in-service).

To ensure that your chosen software tools aid rather than hinder the trustworthiness of the software you are using/producing/procuring, you should take the following into consideration when choosing each tool:

- Functionality: Does the selected tool meet your specific requirements and perform all the required functions to an acceptable level.
- Ease of use: Is it easy to learn/operate/navigate and is it usable by non-programmers and end-users? Furthermore, are all the necessary individuals committed to its use?
- Support and Configuration: What is the warranty, maintenance and upgrade policy for the tool and is adequate support provided (e.g. via email, phone, consultancy, or user groups).
- Cost: Is the tool open source or commercial? What is the licensing used and is it consistent within the estimated price range and consistent with comparable products. Are the extra cost-implications associated with training or support?
- Portability: Does the tool support all of the operating systems, applications, platforms, and/or development tools either currently in use or planned for future use.

6.3. FUNDAMENTAL CONTROLS

The following control measures are required to be established (or demonstrated) to meet the TSE objective 'Use Tools Effectively' (E3):

- TSMS (section 3.1)
- TSDDL (section 3.3)

6.4. BASIC CONTROLS

The controls listed below meet the TSE objective 'Use Tools Effectively' (E3) and each should be implemented in a prescriptive manner as required by the TSMS. Where this is not practicable, this should be documented (in accordance with Section 3), and either alternative controls implemented or the risk arising from non-adoption appropriately treated.

MAKE APPROPRIATE TOOL CHOICES [TE.02]

- **Selection of appropriate Programming Language(s), considering known vulnerabilities and needs for Typing [TE.02.10]**

  Choosing a programming language that is appropriate for the software you are creating or procuring can affect overall development time, ease of maintenance, life-cycle costs and the overall trustworthiness of the finished product. Because of this, you should ensure that the following aspects are taken into consideration:

  - Suitability for the project:
    - Has the language previously been used (and therefore considered to be tested and proven) within a similar domain?
    - Can it cope with the algorithmic complexity required?
    - Does it have built-in support (language-level and implementation) for what you are trying to code?
    - Does it support concurrency?
    - Is the programming language compatible with the Operating Systems, Programs and Applications that it will connect to, and do the proposed connections place any other constraints on the choice of language?
  - Reliability: How does the language cope with abnormal conditions, such as failure modes and errors, and is it more susceptible to failure or crashes as a result of them.
  - Support & Configuration: Does the programming language have readily available tools and resources to support both development and implementation, and what format are they available in (e.g. hardcopy and online)
  - Features of the language:
    - Level of Language and Paradigm: Does the level and style of the language meet the requirements of the system;
    - Production tool: Is the translated code suitable for the target platform and how is error detection and enforcement of correct code handled?
    - Coding style: Although 'ease of coding' is desirable during development, it is important to remember that code is generally only written once but will need to be read numerous times for bug fixing/maintenance/enhancement, therefore 'ease of reading' is also equally important.
    - Type system (Typed or Untyped) of the language: This refers to the rules that assign types to constructs such as variables and includes the following characteristics: the strength of the type (strong or weak); type checking (static: at compilation or dynamic: at run-time); type safety (safe or unsafe); and type expression (manifest or inferred).
  - Security: Although the occurrence of vulnerabilities are not necessarily dependent upon the programming language choice, the way in which some programming languages are designed mean that they are more susceptible to certain vulnerabilities. Because of this you should take time to understand the security concerns inherent in the language in order to reduce the occurrence of vulnerabilities.

- **Selection of appropriate Testing Tools [TE.02.40]**

  In order to ensure that the software being produced both meets your requirements and acts in a trustworthy manner, you need to test continually during the development stages and again before live production. This is to ensure that weaknesses, errors and flaws are captured and resolved at each stage, thereby avoiding the culmination of errors which may affect the time-scale of the project or create/become vulnerabilities in the live system.

The use of automated (or semi-automated) testing tools, in conjunction with manual testing, can be a cost and time-effective way of ensuring that the necessary testing is performed throughout the development and life-cycle of the software. There are numerous testing tools available, for example, Susceptibility Review Scanners (SR-S) and Code Analysers (Binary – BiCA or Source – SoCA), some of which are free and/or Open-Source (FOSS). At a minimum, the following areas should be taken into consideration when choosing each tool:

- Management: Does the tool have the capability to manage the testing process and produce meaningful reports (including the ability to export reports where appropriate)?
- Functionality:
  - Is the tool suitable for the types of testing you wish to conduct (e.g. functional testing (including testing of security features), regression testing, load/stress/performance testing; robustness testing; penetration/susceptibility testing; risk assessment and risk-based security testing or code analysis)?
  - Does it support distributed testing or synchronising tests across networks?
  - Are there options for creating automated (including custom) tests?
  - Does it perform automatic comparison of test results against expected results (including objects, pictures and files)?
  - Does it automatically log the tests performed together with associated meaningful results?
  - Does it support multi-language applications (where appropriate)?
  - Does it support test component reusability?
- Security: In addition to the functionality given above, for specific Security Testing tools, you should also consider the following:
  - Does the tool support application security testing such as: input checking and validation; brute-force password attacks; SQL insertion; fault injection (fuzzing); session management issues; cross-site scripting; buffer overflow vulnerabilities; or directory traversal.
  - Does it have the ability to test the output data created by an application, for example to test cryptography;
  - Does it have the ability to monitor program behaviour, for example, to see how the program would respond to failure modes, or to check for anomalous behaviour which may indicate the presence of a vulnerability;
  - How updates to vulnerability databases are controlled and at what frequency?

USE METHODOLOGICAL PRODUCTION [TE.08]

- **ENABLE and use production tool checking features [TE.08.30]**

  Enable production tool checking features: The job of the production tool is to convert the source code (written in a programming language) into instructions (object code) that the machine understands, and as part of this may perform the following: Pre-processing; Lexical analysis; Syntax analysis/Parsing; Semantic analysis (syntax-directed translation); Code generation; Code optimization; Error Handling; Exception Handling; Security Checks.

  It is important that you use all of the production tool features, in particular:

  - Error Handling: identifies syntactic or semantic errors which prevent the source code from being translated until the identified error is fixed;
  - Error Warnings: this does not prevent the source code from being translated but provides notification of minor errors or areas of code where potential errors may exist;
  - Exception Handling: this does not prevent the source code from being translated but allows unexpected errors to be handled without causing the software to crash (note that once an exception is found, code must be debugged and the error fixed); and

- Security Checks: such as Buffer Security Checks, which although would not prevent the source code from being translated, is considered to be a security vulnerability and therefore should be fixed prior to live production.

7 DEFECT MANAGEMENT (E4)

7.1. OBJECTIVES

The objectives of 'Defect Management' are to ensure that the software is free from (or a course of action has been decided for) all identified defects and deviations, thereby ensuring that the software is trustworthy and operates in accordance with all specified requirements.

7.2. INTRODUCTION

The occurrence of defects or deviations in software can result in unexpected outputs or the software not performing in the manner expected, i.e. not behaving in a trustworthy manner.

The process of finding, recording, evaluating and fixing defects and deviations in software (Defect Management) should be performed throughout the software lifecycle, with the majority of defect/deviations being found and fixed during the development stage (i.e. pre-release). By ensuring that comprehensive defect/deviation remediation occurs earlier in the life-cycle, you can help to:

- Keep the amount of developer time taken to fix the defect/deviations to a minimum, due to familiarity with the code and design of the software.
- Eliminate more than one defect at a time, as some defects may be caused by, or be reliant upon, the occurrence of other defects.
- Reduce the possibility of code needing to be re-written when a system is in-service, which is neither time nor cost effective.
- Reduce the possibility of a lack of goodwill from users in the event of (fixable) pre-release defects/deviations causing problems in-service.

Although Defect Management can be a time-consuming and costly business, by making sure that all identified defects and deviations are correctly evaluated and fixed either pre-release or in-service (through the issuing of updates), you can help to ensure the software continually operates in a trustworthy manner.

7.3. FUNDAMENTAL CONTROLS

The following control measures are required to be established (or demonstrated) to meet the TSE objective 'Defect Management' (E4):

- TSMS (section 3.1)
- TSDDL (section 3.3)

7.4. BASIC CONTROLS

The controls listed below meet the TSE objective 'Defect Management' (E4) and each should be implemented in a prescriptive manner as required by the TSMS. Where this is not practicable, this should be documented (in accordance with Section 3), and either alternative controls implemented or the risk arising from non-adoption appropriately treated.

MAINTAIN DEFECT MANAGEMENT [PR.07]

- **Ensure all Defects identified during Realisation are recorded, reported and assessed, with rectification at earliest opportunity using process for monitoring through a Realisation Trustworthy Software Defect and Deviation List (R-TSDDL) [PR.07.10]**

  One of the main characteristics of trustworthy software is that it performs as it should, when it should and how it should. You can help achieve this through comprehensive defect and deviation management, thereby ensuring that all defects and deviations are appropriately captured (and subsequently evaluated, prioritised and fixed) throughout the software development process. This is especially important for large/complex systems or when defects need to be tracked over long periods of time.

  All Defects and Deviations should be recorded in the 'Realisation Trustworthy Software Defect and Deviation List (R-TSDDL)' (or an alternate internal document that fulfils the same function to the same level of detail, as described in Section 3.3 of this document). in particular, during integration testing, the following outputs should also be recorded:

  - Vulnerability Scanner and/or the VR;
  - Malware Scanner;
  - Susceptibility Review Scanner (SR-S); and/or
  - Code Analyser, either Binary (BiCA) or Source (SoCA).

  You should ensure that all identified defects/deviations are assessed, prioritised and remediated at the earliest opportunity using the TSDDL Management process, and where appropriate the TSDDL Deferral process (as defined in the TSMS), with the actions appropriately recorded in the R-TSDDL.

- **Ensure all In Service Defects and Deviations are recorded in an In-service Trustworthy Software Defect and Deviation List (I-TSDDL), reported and assessed, with rectification at earliest opportunity using process for monitoring of deferrals [PR.07.20]**

  In order to ensure the trustworthiness of the software throughout its life-cycle, you need to continue to ensure that all defects and deviations continue to be captured in the 'In-service Trustworthy Software Defect and Deviation List (I-TSDDL)' or an alternate internal document that fulfils the same function. The level of detail to be recorded should be the same as described in Section 3.3 of this document.

  You should ensure that all identified defects/deviations are assessed with rectification at the earliest opportunity using the TSDDL Management process, and where appropriate the TSDDL Deferral process (as defined in the TSMS).

ENABLE DEPENDABLE DEPLOYMENT [TE.11]

- **Updating and Patching of implemented software, with routine, critical and emergency options, taking due cognisance of R-TSDDL / I-TSDDL [TE.11.60]**

  Software generally requires to be updated in order to for it to be:

  - Fixed: for example, to correct or mitigate against security vulnerabilities, bugs and errors;
  - Improved: for example, to enhance the usability, features or performance of the software; or
  - Changed: for example, to remove or disable components.

  In order to help ensure that vulnerabilities in the software are not exploited and that the trustworthiness of your software is maintained, you need to apply patches and updates within an appropriate time-frame, and in accordance with the severity of the patch/update being issued. Where appropriate, the use of the Common Weakness Scoring System (CWSS) and Common Vulnerability Scoring System (CVSS) can aid in prioritising the remediation of software weaknesses and vulnerabilities.

Patch management procedures should therefore be established and documented, in accordance with TS Patching Guidance [AD.05], to facilitate the application of patches and updates on both a normal basis (scheduled to occur within a regular time-frame) and a critical/emergency basis (scheduled in response to the release of a critical patch/update or emergence of exploits). The following processes should be covered:

- Identifying vulnerabilities in the software (using the R-TSDDL and I-TSDDL);
- Evaluating available patches/updates;
- Evaluating patch priority (based upon vendor criticality, exploits and identified vulnerabilities in the software);
- Testing patches/updates to be deployed;
- Deploying the necessary patches/updates;
- Testing and confirming the successful installation of patches/updates;
- A regression plan in the event of a failed patch/update installation.

USE METHODOLOGICAL PRODUCTION [TE.08]

- **Enable and USE production tool checking features [TE.08.30]**

  Use production tool checking features: In general, the production tool will either generate un-compiled code with associated errors/warnings/unhandled exceptions or generate compiled code with associated warnings/unhandled exceptions.

  In either scenario, you should always fix all errors, warnings and exceptions generated by the compiler prior to live deployment, this is to guarantee that you have made best efforts to ensure that your code is free of defects/deviations.

8 ARTEFACT MANAGEMENT (E5)

8.1. OBJECTIVES

The objectives of 'Artefact Management' are to ascertain that the required level of trustworthiness has been achieved by the software (pre-release) and to ensure that it is maintained at a comparable level throughout its life-cycle.

8.2. INTRODUCTION

It is important that the software asset and any artefacts produced or used by the software are protected and appropriately managed to ensure the continued trustworthiness of the software throughout its lifecycle.

The term 'Artefact', for purposes of this document, cover deliverables such as:

- Source Code;
- Any Libraries, Components or Dependencies that the source code relies upon;
- Software design and development documentation;
- Test documentation;
- Release documentation.

You should ensure that consideration is given to the level of protection assigned to those software artefacts (both within and outside of your control), not only to protect them from unauthorised modification or loss, but to ensure continued availability and maintenance for as long as is required.

8.3. FUNDAMENTAL CONTROLS

The following control measures are required to be established (or demonstrated) to meet the TSE objective 'Artefact Management' (E5):

- TSMS (section 3.1)
- TSRA (section 3.2)
- TSRN (section 3.3)
- TSCDM (section 3.3)
- TSDDL (section 3.3)

8.4. BASIC CONTROLS

The controls listed below meet the TSE objective 'Artefact Management' (E5) and each should be implemented in a prescriptive manner as required by the TSMS. Where this is not practicable, this should be documented (in accordance with Section 3), and either alternative controls implemented or the risk arising from non-adoption appropriately treated.

UNDERSTAND REQUIREMENTS [PH.02]

- **Implement protection of source code, covering Confidentiality, Integrity and Availability (CIA) [PH.02.10]**

  The unauthorised access of your source code, in addition to potential breach of copyright, may also result in scenarios such as:

  - external review of the code with subsequent exploitation of vulnerabilities;
  - injection of vulnerabilities or backdoors into the code; or
  - modification of the way in which the code operates;

  It is therefore important that you ensure your source code repository is adequately protected from unauthorised access (for example through use of access control), that a version control system is in place with all transactions logged and that where appropriate a software escrow service is used to guarantee availability.

PERFORM TRUSTED SOFTWARE ASSET MANAGEMENT [PR.06]

- **Software assets should be reviewed regularly [PR.06.40]**

  Wherever possible, you should either hold local copies of all software artefacts or make use of an escrow service [TE.11.10]). Regular reviews should be conducted on all artefacts to ensure that access is available, they are adequately protected and the versions held are correct [TE.03.30].

ENABLE DEPENDABLE DEPLOYMENT [TE.11]

- **Ensure source code can be obtained throughout lifecycle, such as by Escrow service [TE.11.10]**

  An escrow service is the secure holding of source code by an independent third party that ensures the availability of that source code to users/buyers/licensees under contractually agreed terms of an escrow agreement, for example in the event of bankruptcy, merger or acquisition of the supplier.

  If you are dependent upon a third-party for the provision of software (for example through commissioned bespoke software), you should consider an escrow agreement in order to ensure the continued accessibility of the necessary artefacts required for the ongoing operation and maintenance of that software throughout its life-cycle.

PERFORM INTERNAL PRE-RELEASE REVIEW [TE.09]

- **Issue formal Trustworthy Software Release Notice (TSRN) [TE.09.50]**

  The Trustworthy Software Release Notice (TSRN) is intended to be produced subsequent to Acceptance Testing and should form a summary of the controls considered for each facet of trustworthiness (i.e. safety, reliability, availability, resilience and security) during software specification (E1: Scope for Use), development (E2: Coding Approach, E3: Use Tools Effectively and E4: Defect Management) and release (E5: Artefact Management).

  For each control, the TSRN should cover all constraints, dependencies and assumptions (taken from the TSCDM) together with details of any unmitigated defects and deviations (taken from the R-TSDDL) together with justifications and/or plans for future mitigation.

  The TSRN should be signed by the Trustworthy Software Release Authority (TSRA) who is responsible for ensuring that consideration has been given to all relevant trustworthy software concepts, principles and techniques before software release. Further information regarding the TSRN can be found in Section 3.3 of this document.

9 EVOLUTION

9.1. STATUS/PLAN

This document will be updated on a periodic basis to reflect the constantly evolving nature of the cyber ecosystem and therefore, the need for trustworthiness.

9.2. MAINTENANCE/CONTRIBUTIONS

TSI welcomes input from Stakeholders on all aspects of its activity, including any additions or amendments to the Trustworthy Software Library (TSL).

If you have any suggestions please feel free to engage with TSI, either through your existing contact, or by emailing enquiries@uk-tsi.org.

ANNEX A: LIFECYCLE MAPPING

TSE CONTROLS		SOFTWARE LIFECYCLE STAGES		
		SPECIFY	REALISE	USE
E1: SCOPE FOR USE				
PR.03: UNDERSTAND REQUIREMENTS:				
PR.03.10	Specify Explicit / Functional Requirements	✓		
PR.03.20	Specify Implicit / Non-functional Requirements	✓		
PR.03.30	Understand Implicit / Non-objective Requirements	✓		
PR.03.40	Understand Use Cases	✓		
PR.03.50	Monitor and record Derived Requirements	✓		
TE.05: SEEK TRUSTWORTHY REALISATION:				
TE.05.80	Implement only the minimum set *of features* of users to meet requirements	✓	✓	
E2: CODING APPROACH				
TE.02: MAKE APPROPRIATE TOOL CHOICES:				
TE.02.20	Produce and maintain Coding Standards	✓	✓	
TE.03: FOLLOW STRUCTURED DESIGN				
TE.03.30	Use only proven components		✓	
TE.04: FOLLOW STRUCTURED IMPLEMENTATION:				
TE.04.10	Produce bespoke components in accordance with Coding Standards	✓	✓	
TE.04.20	Use appropriate and recognised data formats		✓	
TE.04.30	Select appropriate algorithms	✓	✓	
TE.05: SEEK TRUSTWORTHY REALISATION:				
TE.05.10	Review Design for failure modes	✓	✓	
TE.05.20	Source and configure off the shelf components in accordance with Design/Effect Pattern(s)	✓	✓	
TE.05.45	Ensure mitigations are used for all identified failure modes	✓	✓	
TE.05.65	Implement measures to control malicious code		✓	✓
TE:06: MINIMISE RISK EXPOSURE:				
TE.06.10	Only grant minimum Privileges required, with all other actions defaulting to not permitted		✓	✓
TE.06.20	Separate program data, executables, and configuration data		✓	✓
TE.07: PRACTICE HYGIENIC CODING:				
TE.07.05	Ensure all variables, pointers and references are properly initialised at first and subsequent uses		✓	
TE.07.10	Ensure all Input Data is Validated		✓	
TE.07.15	Ensure all Messages are Validated		✓	
TE.07.20	Ensure implementations of all Algorithms are Validated		✓	
TE.07.25	Ensure all Output Data is Validated		✓	
TE.07.30	Ensure Error Handling is implemented comprehensively, and		✓	

TSE CONTROLS		SOFTWARE LIFECYCLE STAGES		
		SPECIFY	REALISE	USE
	"fails safe and secure"			
TE.07.35	Apply consistent naming convention	✓	✓	
TE.07.40	Manage resource access explicitly (buffers, stacks, memory, cache and files)		✓	
TE.07.50	Explicitly remove detritus (temporary files/logs)		✓	
E3: USE TOOLS EFFECTIVELY				
TE.02: MAKE APPROPRIATE TOOL CHOICES:				
TE.02.10	Selection of appropriate Programming Language(s), considering known vulnerabilities and needs for Typing	✓	✓	
TE.02.40	Selection of appropriate Testing Tools		✓	
TE.08: USE METHODOLOGICAL PRODUCTION:				
TE.08.30	ENABLE and use production tool checking features		✓	
E4: DEFECT MANAGEMENT				
PR.07: MAINTAIN DEFECT MANAGEMENT:				
PR.07.10	Ensure all Defects identified during Realisation are recorded, reported and assessed, with rectification at earliest opportunity using process for monitoring through a Realisation Trustworthy Software Defect and Deviation List (R-TSDDL)		✓	
PR.07.20	Ensure all In Service Defects and Deviations are recorded in an In-service Trustworthy Software Defect and Deviation List (I-TSDDL), reported and assessed, with rectification at earliest opportunity using process for monitoring of deferrals			✓
TE.11: ENABLE DEPENDABLE DEPLOYMENT:				
TE.11.60	Updating and Patching of implemented software, with routine, critical and emergency options, taking due cognisance of R-TSDDL / I-TSDDL			✓
TE.08: USE METHODOLOGICAL PRODUCTION:				
TE.08.30	Enable and USE production tool checking features		✓	
E5: ARTEFACT MANAGEMENT				
PH.02: UNDERSTAND REQUIREMENTS:				
PH.02.10	Implement protection of source code, covering Confidentiality, Integrity and Availability (CIA)			✓
PR.06: PERFORM TRUSTED SOFTWARE ASSET MANAGEMENT:				
PR.06.40	Software assets should be reviewed regularly			✓
TE.11: ENABLE DEPENDABLE DEPLOYMENT:				
TE.11.10	Ensure source code can be obtained throughout lifecycle, such as by Escrow service			✓
TE.09: PERFORM INTERNAL PRE-RELEASE REVIEW:				
TE.09.50	Issue formal Trustworthy Software Release Notice (TSRN)		✓	

ANNEX B: REFERENCES

B.1 ASSOCIATED DOCUMENTS (AD)

The following TSI documents provide additional information that may be relevant to some or all of the readership.

[AD.1] PAS 754:2014 "Software Trustworthiness. Governance and management. Specification"

[AD.2] TSI Risk Management

[AD.3] TL Guidance

[AD.4] TSE QRG

[AD.5] TSE SA Guidance

[AD.6] TS SAQ-P

[AD.7] TS SAQ-O

[AD.8] TS SAQ-C

B.2 REFERENCE DOCUMENTS (RD)

The following documents from organisations other than TSI provide additional information that may be relevant to some or all of the readership.

[RD.1] "27010:2012 Information technology -- Security techniques -- Information security management for inter-sector and inter-organisational communications", ISO/IEC, April 2012

[RD.2] "Cyber Essentials Scheme", HM Government (HMG), April 2014

[RD.3] "15288:2015 -- Systems and software engineering -- System life cycle processes", International Standards Organisation (ISO) / International Electrotechnical Commission (IEC), May 2015

[RD.4] "12207:2008 -- Systems and software engineering -- Software life cycle processes", ISO/IEC, August 2008

ANNEX B: ABBREVIATIONS

CIA	Confidentiality, Integrity, Availability
CM	Configuration Management
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWSS	Common Weakness Scoring System
ETSI	European Telecommunications Standards Institute
FOSS	Free and Open-Source Software
ISMS	Information Security Management System
I-TSDDL	In-service Trustworthy Software Defect and Deviation List
M/E	Mass Market with Explicit Need
M/I	Mass Market with Implicit Need
N/E	Niche with Explicit Need
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OGL	Open Government Licence
PAS	Publicly Available Specification
PDCA	Plan-Do-Check-Act
PH.NN.NN	Physical (controls)
PR.NN.NN	Procedural (controls)
QAMS	Quality Assurance Management System
QRG	Quick Reference Guide
R-TSDDL	Realisation Trustworthy Software Defect and Deviation List
SA	Self-Assessment
SAQ	Self-Assessment Questionnaire
SAQ-C	Self-Assessment Questionnaire for Components
SAQ-O	Self-Assessment Questionnaire for Organisations
SAQ-P	Self-Assessment Questionnaire for Practitioners
SQL	Structured Query Language

SRS	Software Requirements Specification
TE.NN.NN	Technical (controls)
TL	Trustworthiness Level
TOCTOU	Time-of-Check, Time-of-Use
TSCDM	Trustworthy Software Constraint and Dependency Model
TSDDL	Trustworthy Software Defect and Deviation List
TSE	Trustworthy Software Essentials
TSF	Trustworthy Software Framework
TSI	Trustworthy Software Initiative
TSL	Trustworthy Software Library
TSMS	Trustworthy Software Management System
TSRA	Trustworthy Software Release Authority
TSRN	Trustworthy Software Release Note
VR	Vulnerability Review

ANNEX C: GLOSSARY

For purposes of this document, the following terms and definitions apply:

AVAILABILITY (FACET) The ability of the system to deliver services when requested.

DEFECT A defect can be classed as:

- a coding error/mistake; and/or
- non-fulfilment of an explicit/implicit software requirement.

DEFERRAL A documented and risk managed decision to not resolve a defect or deviation.

DEVIATION A deviation can be classed as:

- an unexpected outcome during run-time that is not caused by a coding error or mistake; and/or
- non-conformity with an explicit/implicit software requirement, due to misinterpretation (or otherwise) of the SRS.

PRODUCTION TOOL A production tool is a program that converts the source code (written in a programming language) into instructions (object code) that the machine understands. As part of this procedure, the following functions may be performed: Pre-processing; Lexical analysis; Syntax analysis/Parsing; Semantic analysis (syntax-directed translation); Code generation; Code optimization; Error Handling; Exception Handling; Security Checks.

RELIABILITY (FACET) The ability of the system to deliver services as specified.

RESILIENCE (FACET) The ability of the system to transform, renew and recover in timely response to events.

SAFETY (FACET) The ability of the system to operate without harmful states.

SECURITY (FACET) The ability of the system to remain protected against accidental or deliberate attacks.

TRAFFIC LIGHT PROTOCOL (TLP) TSI uses the Traffic Light Protocol (TLP), as defined in ISO/IEC 27010:2012 [RD.1], with the following usage convention:

WHITE:	Open Published Material with no restrictions on distribution.
GREEN:	Released Material, not for Website.
AMBER:	Draft (or other relevant) Material, for circulation through direct stakeholder channels and applicable for onward distribution within recipient body.
RED:	Internal (draft or otherwise) material, only for further circulation on reference to originator.