OCT 2017

WHITE PAPER

DISCUSSION DOCUMENT

Towards **Trustable Software**

A SYSTEMATIC APPROACH TO ESTABLISHING TRUST IN SOFTWARE

Institute for Strategy Resilience & Security
University College London

in association with

CodeThink

Foreword

by the Rt Hon. Lord Reid of Cardowan
Executive Chairman, ISRS

As society's dependence on software increases, there has never been a greater need to address the question: should we trust the software on which we depend? This paper examines the critical role that software now plays in virtually all aspects of modern life and, in particular, the degree to which it can be trusted under various circumstances.

The proposed concept of "a trustable software process" is a step towards an essential, underpinning platform to ensure solid foundations of societal resilience, analogous to existing trust-generation mechanisms in key industries such as finance, healthcare and construction.

Within the space of a few decades, virtually all of human society has become critically dependent upon computer software. This trend is set to accelerate as our industries, homes, transportation and a plethora of mechanical and electronic objects – the "Internet of Things" – become increasingly software controlled and interconnected. However, while improved regulation of construction, healthcare, law, financial services and other fundamental building blocks of human society has achieved ever greater levels of trust, the reverse is true of software. Software has evolved from basic electronic circuits and simple control logic to levels of unfathomable complexity containing hundreds of millions of lines of code.

In many critical cases, the notion that computer software may be trusted is a house built on sand. Engineering practices within the software industry, which would be considered irresponsible in construction or mechanical engineering, have led to the creation of systems that are not simply unworthy of trust, but incapable of having their level of trustworthiness assessed.

This blind spot has crept up upon us and it should be of major concern to governments, regulators, the software industry and the general public because it risks future crises of confidence, when these systems inevitably fail in unexpected ways, with far reaching and possibly systemic consequences.

We very much hope that this paper will serve to stimulate discussion of the first principles and steps towards consensus as to how software should be designed, constructed and operated so as to be trustable.

Rt Hon. Lord Reid of Cardowan

Institute for Strategy Resilience & Security
University College London

Codethink

Contents

Foreword	2
Executive Summary	4
Introduction: The Need for Trustable Software	7
The Value of a Trustable Approach	10
Why Industry Cannot Solve This Alone	12
A Manifesto for Trustable Software	15
Conclusions: Applying Trustability to Software	20
References	21

TO PARTICIPATE IN THE TRUSTABLE DISCUSSION, LEAVE COMMENTS ON THIS WHITE PAPER AND HELP SHAPE WHAT NEEDS TO HAPPEN NEXT, PLEASE REGISTER YOUR INTEREST AT

www.trustablesoftware.com

ISRS and Codethink thank the many contributors to this white paper and in particular:

Codethink: Emmet Hickory, Paul Sherwood, Edmund Sutcliffe

ISRS: Peter Domican, David Levinger, Prof JP MacIntosh, Asutosh Yagnik

About Codethink

Codethink is a leading provider of effective software engineering solutions, particularly in the infrastructure critical space. The company develops and maintains system and device-level software supporting advanced technical applications for its international corporate clients, across a range of industries including aerospace, automotive, finance, medical and telecoms. Codethink has pioneered software industry thinking around the concepts of trustable software, with a view to improving the quality of software engineering for societal good.

About the Institute for Strategy, Resilience & Security (ISRS) at UCL

Over the last decade the Institute for Strategy Resilience & Security (ISRS) at UCL has served as a pioneer and forum for next generation thinking. Founded by the Rt Hon. Lord Reid of Cardowan, ISRS provides analysis and assessment of the major issues of resilience with respect to national and global infrastructure and the ability of governments, regulators and businesses to respond to them. The Institute advises industry and the public sector on the persistent challenges to their agility, stamina and capacity in strategic decision making, so as to better face existential threats, risks, and disruptive innovation that are not addressed by conventional strategy and forecasting.

For industry and public sector queries in relation to this paper, please email: info@isrs.org.uk

For media enquiries about this paper, please email: press@isrs.org.uk

EXECUTIVE SUMMARY

Towards Trustable Software

Within the space of a few decades, human civilisation has become highly dependent upon computer software, a trend that shows no signs of reversing, as our industries, homes, transportation and a plethora of mechanical and electronic objects – the "Internet of Things" – become increasingly software controlled and interconnected.

However, while improved regulation of construction, healthcare, law, financial services and other fundamental building blocks of human society has achieved ever greater levels of trust, the reverse is true of software. Engineering practices within the software industry, which would be considered irresponsible in construction or mechanical engineering, have led to the creation of systems that are not simply unworthy of trust, but incapable of having their level of trustworthiness assessed. As software has evolved from basic, highly deterministic electronic circuits and simple control logic to levels of unfathomable complexity contained within billions of lines of code, this blind spot has crept up upon us and it should be of major concern to governments, regulators, the software industry and the general public. When these systems fail in unexpected ways, as they inevitably must, they risk not only far reaching and potentially systemic consequences, but also triggering future crises of confidence in the products and services that they support.

Existing initiatives to improve the *trustworthiness* of software have focused largely on how to build better software by improving safety, reliability, availablility, resilience and security. Complementary to these, the concept of trustable software proposes a general solution that adds auditability to the software development process, enabling parties in the value chain to assess the degree to which they can trust a particular piece of computer code, in the same way that audit trails provide confidence in other industries. Auditability does not *de facto* enforce trust, rather it strongly incentivises behaviours along the value chain that lead to it through increased transparency of the process.

All critical products and services upon which human health, safety and security depend, have, of necessity, evolved recognisable processes to provide transparency and allow assessment of the degree to which to which that product or service is capable of being trusted. We refer to these as trustable processes because they generate the ability to trust. These vary from industry to industry, but generally take the form of laws, regulations, standards and audit practices. They provide confidence that e.g. a pill may be swallowed, that is safe to board an aircraft - conversely, that the risk of using a product or service is worth accepting.

However, there exists an important exception – software. In an age of increasing reliance upon software and ever more complex, interconnected and interdependent systems, we must address the question: to what extent can we trust this software?

Unlike physical construction, software does not have to conform to a set of building standards; unlike the pharmaceutical industry, there are no notified bodies or regulators; unlike the legal profession there is not a single body upholding standards of practice, and unlike accounting, software is unaudited. There is currently no recognisable process, regulatory framework, set of standards or audit trail by which, at any stage, it is possible to assess the degree to which software is capable of being trusted. Instead, software use remains largely an act of faith, built upon a stack of unverified assumptions, as most computer code is written informally and evaluated based on whether or not it works. Little software is formally verified to be error free and it is generally supplied in an opaque manner to its users. Even open source software, while in principle visible in its entirety, is in practice often so large and complex that fully understanding its operation is unfeasible.

We are on the cusp of further dramatic increases in the capabilities and complexity of software, and the issues of trust that are raised by its use are set to increase exponentially with the advent of evolving technologies such as robotics and artificial intelligence. Virtually every aspect of human life will involve or be controlled entirely by devices incorporating software. Software is now used to deliver many essential public services and

to support critical national infrastructure. The loss or denial of service for any reason, accidental or deliberate, has potential consequences that range from mere inconvenience and reputational damage, to financial loss and ultimately loss of life.

The advent of the driverless car will finalise an ongoing engineering paradigm inversion, whereby a vehicle that is currently considered primarily as a mechanical object supported by software will become viewed as primarily software encapsulated within mechanical components. Concerns of safety and security will shift from trust in mechanical components to trust in the software, for example, can a cyber attacker take control of the vehicle?

A Critical Issue to Address Now

In the wake of the global financial crisis of 2007-2008, it became clear that the crisis was avoidable and was caused by widespread failures in regulation and supervision, poor management of accumulated systemic risk, lack of transparency, breakdown in accountability and ethics and failures to correctly price risk.

Analogously, despite an urgent unmet need, the software industry is inexorably drawn towards fuelling growth and will de facto ignore and resist a "push" towards a systematic approach to trust in software. An equivalent "pull" is required by governments and regulators in recognising the problem and encouraging the adoption of trustablility as standard practice, before a series of events or a particular disaster forces this issue into the wider public domain and Government is required to compel industry post hoc to address the issue of trust in software.

By 2020, at least 20 billion devices will be connected to the Internet, each more complex, interconnected and interdependent than ever before. Ignoring the systemic risks, lack of transparency, breakdown in accountability and failure of regulatory supervision holds the potential to accumulate a crisis as potent as any previously experienced.

Operating in an environment with software supplied 'as safe as possible', as it currently is, but without an auditable process for verifying the provenance and testing of that code, is no longer appropriate. Without adopting a process by which the trustability of software can be determined, society will increasingly stumble from one problem to the next. Whether this is experienced as failure in use, increased cyberattacks, or financial loss, the result will inevitably lead to an erosion of public confidence with repercussions for governments and regulators. Despite the challenges of adopting this approach in an industry that has historically been relatively free of constraints, the opportunity for nation states that are early adopters is competitive advantage through the creation of a safer society and a safer place to do business.

Trustability: An Established Key to Trust

A trustable process can be defined as "auditable in such a way that, at any point in the process, one can assess the degree to which it can be trusted". Although this term may be unfamiliar in everyday language, examples in use are immediately recognisable e.g. financial auditing is an established process that evolved over centuries in response to the need for trust in finance. The handling of evidence in the criminal justice system also follows a strict process so that a jury can have confidence in the provenance of evidence and that it has not been tampered with.

The requirements and steps of these trustable processes may at first appear to have little in common. However, all such processes share a set of features that enable trustability: those providing a product, service or information are required to present detailed evidence on the provenance, manufacture, testing and validity of what is being supplied. The evidence required, its format and the standards for preparing that evidence are specified by a regulator or agency, and it is then made available to a nominated body to inspect and audit to certify its accuracy.

To find out more please visit:
www.trustablesoftware.com

Paul Sherwood
CEO, Codethink

In 2016, after decades of private discussion with colleagues and international customers, Codethink CEO Paul Sherwood initiated the public debate on 'trustable software engineering'.

Towards **Trustable Software**

Our concern reached a tipping point with the realisation that society is racing towards adoption of autonomous vehicles, without any ability to prove that their control software is safe or secure.

From two decades of witnessing software practices across multiple industries, three things are clear:

- most software developers lack the knowledge and skills to make their work safe and secure
- there are no consistent or reliable measures for software risks, quality, productivity, or costs.
- most software users do not understand the risks

While mature industries have standards, and regulations that require work to be checked, both by the implementing organisation and independently, the software industry has not addressed the needs for consistent and reliable measures for software risks, quality, productivity, or costs.

As a result all software, even that governing critical infrastructure, is expected to have bugs and vulnerabilities, and is used with little awareness of its risks. We are all therefore exposed to financial, physical and emotional harm from software, without the required processes for redress or improvement that are norms in other industries.

We need to stimulate improvement in a way that raises the bar across the entire industry for service providers, software vendors, operators and users.

Institute for Strategy Resilience & Security
University College London

CodeNink

Introduction

The Importance of this Paper

While software has become critical to virtually all aspects of modern life, processes for determining whether we can trust it are conspicuously absent.

The goal of this paper is to stimulate discussion of the urgent need, potential solutions and proposed next steps to address the systemic risks posed by that gap.

Among stakeholder groups – vendors, purchasers, software engineers, computer scientists, government and regulators – there exists little, if any, consensus as to how software should be designed, constructed and operated to achieve this.

We examine current approaches and deficiencies within the software industry towards the issue of trust and propose the concept of a *trustable software engineering process* as a necessary and appropriate underpinning platform to ensure solid foundations for the trust of software going forward.

The principles of how that process might work are outlined, by establishing software engineering practices that generate audit information at all stages of creation, deployment, change and use, to enable the continual assessment of trust, just as this is done in other industries.

The Need for Trustable Software

Trust is the basis upon which democracy, modern economics and societal stability have been built. Underpinning public and market confidence, trust in our political, legal and financial frameworks generates willingness to delegate control, be governed, accept taxation, invest, partner and respect ownership. Through minimising the pricing of risk and arbitrage into transactions, trust has enabled efficient markets, confidence in banking and the economic expansion of civilisation. The emergence of cryptocurrencies and concepts such as the Internet of Agreements are based on distributed ledgers as next generation systems of trust.

All critical products and services upon which human health, safety and security depend, have, of necessity, evolved recognisable processes to provide transparency and allow assessment of the degree to which to which that product or service is capable of being trusted. We will refer to these as *trustable processes* because they generate the ability to trust. These vary from industry to industry, but generally take the form of laws, regulations, standards and audit practices. They provide confidence that a pill may be swallowed, that a bridge may be crossed, that fire safety has been adequately provided for, that is safe to board an aircraft - conversely, that the risk of using a product or service is worth accepting.

However, there exists an important exception – software. In an age of increasing reliance upon software and ever more complex, interconnected and interdependent systems, we must address the question: *to what extent can we trust this software?*

Unlike physical construction, software does not have to conform to a set of building standards; unlike the pharmaceutical industry, there are no notified bodies or regulators; unlike the legal profession there is not a single body upholding standards of practice, and unlike accounting, software is unaudited. There is currently no recognisable process, regulatory framework, set of standards or audit trail by which, at any stage, it is possible to assess the degree to which software is capable of being trusted.

Instead, software use remains largely an act of faith, built upon a stack of unverified assumptions, as most computer code is written informally and evaluated based on whether or not it works. Little software is formally verified to be error free and it is generally supplied in an opaque manner to its users. Even open source software, while in principle visible in its entirety, is in practice often so large and complex that fully understanding its operation is unfeasible.

There is currently no recognisable set of processes, regulation, set of standards or audit trail by which, at any stage, it is possible to assess the degree to which software software is capable of being trusted.

Five key unknowns lie at the heart of risks posed by this lack of transparency:

- Where does the code come from and who wrote it?
- Does the code do what it is supposed to do and does it not do what it is not supposed to do?
- How was the code built and tested prior to deployment?
- Can we reproduce it exactly as it was originally generated?
- Can we maintain it without breaking it?

Shifting from *ad hoc* to *systematic* trust

It is hard to overstate the degree to which software plays a pivotal role in the critical infrastructure and vital functioning of modern human society.

The operation of our homes, workplaces, government, education system, food and energy production, communications, logistics, healthcare and financial systems are increasingly reliant upon its correct operation. The loss or denial of service for any reason, accidental or deliberate, has potential consequences that range from mere inconvenience and reputational damage, to financial loss and ultimately loss of life. As these system evolve with increasing levels of capability, complexity and interconnectivity, we believe that it is essential to consider replacing the current ad hoc approach to trust in software with a systematic approach.

In engineering, software has become ubiquitous and inseparable from the mechanical systems it supports. The advent of the driverless car will finalise an ongoing paradigm inversion, whereby a vehicle that is currently considered primarily as a mechanical object supported by software will become viewed as primarily software encapsulated within mechanical components. Concerns of safety and security will shift from trust in the mechanical components, such as, whether the brakes work, to trust in the software, for example, can a cyber attacker take control of the vehicle, or the consequences of software failure at high speed.

Operating in an environment with software supplied 'as safe as possible', as it currently is, but without an auditable process for verifying the provenance and testing of that code, is no longer appropriate. Without adopting a process by which the trustability of software can be determined, society will increasingly stumble from one problem to the next. Whether this is experienced as failure in use, increased cyberattacks, or financial loss, the result will inevitably lead to an erosion of public confidence with repercussions for governments and regulators.

Institute for Strategy Resilience & Security
University College London

CodeThink

A Critical Issue To Address Now

In the wake of the global financial crisis of 2007-2008, it became clear that the crisis was avoidable and was caused by widespread failures in regulation and supervision, poor management of accumulated systemic risk, lack of transparency, breakdown in accountability and ethics and failures to correctly price risk.

Analogously, despite the urgent unmet need, the software industry is inexorably drawn towards fuelling growth and will *de facto* ignore and resist this "push" towards a systematic approach to trust in software. An equivalent "pull" is required by governments and regulators in recognising the problem and encouraging the adoption of trustablility as standard practice, before a series of events or a particular disaster forces this issue into the wider public domain and Government is required to compel industry *post hoc* to address the issue of trust in software.

By 2020, at least 20 billion devices[1] will be connected to the Internet, each more complex, interconnected and interdependent than ever before. Ignoring the systemic risks, lack of transparency, breakdown in accountability and failure of regulatory supervision holds the potential to accumulate a crisis as potent as any previously experienced.

A **trustable process** is auditable in such a way that, at any point in the process, one can assess the degree to which it can be trusted.

It enables stakeholders to decide how much they can trust its outputs by providing transparency about the process and its inputs.

Trustability: An Established Key to Trust

A *trustable* process can be defined as "auditable in such a way that, at any point in the process, one can assess the degree to which it can be trusted". Although this term may be unfamiliar in everyday language, examples in use are immediately recognisable and underpin the existence of industries such as construction, financial services, healthcare, aerospace, nuclear power and public transportation, where safety and security are paramount, and the consequences of failure are substantial.

Financial auditing is an established process that evolved over centuries in response to the need for trust in finance. The handling of evidence in the criminal justice system[2] also follows a strict process so that a jury can have confidence in the provenance of evidence and that it has not been tampered with.

The requirements and steps of these trustable processes may at first glance appear to have little in common. However, all such processes share a set of features that enable trustability: those providing a product, service or information are required to present detailed evidence on the provenance, manufacture, testing and validity of what is being supplied. The evidence required, its format, the standards for preparation and storage are specified by a regulator or agency, and it is then made available to a nominated body to inspect and audit to certify its accuracy.

The Value of a Trustable Approach

For Governments: A Systemic Solution Underpinning Public and Market Confidence

Governments are simultaneously eager to exploit the economic growth, investment and jobs associated with software creation yet face a myriad of issues, risks and concerns. Addressing systemic issues due to trust in software would be a major step forward for governments as a resilient and systematic response rather than a case by case approach. An implemented trustable software process would underpin stakeholder confidence in the industry, reducing risk and increasing trust between parties and software in use. In turn this would enable other industries to accelerate innovation, driving economic growth and improved societal safety, security and prosperity.

For Regulators: An Enabler Of Regulatory And Standards-Based Oversight

A well designed trustable process has significant advantages for regulators and provides governments and their agencies with a scalable framework for both assessing risk and regulating the use of software.

The onus on the provider to show that their software is trustable reduces the workload of the regulator. Regulators need only hold top level information, with the participating companies writing agreed detailed code and data into a agreed immutable audit log , which cannot be accessed retrospectively by the company. This reduces both cost and complexity for the regulator who can audit this information as part of a forensic examination in the case of a major incident. Software that fails to meet a documented requirement as part of the operation of the regulated product or service would be detectable and could, if desired, be suspended.

For Insurers: More Accurate Assessment of Liability & Provision of Cover

The vast majority of software licenses explicitly disclaim any liability, even for fitness to purpose, whilst some contain very limited warranties regarding the quality of the licensed software. Financial compensation for loss and damage suffered by the purchaser as a result of defective software is rare. Assessing companies and products against generic compliance checklists is unlikely to evaluate their cyber-risk adequately.

For an insurer, the cost of cover needs to reflect the risks faced and the potential consequential payout to policyholders. For the insured, a policy needs to be meaningful and likely to pay out in reasonable circumstances. A trustable process for software

would form part of a more dynamic method of minimising vulnerabilities, assessing risk, and defining sensible exclusion thus creating an efficient cyber insurance market.

For the Legal System: Immutable Evidence

Software failure that results in loss inevitably leads to disputes around liability. Trustable software provides a body of evidence around the provenance of the code, authorisations and its testing. In the event of a dispute over failure, lawyers and experts would have access to an immutable evidence trail with which to examine the sequence of events and code delivered, which can be trusted when used in court. This, in turn, holds the potential to enable expert witness evidence to be more objective and cases more efficient.

For Industry: Recognition of Value

There is a growing recognition within the industry that a higher standard of trust, beyond simply increasing quality, is required and there is a groundswell within the software industry coalescing around the concept of trustable.

Early adopters will benefit from increased recognition of their trustable products and services. Companies that choose to progress this now will be in a much stronger position to determine the direction that trustable will take and anticipate the needs of governments,regulators and the public rather than have conditions imposed in haste. The barriers to adoption of trustable principles are not insurmountable, if industry is involved in designing the best solution.

For Purchasers: Value For Money

Interpretation of trustable audit information enables stakeholders to evaluate performance and process efficiency. Output measured as lines of code is a poor indicator of the software creation process - trustable provides insight.

For the Public: Confidence

Trustable processes address the asymmetry of knowledge and resources between providers and users. Users can consider a product worthy of trust because they have trust in the process even though they have no detailed knowledge of how it works. Just as a “CE mark” on a product gives the public confidence that a product meets the appropriate EU regulations, so it may, in future, be possible for the public to have confidence in software that has been produced through a recognisable trustable process.

Why industry cannot solve this alone

Our Journey to a New Era of Computing

Human kind is entering a new era of computing, and possibly a new evolution of its own existence, as many functions and critical decisions are increasingly taken over by software, without human oversight. Satellite navigation systems have removed the need for a driver to think about which route to take and users can effectively defer low level selection choices to the algorithms in home assistants such as Siri and Alexa. Software is no longer just an add-on helping the driver to control the car but is increasingly controlling the vehicle -- the "driverless car" is now a reality in software terms.

Non-Deterministic Systems

From the first conception of mechanical computation engines by Babbage and Lovelace in the 19th century, the inception of modern computer devices by Turing in the 1930's and the first implementation of stored electronic memory in the 1940's, software was originally an expression of deterministic mathematics and algorithms designed to solve bounded problems.

In the past, one could place a larger degree of trust in software prior to deployment. Code was small in volume (typically a few thousand lines), simple and deterministic. The code was extensively user tested prior to deployment onto systems running on isolated secure computers and to hack a system required resources and knowledge to exploit vulnerabilities. The situation today is radically different.

Today software has diversified in construction and operation to include innumerable underlying combinations of hardware, firmware, operating systems, programming languages, tools and software services which are constantly being changed and updated. These systems are moving increasingly to non-deterministic, based on artificial intelligence, neural networks and machine learning.

An Explosion of Complexity

It is now common for systems to contain millions (Facebook) or even billions (Google) of lines of code. In the automobile industry, software in cars has evolved from a few thousand lines of code to control specific functions, to 100 million lines of code in the modern luxury car[3] controlling all aspects of the car including throttle control and simple parking manoeuvres. Furthermore, with current microservices architecture, while reviewing many small pieces of code is still possible for an individual, it is becoming impossible for them to have a systematic view of all possible interactions.

The net affect of this complexity is that purchasers have lost

the ability to adequately test the software they are buying, and in practice have to rely on the vendor to assert fitness for purpose, despite the fact that vendors explicitly disclaim fitness for purpose in their legal small print.

In addition to proliferation in scale, interconnectivity generated by the intermeshing of the myriad components involved has resulted in a complex ecosystem. This complexity and interconnectedness make systems significantly more vulnerable to failure or cyber attack from an ever increasing number of people with the skills to do it.

The increasing power and portability of devices on which software operates from mainframes to personal computers, through to smartphones and wearable devices, has led to an exponential increase in available applications of ever broader utility. Key trends in technology and the key indicators of technology growth, indicate that this growth is far from at an end and that this pace of change is accelerating rather than slowing down. As each new device creates new potential applications this in turn generates new demand.

A Gold Rush Culture Where Innovation and Profits Trump Prudence

As of March 2017, Android users were able to choose between 2.8 million apps; Apple users were able to chooise between 2.2 million apps.[5]

Like the Industrial Revolution, the internet and its myriad of devices and applications, has created an entirely new global economy. It appears to be an apparently boundless market where each new development creates ever more opportunity. According to Gartner, worldwide IT spending is forecast to reach $3.5 trillion in 2017, while software spending is projected to grow 7.2 percent in 2017 to $357 billion[4].

Like its 19th century counterpart, in a period of unfettered innovation where productivity increases outstripped worker safety, it appears to have passed a tipping point where the ubiquity and importance of software are such that considerations of trust can no longer just be left to the industry and the market alone.

Like banks in the pre-crisis era before 2007, under “gold rush” conditions, capturing growth and market share is all important for commercial developers and there is little incentive to promote a regulated environment in which innovation and profits are constrained. Rather problems can be “fixed” later, once they are experienced in use.

The high growth labour market for developers also places pressure on the level of requisite experience. Today, an “experienced” developer may only have five years of real world experience with a particular programming language. With labour at a premium, the industry is focused on training which produces more developers yet resistant to requirements for professional qualifications and evidence based professional development which

slows down supply. This makes rational sense in terms of the industry but, in terms of external trust, the opposite is often true. No reputable civil or mechanical engineering firm would use an unqualified engineer, irrespective of their talent.

Existing Software Quality and Trust

Much activity has been dedicated to improving software engineering processes by which code is produced. Generally accepted standards are collected in the Software Engineering Body of Knowledge (SWEBOK), and international standard ISO/IEC TR 19759:2005[5].

Previous initiatives around trust in software have concentrated on quality standards and improving engineering processes on the premise that a reliable system creates trust. The recent Trustworthy Software Framework[6] aimed to introduce methodologies to improve quality and hence trust. Like other quality-based methodologies, it fails to consider the provenance of the software, which is an essential consideration in the case of any major failure or cyber attack. Though the impact of this approach is yet to be fully assessed, it seems to counter the industry trend for minimum regulation and maximum flexibility.

Formal Methods

Formal methods[7] are increasingly being used to help reduce errors in programs as a response to cyber attack. Formally verified software is software for which there exists a mathematical proof that it does something, (and often that it only does that thing). Using this method, entire programs may, in theory, be tested with the same certainty that mathematicians prove theorems. However, in practice, formal methods are used to verify smaller but especially vulnerable or critical pieces of a system, like operating systems or cryptographic protocols and there are claims that formal methods are not performant[8]. Formal methods will make systems more trustworthy over time but the possibility of vulnerabilities especially in complex interconnected systems still exists. Furthermore, the quality of what was built may be "perfect" but whether what was built was what the user wants or expects is an entirely different question.

While it may be possible to prove epistemologically that an entire system is "correct", if only the vendor can assert this and there is no method of verifiying or auditing that assertion, then the information asymmetry between the parties still makes the software untrustable.

Institute for Strategy Resilience & Security
University College London

CodeThink

A Manifesto for Trustable Software

Guiding Principles for Trustable Software

Just as blockchain technology[9] is redefining trust in transactions as a technology, the software industry needs to redefine its approach to trust in building software. Deploying the software and hoping that it is trustworthy in use, however good the build, is no longer appropriate for many applications if software of unknown provenance is added to a critical network. Trustworthiness in software may evolve over time but, on its own, will always be a subjective measure.

A trustable software process is not a prescribed methodology. Rather it sets the principles and the supporting evidence required to support that the principles have been met (Table 1). As long as the principles of trustable are evidenced, then the method by which the evidence is produced is not subject to restriction. However, in reality, unless a software package is extremely simple, requirements, testing etc, will likely need to be automated in order to ensure that there is sufficient quality of record keeping.

TABLE 1: CORE PRINCPLES OF TRUSTABLE

Core Principles of Trustable	Audit Evidence Required
· We know where the code comes from · Does the code do what it is supposed to do and does it not do what it is not supposed to do? · We know how the code was built and tested prior to deployment · We can reproduce it exactly from source code · We can update the code and be confident it will not break or regress	· Evolution of problem/scope (requirements, standards and verification criteria) · Evolution of solution/architecture · Selection/production of software (including tests) · Evolution of tests, test results and satisfaction of validation criteria · Traceability back through all previous phases · Maintenance changes/upgrades are being applied to the whole pipeline, not just code

This enables industries and companies to adapt the concept to their own particular planning methodologies and processes. It is a process which sets out to evidence, at every stage of software development, that the principles of trustable have been met. The degree of information required between parties is flexible beyond some minimum criteria. Hence the process can be used in many different circumstances. By being as non-prescriptive

as possible in the way that trustable is achieved, the process fits with different philosophies, methodologies and technologies. This is vital for its adoption and adaptation to future technological developments in the industry.

Just as accounts are asserted to be "true and fair" and are specified a way such that they may be audited or inspected, so we could monitor software production, perform a series of checks, produce metadata to evidence those checks, keep a record of that information and provide an output in an agreed format to stipulated parties.

The Components of a Trustable Software Process

A trustable software process sets out to capture the evidence that proves that the principles of trustable have been met. Figure 1 shows how this process could operate in practice. Drawing an analogy with the construction industry, trustable software needs plans (requirements), regulations to be met (standards, tests and data to evidence those principles have been met), recorded information about who is building each piece and signing it off (developer and approver identities), a record of what was built and how it was built (final code and its development branches) and the sign-off process by a building control officer (testing results and release to deployment). These records would also be available to a third party inspector should there be a requirement to inspect the records at any time.

Software Requirements Specification

In a trustable process, any code must be matched to a requirement whether it is an original requirement or a new or modified requirement as the project progresses i.e. there is evidence of evolution of the problem/scope and evidence of evolution of the solution/architecture. Evidence that maintenance changes/upgrades are being applied is also required. A new piece of code must relate to a new requirement or an identified vulnerability or issue. The mechanism for capturing the original agreed requirements and subsequent approved changes is the software requirements specification. The agreed specification and updates are part of the evidence required as part of the trustable process.

The trustable process is independent of the methodology used for the project: both a traditional waterfall approach or Agile Methodology in all its various forms can be accommodated, as long as evolving requirements are documented and the code produced is against those requirements.

Process Information Capture

The data relating to a trustable software process needs to be captured frequently and in a way that it cannot be altered retrospectively. This would be achieved via an immutable logged audit trail around the development of the code available to the purchaser and to any required 3rd party.

Two primary methods are available. The first, echoing current regulatory practices, requires a third party – a regulator – to be the trusted holder of the audit trail. Hashes of information would be continually deposited with the regulator enabling them to reconstruct the chain of change. A second method uses blockchain technology incorporating chained hashing of the trustable data captured – a process that can be decentralised, with a distributed ledger. In both cases, hashing transforms data of any size into short, fixed-length values. Transactions stored onto blockchain become increasingly more difficult to alter over time increasing the difficulty and cost of possible fraud.

The trustable software process would record the agreed information required which relates to all the events that contributed to the development branches of the final code and send this to the agreed immutable audit log, which can be examined by anyone who is authorised e.g. the purchaser, a third party or regulator. The exact detail of what is recorded and who can view what would be agreed via the taxonomy.

FIGURE 1: EXAMPLE OF AN END-TO-END TRUSTABLE PROCESS

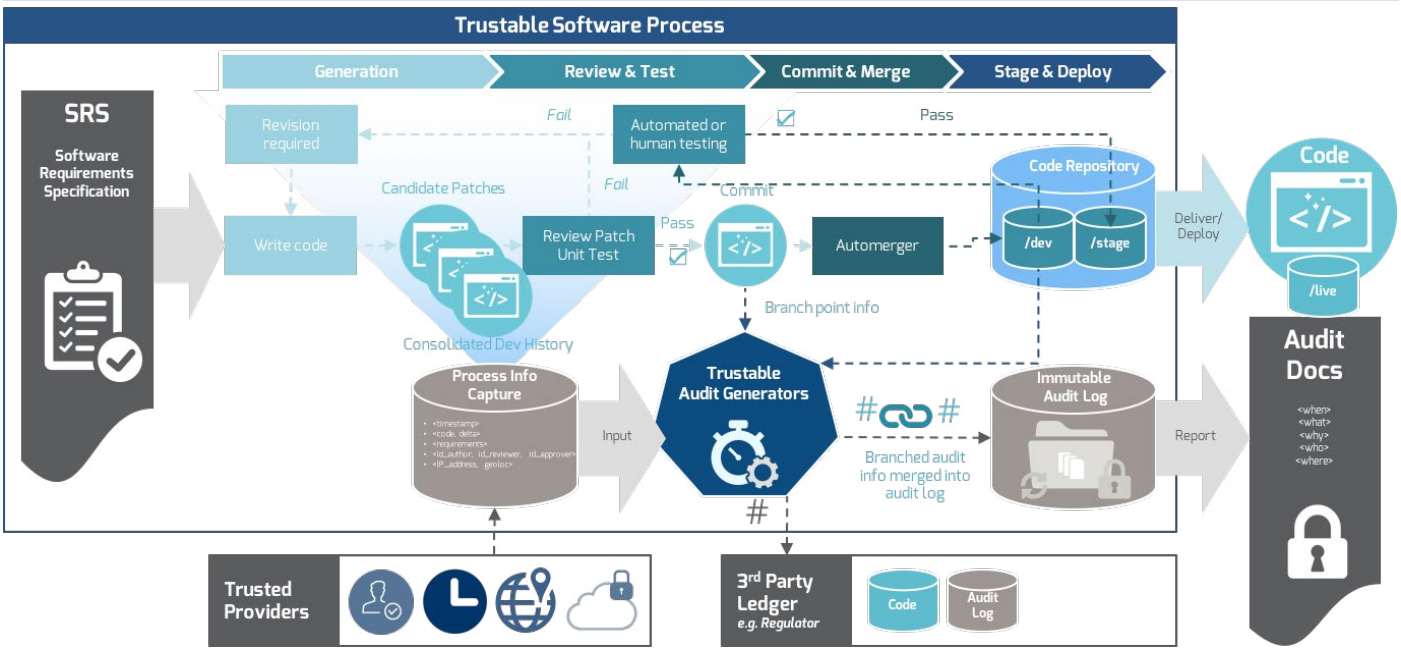
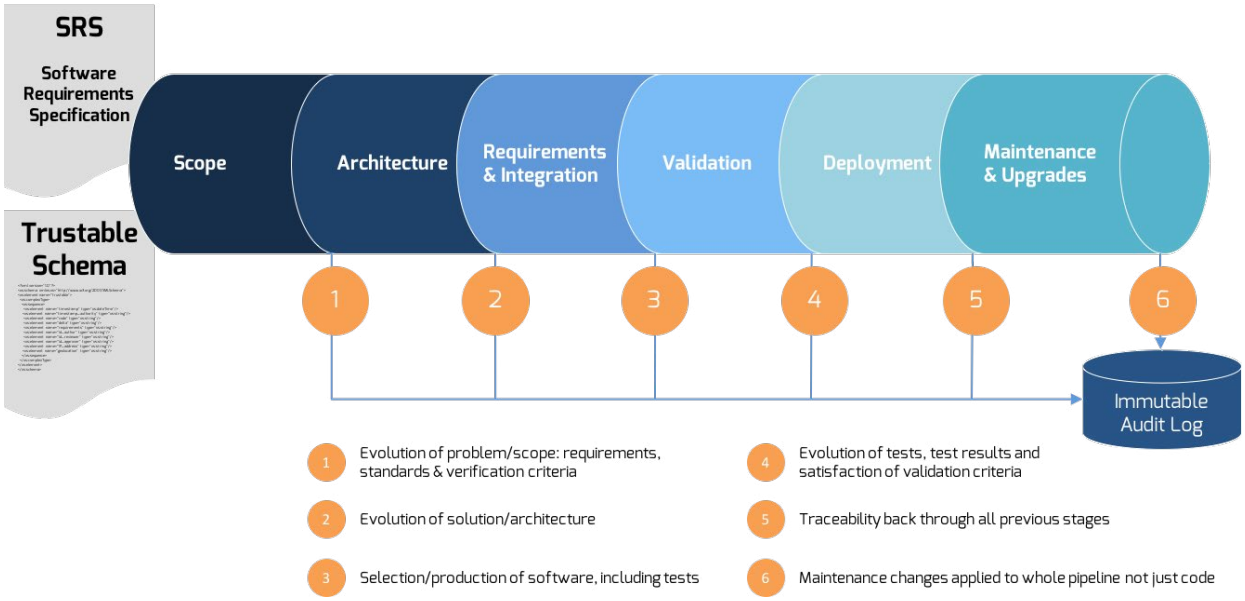


FIGURE 2: EVIDENCE IS COLLECTED AT EVERY STAGE OF THE TRUSTABLE PROCESS



The process for the generation of the meta-data is shown in Figure 2. Metadata, defined by the taxonomy, is output at each stage and stored in the immutable audit log.

External Audit Data and Audit Mechanisms

An extract of the trustable process data can be sent from the software vendor to a regulator, third party or purchaser as required. This data must be linked to the immutable log enabling an approved 3rd party to trace the provenance of the code back through the development process, much the same way as an auditor auditing financial accounts can request and examine individual records if required. In trustable, an "auditor" would be looking for discrepancies such as source keys which do not match up, built artifact checksums that do not line up, builds that are not reproducible and patches that did not pass in testing.

Trustable Taxonomy and Ontology

A trustable software process requires both a taxonomy and an ontology. The evidence needs to be routinely captured in order to assert the provenance and testing of code. The "rules" about what needs to be captured when is agreed prior to commencement of coding. As best practice, one would envisage that the generated data for a trustable process is machine readable in order for the evidence to be analysed using automated tools.

A taxonomy is a structured framework which specifies how to formally describe the data required to evidence agreed trustable principles. For example, HM Revenue and Customs specify

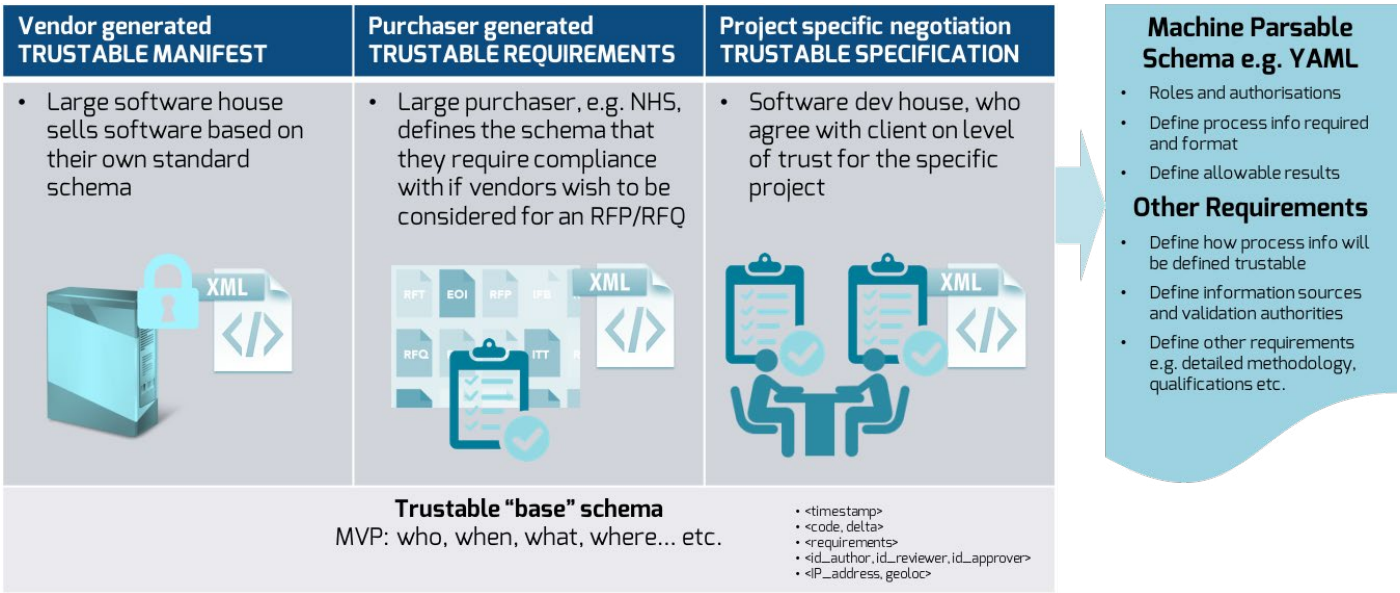
XBRL requirements for the filing of company accounts[10], which enables the data to be processed automatically by software. An ontology is a formal naming and definition of the types, properties, and interrelationships of the entities. In trustable software this would be used to define the basic evidence produced and enable a means of identifying data between the parties that could not be shared publically, e.g. a user identity which can be traced back via company records to an individual developer.

Because of the differing nature of requirements and power between vendor and supplier, a "one size fits all" taxonomy would be impractical just as building regulations for a skyscraper would be inappropriate for a family home or vice versa.

There will likely be a number of options for trustable taxonomies (Figure 3). Three common scenarios would be:

1. A vendor generated trustable taxonomy suitable for Business to Consumer models or Business to Business models where there are a large number of customers. The vendor would create a standard taxonomy which would be supplied to an agreed third party or regulator where required.
2. Relationships where there is a large degree of purchasing power e.g. the NHS The purchaser will specify the taxonomy which needs to be followed as part of the tendering process along with other requirements.
3. The vendor and purchaser would negotiate a mutually agreed taxonomy depending on the requirements of the project.

FIGURE 3: EVIDENCE IS COLLECTED AT EVERY STAGE OF THE TRUSTABLE PROCESS



Conclusions: Applying Trustability to Software

Resilience (noun.)

The enduring power of a body or bodies for transformation, renewal and recovery with the flux of interactions and flow of events.

Towards **Trustable Software**

Applying Trustability to Software

Today, software purchasing and use relies largely on a combination of reputational and experiential trust. Purchasers largely rely on brands, recommendations and experience in use. A reputable brand has value because it implies that others were satisfied. Reputational trust is achieved through recommendation or the collective opinion of others. Experiential trust derives from successful use of a product or service to the point where it is considered trustworthy, regardless of other evidence.

While issues with trust in software have long been recognised, the default approach of the industry has been to focus on improving the quality and trustworthiness through better code, new programming languages, greater attention to bugs, and more frequent and improved security patches. Though logical for providers, this is an subjective and non-systemic response.

Various established approaches have attempted to create arbitrary standards for *trusted* and *trustworthy* software, but these are application-specific and apply to systems in a particular state of delivery. A holistic solution is required, which provides for a far higher standard of evidence of the whole process by which a system is built, operated and maintained to stated requirements and standards agreed at the start of the project, and adapted as the requirements change.

Supporting Learning and Resilience

Trustable processes are not infallible, rather their efficacy derives from strongly incentivising participants in the value chain to act professionally and responsibly – or suffer sanction. It is the combination of the need to provide proof that the process is being followed, the provision of key data to interested and independent parties and the subsequent auditing of that data that encourages integrity, and, through compliance, evolves trust towards the standards required for society's needs.

As risks can never be totally eliminated, a trustable process needs to maintain confidence, even when the process has not delivered the desired result, by providing resilience to failure. Resilience, in the case of a trustable process, is therefore not just about a government, regulator, company or other body responding to an individual event or disaster but the ability for the system to respond to that event systematically. If there is a disaster, the trustable process maintains that trust by providing the relevant authorities with data and documentation to help investigate the root cause. Once the root cause is known, action can be taken to eliminate or reduce the same potential risk in similar conditions. Trustable processes give people confidence in a product or service even in the aftermath of a disaster.

Institute for Strategy Resilience & Security
University College London

CodeLink

Towards Trustable Software

The proposed approach of trustable software described here adds transparency to the design, development and testing process for software code, and generates and collects together assurances on each piece of software. Snapshots of software at key points in its development are accompanied by a linked immutable audit log containing key information about the process by which the software has been produced, installed and maintained.

A downstream party relies on the producing party to capture the evidence correctly, using chained hashes or "blockchain", to ensure that it is computationally unfeasible to alter the code-log relationship. This metadata can be made available to any third party, improving transparency and enabling downstream customers to assess the degree to which software can be trusted.

In turn this places pressure on all of the entities in the design, development and deployment chain to act according to standards and leads to trust in exactly the same way that it does in construction, pharmaceuticals and financial reporting.

How a trustable software process would work in practice needs to be explored and discussed further with a view to generating a reference implementation. The generic trustable software process that we present in this paper is a first step in this direction.

We invite comment and feedback from all stakeholder parties with a view towards a robust debate on the role that trustable may play. Please visit our website at **www.trustablesoftware.com** to include your comments and to contact us.

References

1. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016 http://www.gartner.com/newsroom/id/3598917
2. Police and Criminal Evidence Act 1984 (PACE) codes of practice https://www.gov.uk/guidance/police-and-criminal-evidence-act-1984-pace-codes-of-practice
3. How Many Millions of Lines of Code Does It Take? http://www.visualcapitalist.com/millions-lines-of-code/
4. Gartner Says Global IT Spending to Reach $3.5 Trillion in 2017 http://www.gartner.com/newsroom/id/3482917
5. Software Engineering Body of Knowledge https://www.iso.org/standard/33897.html
6. Trustworthy Software Foundation http://tsfdn.org/ts-framework/
7. Formal Methods - an introduction to the topic https://users.ece.cmu.edu/~koopman/des_s99/formal_methods/
8. https://www.cs.princeton.edu/~appel/certicoq/
9. How Does Blockchain Technology Work? https://www.coindesk.com/information/how-does-blockchain-technology-work/
10. HM Revenue and Customs XBRL guide for UK businesses https://www.gov.uk/government/publications/xbrl-guide-for-uk-businesses/xbrl-guide-for-uk-businesses

Towards **Trustable Software**