<u>Why Aren't Software Systems Trustworthy?</u>

By Tom@Gilb.com

Trustworthy Defined: "worthy of trust or belief." (web dictionary)

Software System defined: We may wish to focus on the issue of software, for various reasons, but it is too narrow to just look at the software *alone*. The software is *one* component in a system comprising other fallible system elements *such as* individuals, organizations, stakeholders, hardware, communications, laws, economics, data, databases, cultural mores, and motivations. So it would be dangerous to seriously discuss 'software trustworthiness' alone. Microsoft sometime uses the concept 'trustworthy **computing'** [1] to make this point.

Even if the pure 'software' were perfectly trustworthy, any one of the *other* non-software elements would be a sufficient 'weak link'; and could alone, or in combination with other system elements, cause the system to be perceived as untrustworthy by a stakeholder or user. The *system* user or stakeholder would not necessarily be able to distinguish, nor would they care, about whether the *software* element caused their system to be untrustworthy. Consequently we are *forced to consider the entire system as a whole*.

We cannot *usefully* look at the software element alone. So for the purposes of this paper the 'software system' means 'the software and all other system elements that can influence trustworthiness as perceived by a user or stakeholder'. We will simply refer to the **system**, meaning the set of elements (or components) including software, that can determine the objective and subjective trustworthiness of a system for a defined stakeholder's point of view.

System Trustworthiness: defined:
A defined system is 'trustworthy' from the point of view of a *defined* stakeholder or user. The same system states may be evaluated as 'untrustworthy' from some points of view, and 'trustworthy' by others.

Trustworthiness is a *relative point of view*, not a system state independent of such points of view. The 'Trustworthiness View' is subjective. That is, any given observer is at liberty to define a set of system states they consider trustworthy or untrustworthy. We cannot prevent them from having those perceptions.

We can identify their perceptions, make formal agreements about their perceptions, and attempt to engineer and operate a system to be trustworthy in accordance with those stakeholders we care to serve. We can also choose to ignore, or give lower priority to, the perceptions of stakeholders that we do not care to serve, to prioritize, or who are uneconomic; or who have requirements outside of the state of the art.

Belief and Fact.

We need to distinguish between trustworthiness as a matter of *belief*, and as a matter of *fact*. Stakeholders may *feel* that a system is trustworthy due to information or experience, when in fact it is not trustworthy according to their standards. They just don't know that, yet. We can distinguish between 'Trustworthiness Perception', and 'Objective Trustworthiness'.

The 'state' of system trustworthiness:

As Microsoft [1] is clear about, there is no one single system attribute that alone determines 'trustworthiness'. If there are S stakeholders that we choose to consider the opinions of, and N states that any one of them might require 'present' (or absent) in order to consider the system trustworthy, then we as system engineers must consider (in requirements, design, testing and operational evaluation) all N states in order to evaluate whether all our S stakeholders will consider the one system 'trustworthy'. S could easily be a number in the range of 35 to 350 stakeholder classes (then we have individual variation with a class like 'User'). N, the number of trustworthiness states we must manage, can easily be dozens to more than thousands for a single stakeholder, and thousands to more than millions for the entire stakeholder community for a large system.

It is the task of the systems engineer to determine the trustworthiness needs of all potential serious stakeholders. Then from the stakeholder needs, the engineers must determine which of those needs the system can and will *actually* attempt to satisfy at a given time, under given conditions. These can be specified as the system requirements (for 'trustworthiness'). A suitable comprehensive language for such specification is defined in Competitive Engineering [2].

The usual narrowness of conventional software engineering for defining requirements (e. g. 'functions' and 'use cases') is completely inadequate for describing many of the central trustworthiness characteristics, such as qualities (like security and reliability), costs, constraints and many other aspects (covered in detail in [2]). It is unconditionally necessary to take a 'systems engineering', NOT a *software* engineering, perspective in even *defining the problem* of software trustworthiness. Anything less will immediately fail to deliver any reasonable stakeholder notion of trustworthiness.

This is one of the core problems of the software trustworthiness – the failure to treat it as a *systems* engineering problem, and failure to define the major trustworthiness attributes of systems, such as the multitude of qualities (e.g. security, availability, portability, usability, maintainability, connectivity and many more), quantitatively.

Most software engineers have no training or ability to quantify qualities at present. Their vocabulary and culture are massively inadequate for dealing with the problem. A systems engineering culture is a necessary minimum. This is partly because a systems engineering culture *can* deal with the critical qualities and costs associated with trustworthiness. Partly because the discipline for *software* trustworthiness must always deal with the *non-software elements* of the system that the software is a *part of*.

<u>The Core problem</u>: (based on definitions above!)

1. Software culture does not have adequate intellectual tools for dealing with the necessary 'system' (software plus all other related components) problem.

<u>Recommended Solutions:</u>
1. Require that software-dominant systems be engineered, and operated, using a sufficiently rich *systems engineering* discipline.
  - it is 'sufficiently rich' when the defined stakeholders needs we have chosen to satisfy, are in fact satisfied for the life of the system.

2.

<u>Suggested principles for Trustworthy Software:</u>

1. Trustworthy Software is dependent on all *non-software* elements of the software host system. (people, organizations, hardware, data, laws).

2. Trustworthy Software is based on the subjective perception of a potentially large number of different stakeholders.

3. Trustworthy Software is determined by a potentially large number of system states (conditions, levels) being met.

4. Trustworthy Software is realistically a matter of what we can afford, what is technically possible, what is politically necessary, and what pays off.

5. Trustworthy Software cannot be built using current software engineering disciplines – because it is not about programming, and programs – it is about complex *systems*.

6. Trustworthy Software requires the management of all trust-critical performance characteristics, including all variable trust-critical qualities. This requires quantification of the problem and measurement of the solutions – *engineering*.

7. Trustworthy Software will come about gradually by the conscious and intelligent responsible stakeholders demanding well-defined 'trustworthiness' results, and refusing to accept or pay for less than agreed.

It will *not* happen with the help of computer scientists, or software engineers – any more than it will happen using nuclear physicists or mechanical engineers. The disciplines are too narrow to even *define* the *real* problem, let alone *solve* it. At best they might ultimately learn to make more trustworthy software *components* – but not unless they adopt a systems engineering approach to their work.

8. Trustworthy Software is a management problem, and management responsibility.

The *technical* means are already there. Management has *not chosen* to use them. Management has probably not been well advised by their technical advisors – who may be software people, not systems engineers – and therefore culturally incapable of giving comprehensive enough advice.

9. Trustworthy Software is to some degree an economic matter. We can increase the state of the present art, and that will primarily allow us to deliver more trustworthiness for our limited resources.

10. Trustworthy Software is also a matter of clear notions of minimum acceptable system conditions (at any cost) that might be determined by law, regulation, contracting, or other devices. This is primarily a management responsibility, not a technical problem.

Conclusion:

Trustworthy Software is a technical opportunity awaiting serious management action to exploit known technical methods.

--- end of Jan 7 2006 draft by Tom@gilb.com ----

References

1. Microsoft, http://research.microsoft.com/ur/us/twc/default.aspx

"Trustworthy Computing

No issue is currently of greater importance to Microsoft and our industry than trustworthy computing. Amid increasingly frequent and sophisticated network attacks, users expect their systems to remain resilient and available. They expect data to remain intact and confidential at all times. As they increasingly use computers to manage information important to their everyday lives they expect and demand control over access to and use of their personal information. Ultimately, it is essential that computers perform as expected and that users enjoy a consistently trouble-free computing experience.

These are large goals, not only for Microsoft but also for the industry as a whole. It would be very reassuring to all if there were a single step or strategy that could achieve these goals. But in reality, trustworthy computing is complex. Changes in the way software is designed, built, and tested are critical. We must better understand the types of failures and threats to which any particular piece of software is vulnerable. We must better anticipate the habits and inclinations of different types of users and make the right assumptions about their desire and ability to adjust or maintain the configuration of their systems. And we must effectively train designers and developers of software to focus on trustworthiness and to have the essential knowledge to build consistently trustworthy software.

Computer security and cryptography have been subjects of valuable academic research for some time. But this is only a part of the larger set of issues that define trustworthy computing, and even computer security often receives only very limited treatment in computing curriculum. Academic research has made valuable contributions to Microsoft through Microsoft Research's Trustworthy Computing Advisory Board. Now, there is an urgent need to raise awareness of the full range of trustworthy computing issues in academia and begin to develop the kind of innovative approach and materials that can place trustworthiness at the center of the computing curriculum."

2. Gilb, Tom, **Competitive Engineering**, A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, ISBN 0750665076, 2005, Publisher: Elsevier Butterworth-Heinemann.

Version January 7 2006 (1st Draft)

.