1. Security

The development and proliferation of aircraft internal networks tied with air traffic management and airline operations as presented for E-enabled aircraft [51] and NextGen [3, 52] systems bring security to the forefront, because security violation may adversely affect flight safety.

Security is understood primarily as the means to protect the system assets but this traditional view, entirely justified and warranted, obscures the scientific approach to security viewed as a system property.

When one looks at the definitions of security in established standard glossaries, such as [47] or [53], it becomes immediately clear that in none of these documents security is defined as a system property. Nearly all these definitions refer to security as a state, as opposed to ability. A definition of security as a system property must imply that one wants to measure it. In this regard, just like for several other system properties, the definition should include a phrase “the extent to which” or “the degree to which.” Consequently, the definition of security adopted from [47], reads as follows:

Security. The extent to which information and data are protected so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.

Reading the above definition, one can realize that it is related to information security in three different ways, capturing its three essential aspects. First, it pertains to protecting information from unauthorized
reading, which means Confidentiality. Second, it concerns protecting information from unauthorized modification, which means keeping its Integrity. And finally, what the definition captures quite well is the fact that the secure system must be not only protected against unauthorized access and threats but also be accessible to those authorized, which means Availability. This is the real meaning of the often quoted acronym C-I-A = Confidentiality + Integrity + Availability [94].

1.1. Basic Concepts

Since security is so tightly related to safety, a model presented in Fig. 1 can illustrate fundamental concepts of both properties. Both safety and security concern external factors that can interact with the system from its surroundings, commonly called the environment. The external influences are included as an unknown factor in the design, but because their detailed characteristics are unknown they are grouped into disturbances. Unintentional disturbances are known as hazards and are taken into account in processes assuring safety. Intentional disturbances are called threats and commonly used by attackers, so have to be included in processes assuring security. Human errors, also called mistakes, obviously contribute to the problem. All these factors are illustrated in Fig. 1 showing how external events can affect both safety and security.

Vulnerabilities, from the security viewpoint, are analagical to faults in safety domain, since they both represent potential defects in the design or implementation which, when activated (fault) or exploited (vulnerability) can lead to a failure (regarding safety) or a breach (regarding security). This is shown schematically in Fig. 1. Both respective concepts overlap, since they may interact with each other.
Figure 1. Hazard and Threat impact

Figure 2. Extended view of a control system
Threats can be treated as events analogous to hazards, except that, unlike hazards which are incidental, threats are intentional, and result in attacks executed by an attacker. This is illustrated in Fig. 2. In this architectural diagram, similarly to what is shown in Fig. 1, threats can exploit vulnerabilities (just like hazards activate faults) and lead to breaches (analogous to failures in the safety domain).

Even in this context, computer security has many different aspects and may be viewed from diverse perspectives—so the concept of security in aviation software is no exception. From the software engineering perspective, there are essentially two viewpoints of system and software security. Just like in the domain of system safety, we are first concerned with designing systems for security to ensure later secure system operation.

2.2. Software Security in Aviation

Disappointingly, the existing aircraft system certification guidance never considered potential malicious attacks and thus is does not address airborne networks and data security issues from the safety perspective. Independently, the RTCA committee on Aeronautical Systems Security, SC-216, DO-326/ED202, completed Airworthiness Security Process Specification guidance in 2010 [66]. However, the committee work focuses on processes, methods and considerations, staying away from engineering and scientific approach based on measurements and analyses. Often the terminology used in the documents contradicts that used by scientific community. As an example, the aviation community uses term “measures” to represent the procedures, approaches, and tools used to mitigate the security threat (which in common language are “mitigation measures” or “countermeasures”).

There is an evident challenge to quantitatively characterize the above properties. However, there is a noteworthy practice, established in the safety domain, to assign categorical metrics. Clear and unambiguous determination of the metric categories would allow developing effective measures leading
to modeling of security for specific assets. However, the measurements would need to be based on the developers' experience and collection of well scrutinized historical data. The resulting measurement (category) would be representing the value, while the accuracy is defined by the category boundaries. Due to the subjective nature of measuring it might be useful to explore fuzzy metrics.

Similar to safety domain, which defines risk as combination of probability of hazard and severity of the potential consequences, security domain also uses these two concepts. The metrics used for such security components as attacker profile, vulnerabilities, operational conditions, or threat conditions are defined in terms of likelihood (or probabilities). These metrics is rather ordinal than numerical. Metrics like likelihood of attack, probability of exploitation, probability of exposure, likelihood of successful attack are very subjective and collection of data is an obvious challenge. The typical categorization of the attack likelihood is presented below:

- Frequent - anticipated to occur routinely in the life of each asset.
- Probable - unlikely to occur during a routine operation but may occur a few times in the life of an asset.
- Remote - unlikely to occur during its total life but may occur several times in the total life of an entire group of this type of assets.
- Extremely Remote - not anticipated occurring during its total life but which may occur a few times in the total life of entire group of this type of assets.
- Extremely Improbable - not anticipated occurring during the entire operational life of all assets of this type.

The obvious questions are: What does it mean “routinely”, “unlikely”, “not anticipated”? How much is “few” or “several”? There is no agreement on specific numerical values and assessment of these likelihoods is difficult. Similarly, typical categorization of a successful attack impact consequence may be
a failure as defined by Tab. 1, i.e. catastrophic, hazardous, major, minor, or no effect. However, these definitions use not quite precise terms. Again, the questions are: what is “slight”, “significant” or “large”?

Using similar categories we can classify vulnerability level of the asset (e.g., highly vulnerable, vulnerable, marginally vulnerable, not vulnerable) and the effectiveness of the applied countermeasures (e.g., highly effective, effective, marginally effective, not effective).

The current state of the art in aviation security [66] uses the term "characteristics" to denote "property" used in this paper. The aviation community agrees on set of parameters defining Security property (S) under specific Operational Conditions (O):

- $V$ - Vulnerability of the asset.
- $E$ - Efficiency of applied countermeasures.
- $I$ - Impact and consequences of a successful attack.

There has been little discussion on how these parameters should be measured, less even what the models are reflecting their interrelations. Considering the discrete and categorical nature of the above parameters, there is possibility to create mathematical model of security S in a form of a discrete function:

$$S = f(A,V,E,I, O)$$

Evidently, higher categories of parameters $A$, $V$, and $I$ would have a negative impact and thus decrease the security, while higher category of parameter $E$ would have positive impact on security as the system property. Based on historical data and assessment of security an attempt can be made to identify the $f()$ function.
3.3. Security Process

System and software security is a significant concern in the design and development process. Subsequently, once the system is released, security must be continuously monitored and maintained during system operation. This is schematically reflected in a simple waterfall model (Fig. 3).

![Figure 3. Security development cycle](image)

These two viewpoints of security (development vs. operations), although refer to the same system property and are significantly overlapping, create two different perspectives and are addressed separately. The issue of system development for security received a lot of attention in recent years, but the theory behind it is not well developed, yet. The concepts are still vague and various stakeholders often differ on the specific practices and even on the use of fundamental terms.

A general view on including security concerns in the software development life cycle has been presented in several reports [54-57]. In this section, we present a summary of current views on dealing with security issues in the major phases of the waterfall model.
System requirements addressing security issues are critical for the majority of modern interconnected systems. A report on Security Quality Requirements Engineering (SQUARE) Methodology developed in the Software Engineering Institute, states [58]:

“A recent study found that the return on investment when security analysis and secure engineering practices are introduced early in the development cycle ranges from 12%- 21%, with the highest rate of return occurring when the analysis is performed during application design. NIST reports that software that is faulty in security and reliability costs the economy $59.5 billion annually in breakdowns and repairs. The costs of poor security requirements show that there would be a high value to even a small improvement in this area. By the time that an application is fielded and in its operational environment, it is very difficult and expensive to significantly improve its security.”

SQUARE is a nine-step process that helps organizations to build security, including privacy, into the early stages of the production life cycle. The long term goal of SQUARE is to integrate security considerations into the early stages of the development life cycle. The process starts when the requirements engineering team and project stakeholders agree on definitions and identify security goals. Next, the security artifacts and documentation are developed including system architecture, attack trees, use and misuse cases, and essential assets and services. Subsequently, a structured risk assessment and selection of elicitation techniques allows developers to elicit security requirements. The following steps involve categorizing and prioritizing these requirements for management’s use in making trade-off decisions. Finally, an inspection stage ensures the consistency and accuracy of the security requirements that have been generated.

The essential guideline document for secure software design [59] addresses the design activities by focusing on architectural design and detailed design, defining detailed design as “describing each
modeling security properties, and
• using secure design patterns.

Modeling stresses the importance of grasping the attacker’s perspective and applying at this stage: use cases (misuse and abuse), attack trees and threat models. The recommendation is that threat models should be created as early as possible in the life cycle and should be revisited as the system evolves and the development progresses.

Defining software design patterns as “a general repeatable solution to a recurring software engineering problem”, following earlier work the guideline distinguishes between architectural-level and design-level patterns [60]. The architectural level patterns focus on the high-level allocation of responsibilities between different components of the system and define interactions between those components. The document lists three essential patterns of this type:
• Distrustful Decomposition,
• Privilege Separation, and
• Defer to Restricted Application.

Design-level patterns address internal design of a single high-level component and include the following:
• Secure Factory to separate the security dependent logic from the basic functionality of the reference object.
• Secure Strategy Factory to provide easy to use and modifiable method for accessing the appropriate objects.
Secure Builder Factory to separate the security dependent rules from the actual creation of an object.

Secure Supply Chain of Responsibility to decouple the logic that determines trust dependent functionality from the part of an application requesting the functionality.

Secure State Machine to provide clear separation of security mechanisms from user-level functionality, and

Secure Visitor to allow components to lock themselves unless the visitor supplies the proper credentials.

There is a vast amount of coding standards for secure software, beginning since about 2002 [61-64]. One of the recent ones [62] provides a collection of recommendations adopted from other standards. Two others are worth mentioning, because they include more specific recommendations as checklists.

The second edition of the CERT C Coding Standard [63] provides the rules for secure coding related to C language constructs and the language environment, including: the preprocessor, declarations and initialization, expressions, integer, floating-point, character and string data types, arrays, memory management, input/output, signals, error handling and concurrency. Overall, above a hundred rules are listed and described, which can be used as a checklist in addressing coding errors that are causes of current C language vulnerabilities. The document prioritizes them by severity, likelihood of exploitation, and remediation costs. Each rule includes examples of insecure code as well as secure, alternative implementations compatible with earlier C standards.

Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving
application security. OWASP's projects cover many aspects of application security with many active projects building documents, tools, teaching environments, guidelines, checklists, and other materials to help organizations improve their capability to produce secure code. One of the projects resulted in developing Secure Coding Practices [64] defining software security and risk principles and identifying secure coding practices checklist.

Although there may be several compatible definitions of software testing, the essential issue in testing is always checking whether the software product meets the software requirements. This is how the test cases are built, essentially one or more per requirement. Thus, testing software for security does not involve any unusual techniques; it is just applying a set of testing procedures to unveil if the product has been built right.

This is the approach taken in [65], which adopts the view that “testing is a process of comparing the state of a system or application against a set of criteria.” In this approach, the document advocates performing specific activities during the life cycle, including requirements, design and code reviews and walkthroughs. Actual testing of the code is done with unit and system tests, as well as using penetration testing after deployment. The document lists specific tests for website testing, in the following ten categories:

- Configuration and Deploy Management
- Identity Management
- Authentication
- Authorization
- Session Management
- Data Validation Testing
- Error Handling
• Testing for weak Cryptography
• Business Logic
• Client-Side Testing

The document also makes suggestions on using specific testing tools and writing the test reports.

1.4. Techniques and Tools

It is widely recognized that the theory of computer, information or software security has not developed yet, except for some narrow areas, such as cryptography [67]. Thus, the techniques and tools used in software security are mostly experimental and not well established at the time of this writing.

Obviously, the issue is in the center of attention and multiple projects have been undertaken, some of them summarized in [68] and [69] reports. The current situation is well characterized by a statement from [68] that “There are some very sophisticated approaches […] that can be used to assess and reason about the security of current systems, but they are not widely available today in the form of developer tools.” Among those approaches the author lists: model checking, cryptography, randomization, type theory, and game theory.

Report [69] reviews a number of theoretical approaches to security assessment, including attack and other types of trees, Boolean logic driven Markov processes, and several others focusing mostly on security risk analysis. Then the report proposes a method, based on BBN combined with probabilistic relational models. The problem is defined as finding “a security assessment method so that decision makers only need to describe their systems in order to obtain the assessment of their enterprise architecture.”

Since there is only a limited amount of data on computer or software security, and the measurement techniques are imperfect, mostly due to the lack of firm security theories, the most successfully studied
techniques, thus far, are those dealing with insufficient information and uncertainty based methods, such as BBN [70], fuzzy sets [71], rough sets [72], and similar [73]. Their applicability depends on the exact objective, which is related to the focus in architectural description of a system.

With the lack of security theory, the two remaining approaches to study security are simulation and experimental techniques. When one looks at the model depicted in diagrams in Fig. 4 and 28, it becomes clear that there are three essential ways of behavioral modeling of security, based on the individual components of the model, which involve the attacker, the system itself, and the attack surface:

1) how the attacker behaves and what are the threats he poses;
2) what kind of damage or degradation can be done to the system; and
3) how the attack surface is impacted in real time.

Consequently, three different kinds of tools have been developed and used, with functionality focused essentially on one of these areas, but often overlapping:

1) Threat Modeling Tools, which are simulators allowing a modeler to build a model of prospective attacks and draw conclusions on potential threats:
   - Microsoft SDL Threat Modeling Tool [74]
   - ThreatModeler [75]
   - Trike [76]
   - PASTA [77]

- Security Assessment Tools, which are simulators based on discrete event models allowing model building to imitate system behavior under hypothetical security breaches:
  - Windchill [46]
  - Mobius [78]

- Network Security Assessment tools, which, as opposed to off-line tools in two categories mentioned above, are real-time penetration tools that can probe various aspects of the target
system, addressing specific objectives, and collect data that can be used for estimating actual threats:

- Metasploit [79]
- Wireshark [80]
- and many others (for a comprehensive list, see [81])

1.4.1 Threat Modeling

Threat modeling is a relatively well established technique [82-84] that assists software engineers to identify and document potential security threats associated with a software product. It provides development teams a systematic way of discovering strengths and weaknesses in their software applications during the Security Design Lifecycle (SDL). Microsoft’s SDL Threat Modeling Tool offers automated analysis of security threats to system [84]. Building the threat model is based on Data Flow Diagrams.

Data Flow Diagrams (DFDs) are a well-established technique used to visualize how a system processes data [85]. Critical in security modeling using DFD’s are trust boundaries on paths between the system entities. Transition from one trust boundary to another has to validate the following six threat types: Tampering, Information Disclosure, Denial of Service, Spoofing, Elevation of Privileges, and Repudiation.

For an aircraft system, the security risk assessment shall focus on attacks that may affect safety of the aircraft. However, the complexity of most of the aircraft systems is such that it is not easy to assess their safety. The standard approach is to identify the conditions that control the airworthiness of the aircraft, and to classify the severity of the impact of all other events in terms of these top-level conditions [26]. The safety analysis considers failures and defects, and a chain of adverse events and conditions causing
potential safety hazard. The effect of a hazard is a failure condition, with an adverse effect on the aircraft, the crew, and the passengers. In a complex system such as an aircraft, one can identify a top-level failure condition composed of combination of item defects or failure conditions along with other operational conditions or events.

A security attack also involves a chain of adverse events and conditions. In this case, by analogy to failure condition, a threat condition will occur through an information security attack. Such system can be presented in form of data flows with appropriate threat boundaries between the processes and thus allowing applying the proposed threat modeling technique.

The analysis would allow designers to apply security measures (deterrent, preventive, detective, corrective, recovery) appropriate to the identified threat. These measures, implemented on threat boundaries when results of analyses identify the specific threats, can be procedural (procedures, policies, and people) or technical (functions, systems). A snapshot of the case study presented in Fig. 4 captured using the SDL Threat Modeling tool demonstrates the applicability of threat modeling to reveal some of the threat conditions.

The Microsoft SDL Threat Modeling Tool is a tool to analyze graphically threat models expressed using DFD diagrams. Once the model is created, the tool can analyze it to identify threats, allows the user to enter additional information describing the environment and generate reports. The critical feature of the tool is to display potential threats generated by the DFD and to suggest applicable mitigation strategies.
By selecting the potential threat, the tool provides a section where the developer can explain the impact of the threat and select possible solution to mitigate it. A completion bar is displayed when a solution to mitigate the threat is entered by the user. Considering other useful features of the tool allowing for describing environment, bug tracking, recommendations for fuzz testing, and generating reports, the tool is suitable to assess security of a complex aviation system.

1.4.2. Markov Chains Models

Markov Chains is well known technique used in discrete event simulation to model behavior of networked systems and their reliability. It can be also used to reason about security properties based on degradation of services. Due to the dynamic state transition, describing the behavior using Markov models is suitable, which allows analysts to study events over time. The Markov model of a real system typically includes the state with all elements operating and a set of intermediate states representing partially failed conditions, leading to the state in which the system is unable to perform its function. The
A Markov Chain may be constructed to model the changes of the system state depending on the probability of certain security vulnerabilities and probabilities that the system can be restored back to the regular state. The transition paths and the probability of transition between the various states define the system equations whose solution represents the time-history of the system. In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters.

In the system where deviations from regular state are due to potential external attacks, we can identify the attack’s effects and their probabilities (e.g., for data communication models, percentage of messages corrupted, lost, incorrectly inserted, etc.). In turn, system transitioning to such degraded state can return to the regular state due to application of a mitigation mechanism or fail (also with assumed rate). Even after failure, we can consider system that can be repaired assuming certain repair rate. Such model is useful to quantify how a specific vulnerability, expressed in terms of probability of potential attacks, affects the system and what is the effective reliability and availability of the system. Respective simulations can give a basis for understanding how effective each mitigation method needs to be in order to ensure that the system meets or exceeds its required level of security.

For a system with no security concerns, it will not be possible to transition from a degraded state back to the normal state. By adding security, this transition will be made possible and so the resulting system reliability will be increased. The level of security added will determine the probability with which this
transition will occur. As the probability for detection and recovery increases the overall reliability of the system increases.

The essential assumption in the approach and the related model is that a security breach may cause degradation of system services and ultimately a failure. For an embedded system working in a 24/7 regime, at any given point in time the system will be in one of the following states: (1) the normal state; (2) the failure state; or (3) one of the possible intermediate degraded states, depending on the occurrence of failures and the system’s ability to detect and recover from those failures. This is true for any system whether it has security concerns or not, and is illustrated for sample case study in Fig. 5.

Figure 5. Security study - Markov diagram example

Respective mitigation techniques include authentication, encryption and other means designed to avoid and/or detect intrusions. There is no single method to evaluate the security of a new mitigation technique before it is used in a system. This is partly due to the fact that security is relative to the currently known
possible attacks. Thus, we can only say that a system is secure against these known attacks, whose nature, that is, a model is known.

1.4.3. Penetration Techniques

Experimental evaluation of security models requires the use of tools. Such tools for network monitoring and penetration are available in the public domain [81], and respective techniques are sufficiently described in the literature [86-87]. Among the most popular tools, one can find the following:

- Wireshark - http://www.wireshark.org/
- Nessus - http://www.nessus.org/
- Snort - http://www.snort.org/

Each tool has its own learning curve, sometimes significant, so certain amount of time is needed to grasp how to use it and what would be the potential benefits for assessing security. It has been reported [99] that Metasploit and Wireshark are being actively used as auxiliary tools in the Sandia Virtual Control System Environment (VCSE). Developers used Sandia's graphical environment to drive Metasploit and launch various malware codes. They also used Wireshark "to view and understand control system traffic and [...] then apply their own intrusion detection software to understand its applicability to the modeled threats."

The vulnerability scanner will have predefined sequence of instructions how to check for a given vulnerability, how to evaluate the response from the host being scanned, and how to assess the security level. This will normally follow the list of vulnerabilities defined by the National Institute of Standards and Technology [88]. There may be tens or even hundreds of different vulnerabilities that affect a single program, service, or port. For example, the NIST lists over two dozen vulnerabilities for SSH alone that
can be fixed by updating to the latest version [88]. Each of the vulnerabilities can be scanned for via an SSH terminal command using a shell script.

A response from the server can be assessed against the Common Vulnerability Scoring System (CVSS) [89] to evaluate the security score. This would give an operator and the management an insight into the level of security the system is protected at, as one of several factors in the overall picture.

To be fair, one has to include in the list techniques that are based on formal, mathematical approaches, addressing mostly program correctness. These techniques have been intensely researched but they are still far from reaching industrial maturity. A good overview of these tools, for both static and dynamic program analysis, along with pointers to theories they are based on, has been presented in [90].

1.5. Regulatory Issues

Historically, the first attempt to standardize computer security assessment efforts was the so called Orange Book, which shortly evolved into a standard [91]. Its contribution was mostly establishing the security levels for system protection, from Level D Minimal Protection, to Level C Discretionary, to Level B Mandatory and Level A Verified Protection. It provided rationale and guidelines for security testing for levels C through A.

The next chronological step in standardizing security protection was the development of so called Common Criteria [92], which evolved over a number of years. A three volume document defines a model and presents “a set of well understood security functional requirements that can be used to create trusted products reflecting the needs of the market.” For the security requirements, security assurance components are defined to help in the verification of achieving security of a target system. Suggested evaluation techniques include the following:
a) analysis and checking of processes and procedures;

b) checking that processes and procedures are being applied;

c) analysis of the correspondence between target system design representations;

d) analysis of the target system design representation against the requirements;

e) verification of proofs;

f) analysis of guidance documents;

g) analysis of functional tests developed and the results provided;

h) independent functional testing;

i) analysis for vulnerabilities (including flaw hypothesis);

j) penetration testing.

Common Criteria have been ultimately standardized by the ISO [93].

Two other standards are worth mentioning in this context. Even though they are of lesser importance for aviation, selected concepts may be still applicable. The FIPS Standard for Security Categorization [94] deals with the basic terms related to security, and its value is mostly in defining them and the qualifying the impact security violations may have on the target system. The PCI standard [95] covers security requirements and assessment procedures in the banking industry, regarding “any network component, server, or application that is included in or connected to the cardholder data environment.” The requirements are set on: (1) building protecting a secure network; (2) protecting cardholder data; (3) maintaining a vulnerability management program; (4) implementing strong access control measures; (5) regularly monitoring and testing networks; and (6) maintaining an information security policy;
2. Software Assurance

The Committee on National Security Systems defines software assurance as follows [53]:

*Software assurance (SwA) is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner.*

Technical report [100] expands this definition by emphasizing the importance of not only technologies but also processes to achieve software assurance. The report expands the picture by adding that the system provides security capabilities appropriate to the threat environment, and is able to recover from intrusions and failures. Such extended definition focuses on both software systems and services.

2.1. General Concepts

Software capabilities can be initiated from new system development, legacy system evolution, or acquisition. Software capabilities can also originate from service acquisition through cloud computing, virtualization, or service-oriented architecture.

Systems often aggregate combinations of these sources, all of which require a level of assurance with respect to correct functionality, safety, and security. In some cases, such as service acquisition, the software itself may not be available for analysis, and assurance must be achieved through other means.

For assured system, the software and services:
• Must function in the intended manner exhibiting levels of quality and correct functionality commensurate with the consequences of their failure thus rigorous software engineering capabilities and best practices in technologies and processes must be applied.

• Must be free from accidental or intentional vulnerabilities so both legitimate and malicious users are prevented from disrupting operations or obtain access to information, therefore a thorough analysis and rigorous security requirements engineering, architecture and design, coding, and testing techniques are required.

• Must provide security capabilities appropriate to the threat environment whose virulence can vary with the value of the functions and information the systems provide, therefore systems must implement security capabilities, such as authentication, authorization, non-repudiation, and privacy, and support the properties of availability, confidentiality, and integrity.

• Must be able to recover from intrusions and failures and operational continuity and survival must be assured even in adverse circumstances.

These objectives are to be achieved by:

(a) Application of appropriate technologies and processes

Assurance technologies include analytical verification of software functionality, software vulnerabilities analysis, assessment of threat environments and security capabilities. Assurance processes to determine required levels of confidence can be integrated into traditional software development and acquisition process including process-oriented assurance activities with defined organizational goals, objectives, and constraints, risk analysis and reduction.

(b) Achieving a required level of confidence that assurance goals are met

A key responsibility of software assurance is to create auditable evidence that supports achievement of assurance goals. Assurance requirements can vary with business objectives, threat environments, system capabilities, risk analysis, legal and compliance requirements, and internal and external standards.
2.2. Organizations

The CERT Division of the Software Engineering Institute (SEI) has evolved dramatically since it was created in 1988 as the CERT Coordination Center in response to the Morris worm incident [101]. The small organization established to coordinate response to Internet security incidents now has more than 150 cybersecurity professionals working on projects that take a proactive approach to securing systems. The CERT Division works closely with the Department of Homeland Security (DHS) to meet mutually set goals in areas such as data collection and mining, statistics and trend analysis, computer and network security, incident management, insider threat, software assurance, and more.

CERT provides direct support to the Department of Defense (DoD) working with the Defense Information Systems Agency in an effort to increase global situational awareness and provide core analytical systems in the area of malicious code analysis that are used across the DoD. CERT cooperates with Navy's Space and Naval Warfare Systems Center and the MITRE Corporation developing a proof-of-concept vulnerability remediation capability, and provides core analytical support to the Defense Industrial Base Collaborative Information Sharing Environment (DCISE), the focal point and clearinghouse for referrals of intrusion events on defense organizations' unclassified corporate networks and maintaining threat information base for industry partners who share relevant information to more effectively protect critical data.

2.3. Software Assurance in Aviation

Training conducted in the Systems Certification and Integrity Division of the Australian Directorate General Technical Airworthiness (DGTA-ADF) [102] captured several relevant points regarding software assurance in aviation. The presenters identified the major principles of aviation software assurance:
a) Requirements Satisfaction: have the required behaviors been translated into the product?

b) Requirements Validity: does the software have the right behaviors?

c) Requirements Traceability: are the all behaviors of the software accounted for?

d) Non-Interference: do other functions interfere with critical functions?

e) Configuration Consistency: does the verification evidence have traceability to the product?

f) Design Integrity: were good design practices observed?

To meet requirements of higher software assurance levels there is evident need to increase the degrees of rigor in verification of requirements and software architecture, test coverage and robustness testing, configuration control, independence of software verification process, compliance with standards, and traceability.

In conclusion, to provide assurance that the system may be considered safe and secure, the following error and accident causes need to be eliminated:

- Discrepancies between documented requirements specifications and the requirements needed for the correct functioning of the system.
- Misunderstandings about the software’s interface with the rest of the system.
- Requirements specifying behavior that is not safe or secure from the system perspective.
- Requirements not specifying some particular safety and security behaviors.
- The software having unintended (and unsafe/unsecure) behavior beyond what is specified in the requirements.

Despite ever increasing complexity of software-intensive aviation systems, due to the strict observance of principles and rules outlined in this chapter, the commercial aviation has one of the best records regarding system safety, and aim to achieve the same regarding system security.