



Tokeneer: An Open Source Demonstration of High-Assurance Software Engineering



Janet Barnes, *Praxis High Integrity Systems*



Demonstration Project Goals

- Demonstrate that Common Criteria requirements for EAL5 are achievable in a cost effective manner
- Show how the Praxis “Correctness by Construction” approach matches up to EAL5
- Measure productivity and defect rates under controlled conditions

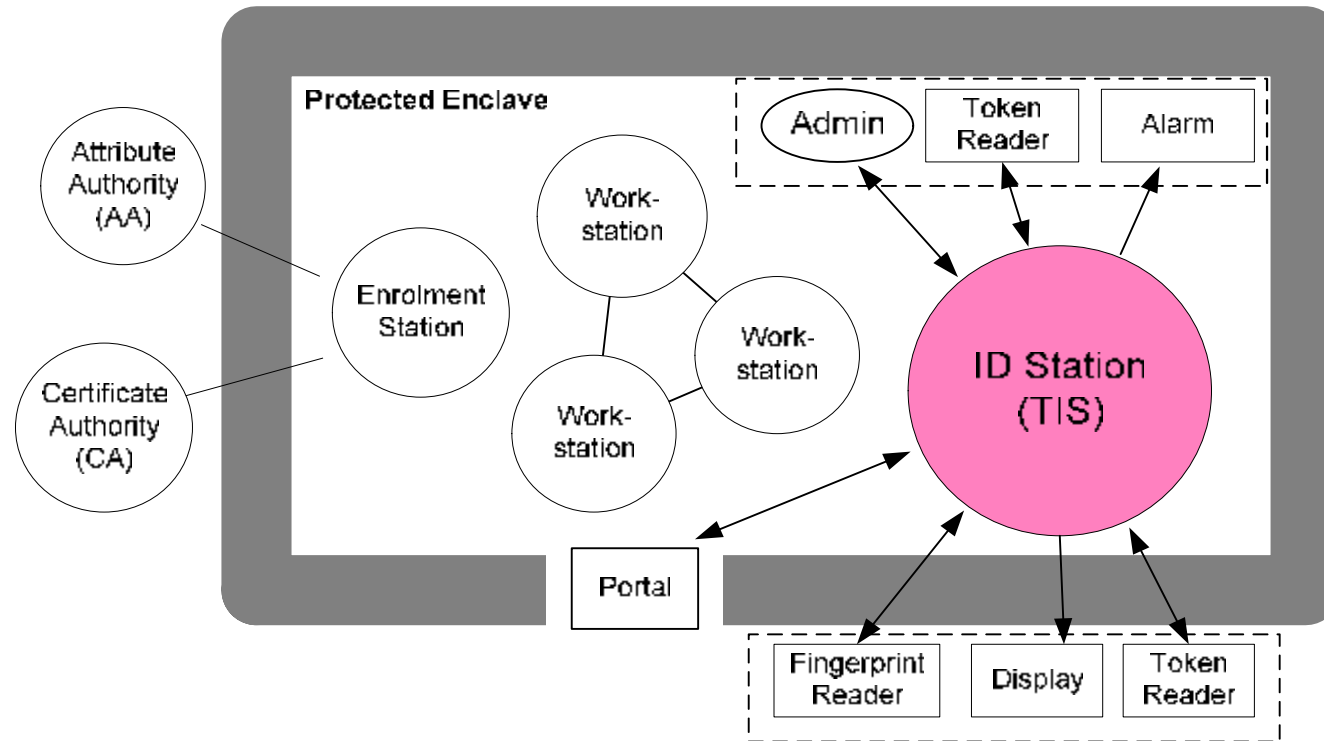


Common Criteria

- Assurance Requirements for Evaluation of Security Software
- 7 Evaluation Assurance Levels (EAL1 – EAL7)
- Covers key aspects of software
 - Configuration Management
 - Development
 - Guidance Documents
 - Lifecycle Support
 - Tests
 - Vulnerability Assessment
- EAL5 Represents a semi-formal development process



What is TOKENEER ?



- Provides protection to secure information held on a network of workstations situated in a physically secure enclave
- Demonstrates use of **Smart Cards** and **Biometrics**



Tokeneer ID Station

- Praxis re-developed the Core functions of the Tokeneer ID Station (TIS)
 - Controls User access to the enclave
 - Checks Token and Biometrics, and unlocks door
 - Audits all activity
 - Monitors door state and raises alarms
 - Administrative functions
 - Guard - Override Door
 - Security Officer – Shutdown or Change Configuration
 - Audit Mgr – Archive Log
- All peripherals simulated

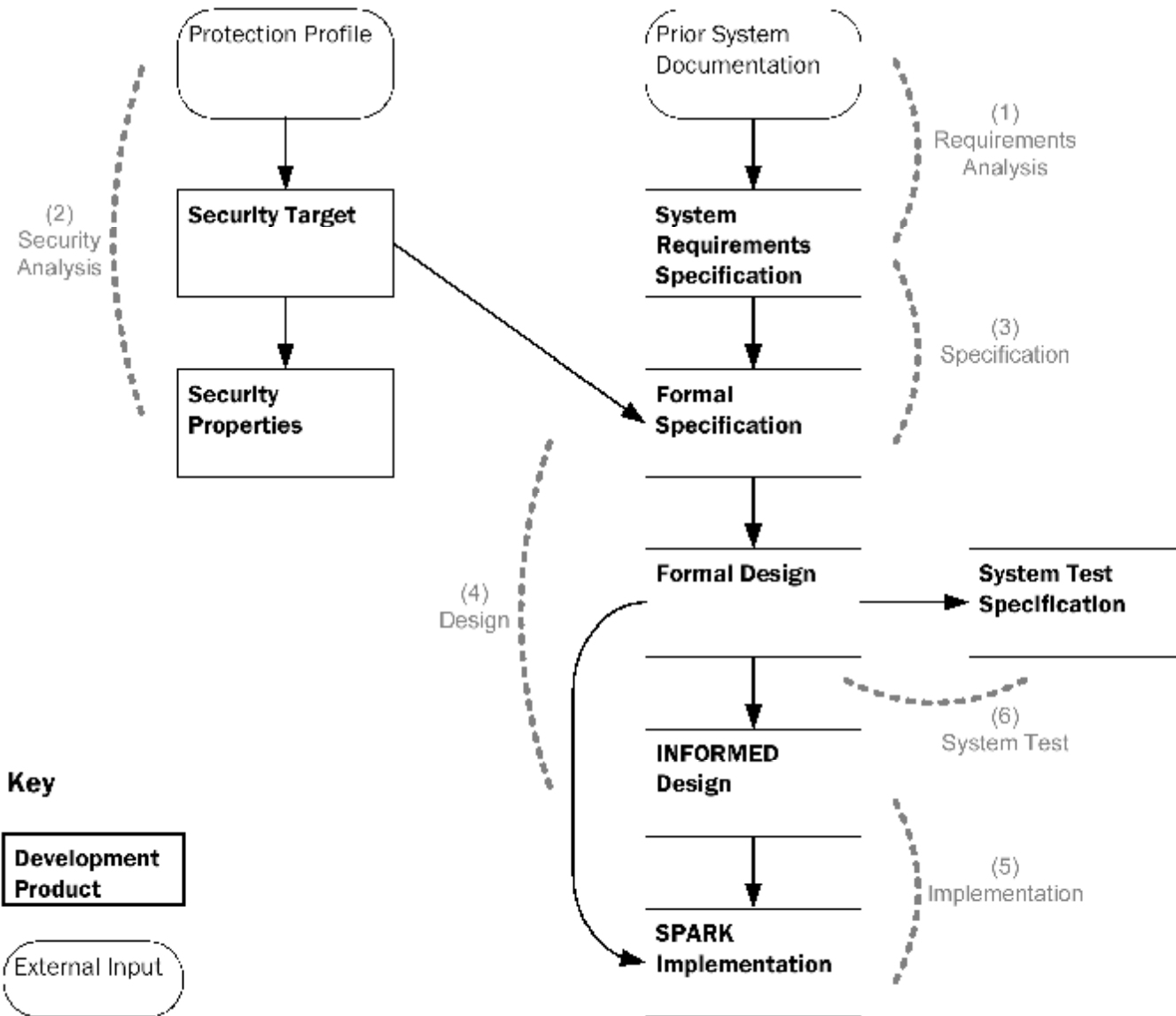


The principles of CbyC

- Avoid introducing errors
 - use sound, formal notations
 - say things only once
 - keep it simple
- Remove errors as soon as possible
 - use strong (tool-supported) validation
 - carry out small steps
 - do the hard things first (manage risk)
- Generate evidence as you go



Development Process





Why Formalise Specifications?

- Key Benefits
 - Unambiguous and demonstrably complete
 - Amenable to formal verification
 - Forces designers fully understand requirements
- We used the **Z Notation** a mathematical specification language for
 - Formal Specification and Formal Design
 - Security Properties
- By choosing Z notation
 - Tool supported Type-checking
 - Using common notation allows proof of relationships between documents



Implementation using SPARK

- SPARK is an contractualized subset of Ada
 - Contracts enrich code specifications
 - Global state
 - Information flow relationships
 - Operation pre- and post- conditions
- Statically analysable using the **SPARK Examiner**
 - Data and information flow analysis
- Code can be proved
 - Proof of freedom from many classes of run time errors
 - Partial correctness proof

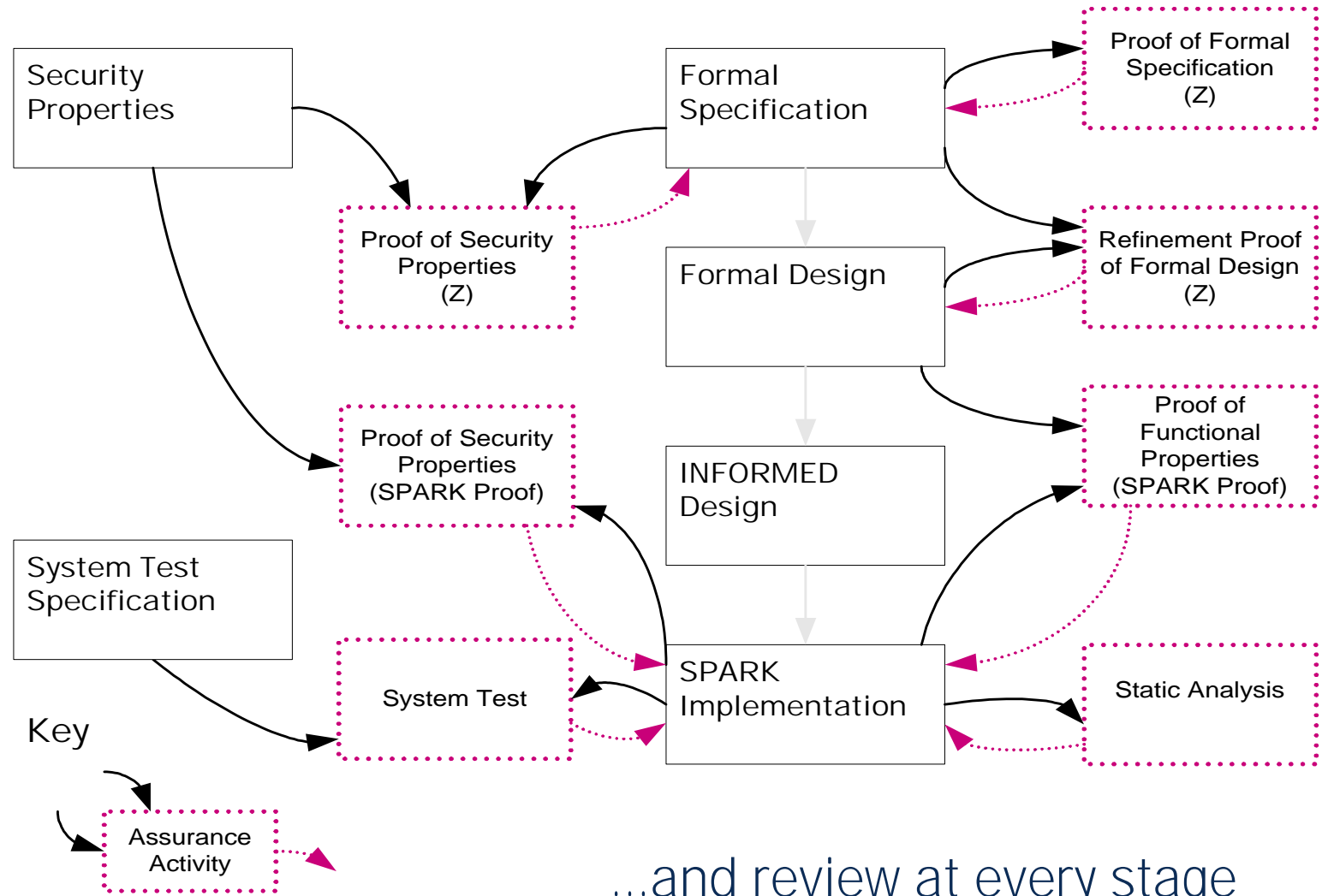


Why a SPARK Implementation?

- Key Benefits as Implementation Language
 - Unambiguous formally defined language
 - Code can be **statically analysed** early
 - Flow properties of code visible and checkable
 - Static demonstration of absence of many errors
- Key Benefits as Proof Language
 - Proof contracts use an enhanced dialect of Ada so easily readable to Ada programmer
 - Properties specified in SPARK proof contracts can be tailored to required level of detail
- Tool supported
 - Examiner, Simplifier, Checker



Assurance process





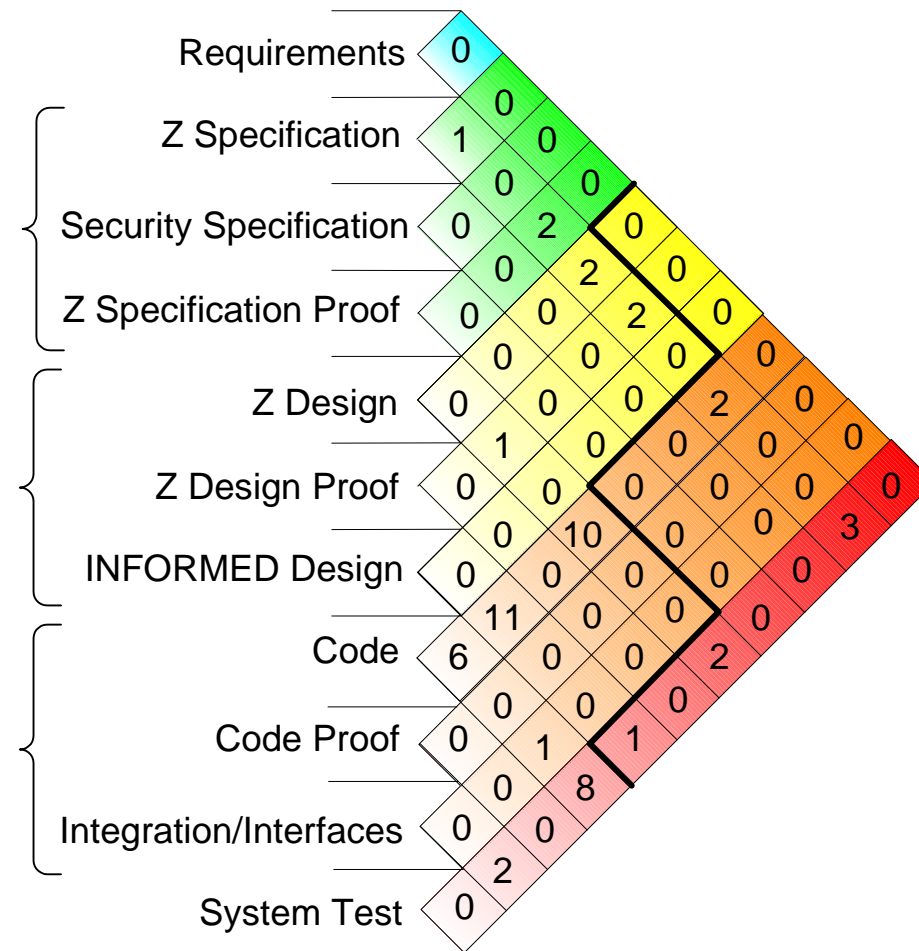
Development Statistics

	Ada Source Lines	SPARK	LOC/day coding	LOC/day overall
Core	9,939	16,564	203	38
Support	3,697	2,240	182	88

- Total effort - 260 man days
- Team - 3 people part-time
- Total schedule - 9 months elapsed



Metrics – Development Defects



- 54 defects
- 8 found late



Metrics – Post Delivery Defects

- Software Defects found in independent test : zero
- By NSA prior to public release (2004 – July 2008) : zero
- In preparation for public release : one
- Since public release : one

- Defect rate 0.2 per KLOC



Tokeneer public release

- NSA “Technology Transfer Agreement” licensed the Tokeneer project for public release
 - This is essentially an open-source license for the entire project archive
- To obtain the release
 - www.adacore.com/tokeneer
- Related News : GPL release of SPARK Tools in June '09.



Tokeneer public release

- Why release the project?
 - An original aim of the project was to demonstrate that EAL5 software could be written cost effectively
 - As a model of high assurance software engineering
 - Rare opportunity to demonstrate Praxis' High Integrity CbyC process in practice
 - As a vehicle for teaching and research in high assurance software and security



Conclusions

- CbyC process
 - Produces high quality, low defect software
 - Is cost effective
 - Conforms to Common Criteria EAL5
- It requires
 - Clear goals
 - Precise notations
 - Discipline of engineering
- Wider community can now gain a greater understanding of what is needed to develop high assurance software



Janet Barnes

Praxis High Integrity Systems
20 Manvers Street
Bath BA1 1PX

janet.barnes@praxis-his.com

Tokeneer

www.adacore.com/tokeneer