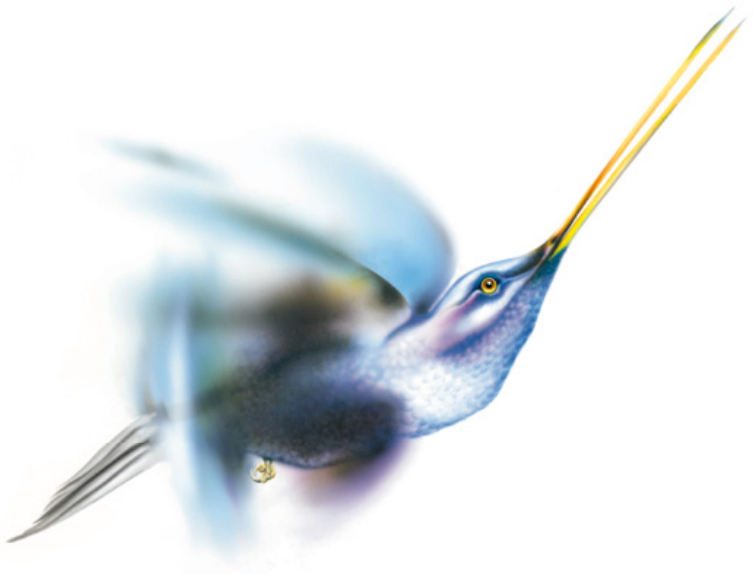
Correctness by Construction: Developing a Commercial Secure System



Roderick Chapman
Praxis Critical Systems

Praxis Critical Systems

Outline

- Background - The MULTOS CA
- Development Approach
- Formal Methods
- Results
- Conclusions
- Resources

Background - The MULTOS CA

- **Certification Authority for MULTOS scheme**
  - enable cards
  - sign application load certificates
- **Distributed multiprocessor system**
  - security
  - throughput
- **“Certifiable to ITSEC E6”**
  - not to be certified within project timescale
- **COTS mandatory**
  - infeasible to build from scratch

Development Approach

- **Overall process conformed to E6**
- **Conformed in detail where retro-fitting impossible**
  - **Development environment security**
  - **Language and specification standards**
  - **Configuration management and audit information**
- **Our deliverables could individually be certified to E6**
- **Reliance on COTS for E6 claims minimised/eliminated**
  - **Assumed arbitrary but non-byzantine behaviour**
  - **Assumed machines fail-silent on crash, for instance**

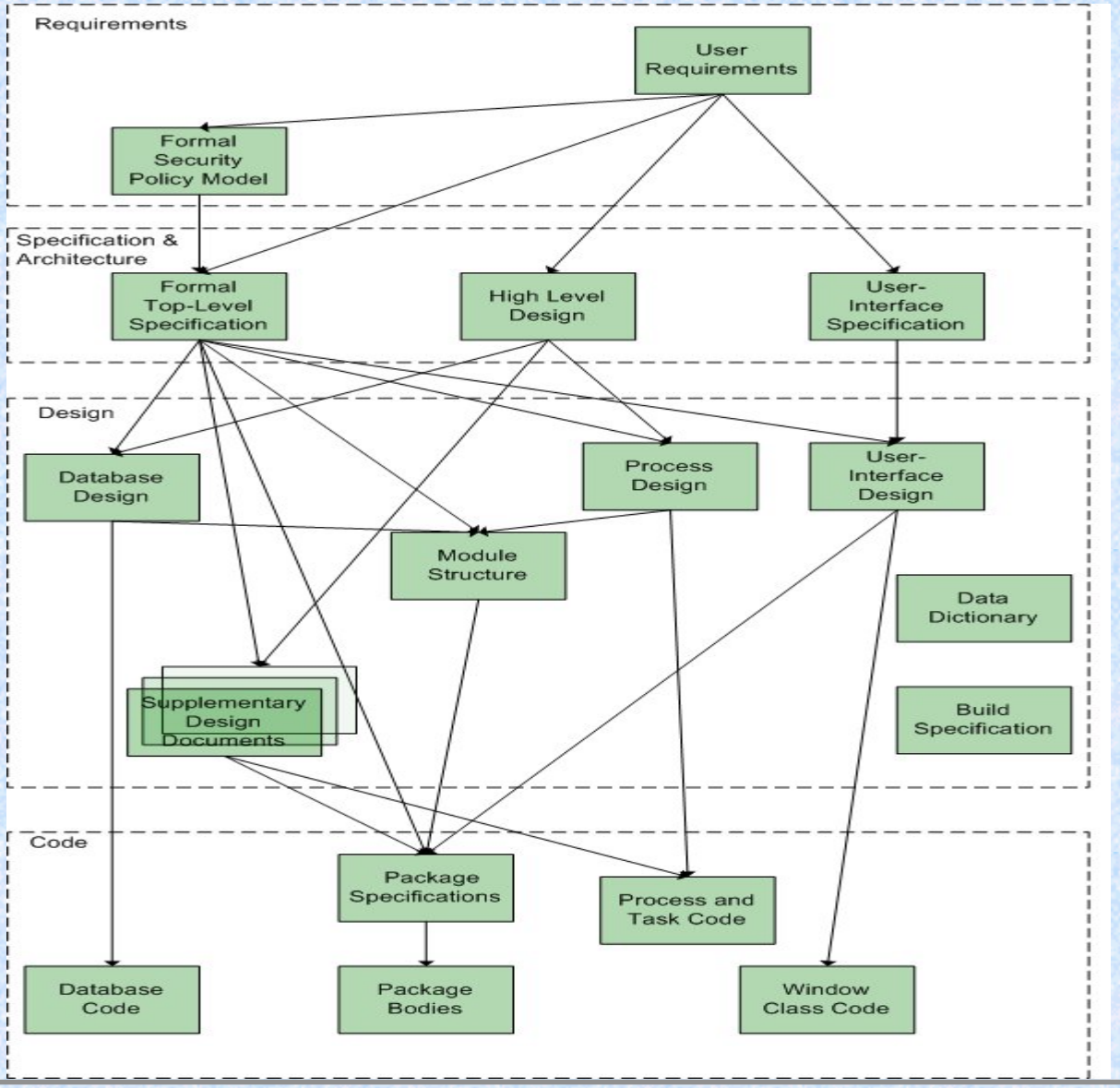
Development Approach - Limitations

- **COTS not certified**
- **Praxis not responsible for all items necessary for certification**
  - **operational documentation**
  - **operational environment**
- **No formal proof**

Development Lifecycle

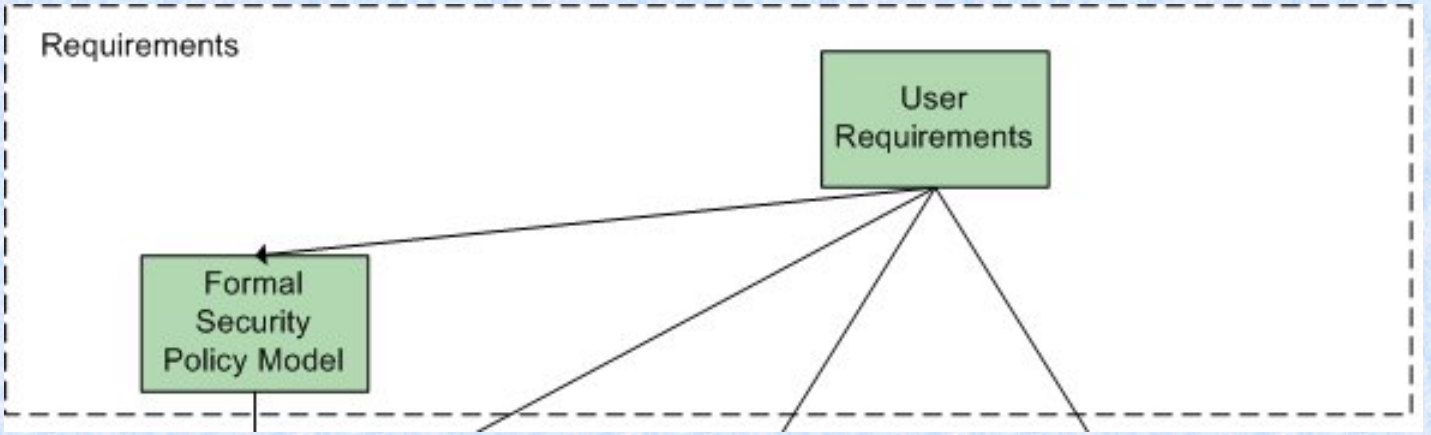
- User requirements definition with REVEAL®
- User interface prototype
- Formalisation of security policy and top level specification
- System architecture definition
- Detailed design including formal process structure
- Implementation in SPARK and VC++
- Top-down testing with coverage measurement

Lifecycle Deliverables



Praxis Critical Systems

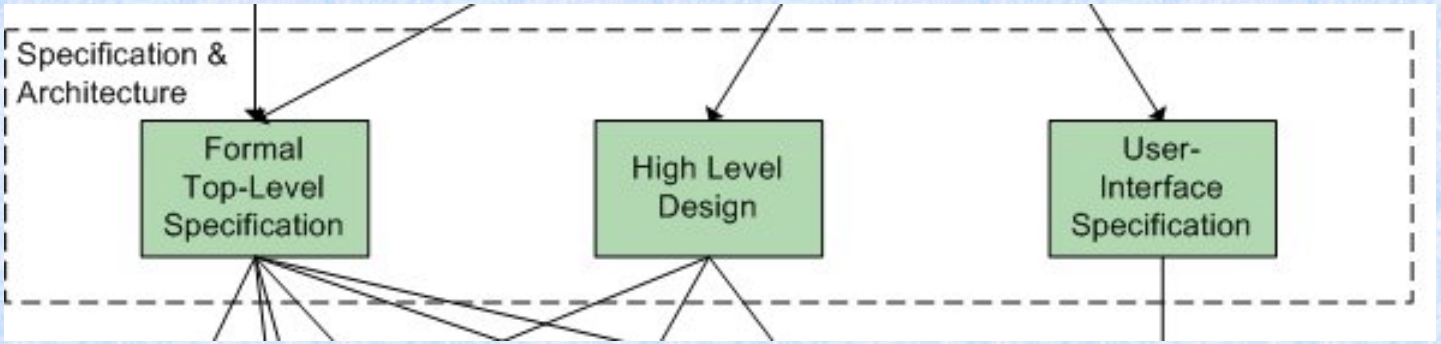
Phase 1 - Requirements



- **User requirements included informal security policy**
  - assets; threats; countermeasures
- **Appropriate items formalised**
  - only technical items
  - related to lifecycle stage
- **Z used to express formal model**
  - simplified GCHQ CESG Manual F
- **Tracing incorporated from the start**
  - Threats ⇐ Policy ⇐ FSPM

Praxis Critical Systems

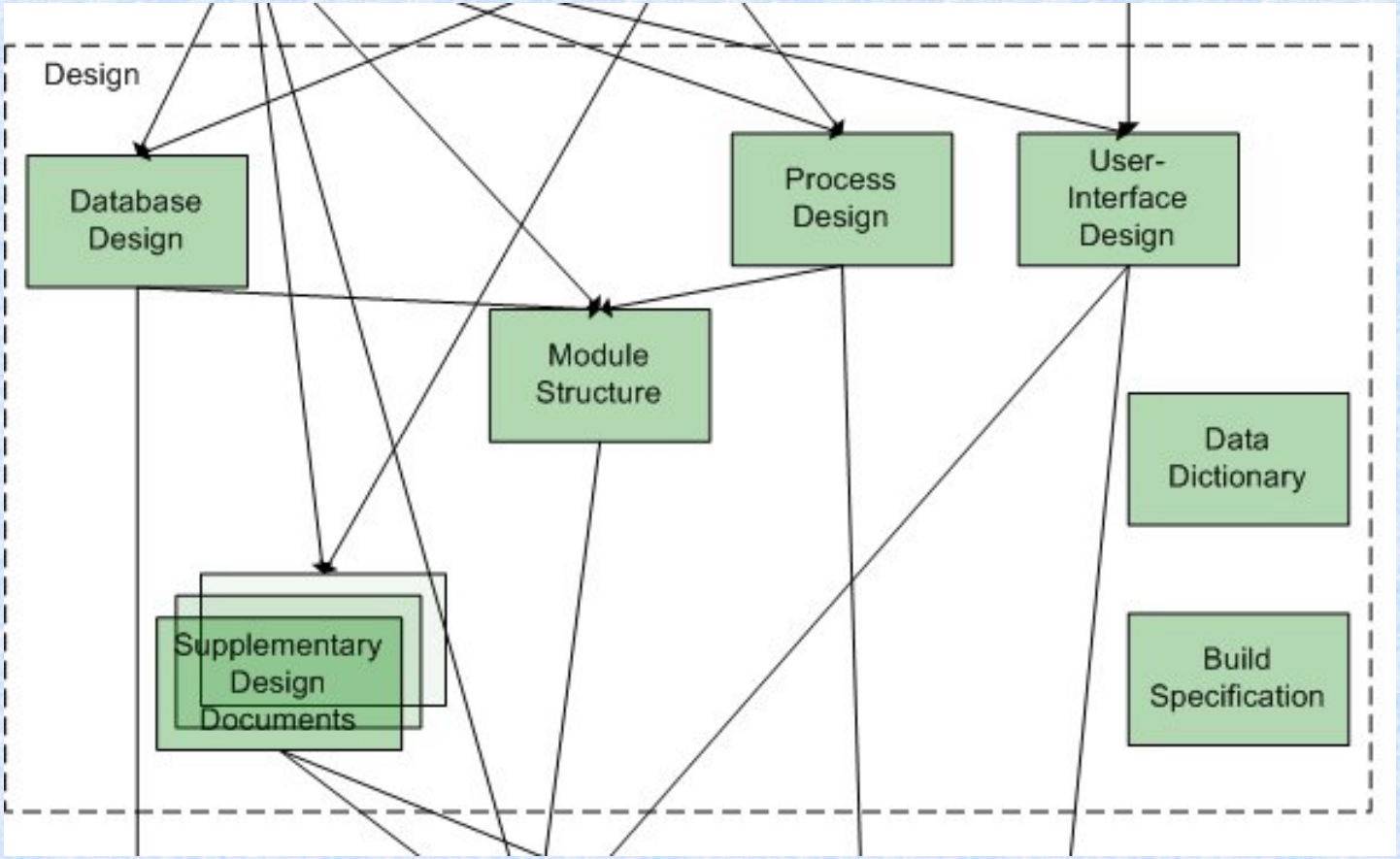
Phase 2 - Specification and Architecture



- **Distinguish "top level description" from "top level design"**
- **FTLS**
  - **fully formal top level description in Z**
  - **traced to FSPM (as well as URS)**
  - **no formal demonstration of correspondence**
- **HLD**
  - **Many aspects**
  - **NOT all formalised**
  - **CSP for process structure**
- **UIS**
  - **Look and feel**

Praxis Critical Systems

Phase 3 - Detailed Design



Praxis Critical Systems

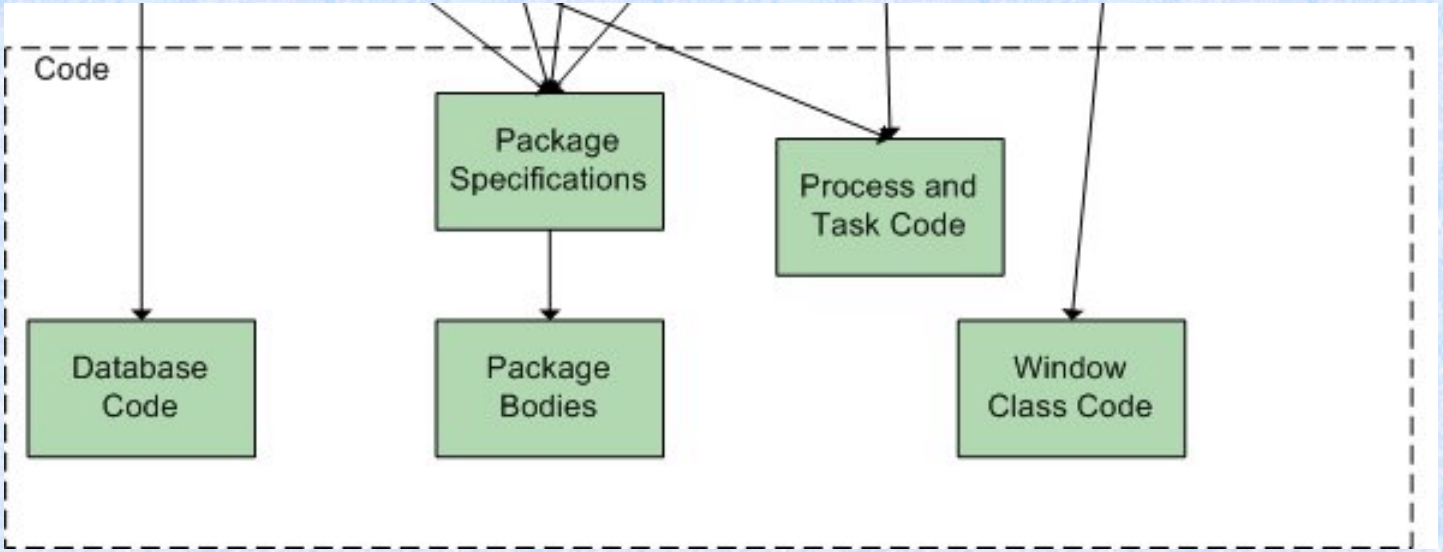
Phase 3 - Detailed Design

- **Database Design**
  - ERA modelling of persistent system state
  - Logical, physical DB design
  - Protection mechanisms (e.g. MACs, encryption)
  - Transactions, recovery, sizing etc.
- **Process Design**
  - CSP model
- **User Interface Design**
  - Windows, Dialogs, Messages, State machines
  - Interface between the GUI and the application software

Phase 3 - Detailed Design

- **Module Structure**
  - Software architecture
  - Information-flow centric view. Careful separation of security enforcing from non-secure functions
- **Supplementary Designs**
  - Refinements (some formal) of key components - e.g. crypto key storage manager
- **Build Specification**
  - Very detailed “how to build the MGKC” document
- **Data Dictionary**

Phase 4 - Code



- **Which Languages to use?**
- **Which development technologies?**
- **Principles:**
  - **Use what we know from safety-critical systems**
  - **Aim for 6 months between re-boots - on Windows NT**
  - **Prefer sound technology over “fast-moving” or “fashionable” technologies**

Praxis Critical Systems

Coding the CA

- **No one language or technology could do the job.**
- **Mixed language development - the right tools for the job!**

– SPARK	30%	"Security kernel" of tamper-proof software
– Ada95	30%	Infrastructure (concurrency, inter-task and inter-process communications, database interfaces etc.), bindings to ODBC and Win32
– C++	30%	GUI (Microsoft Foundation Classes)
– C	5%	Device drivers, cryptographic algorithms
– SQL	5%	Database stored procedures

Phase 5 - Verification and Validation

- **Review everything, involving customers where possible.**
  - Automate as much as possible, so manual reviews are focussed on what's important
- **Testing**
  - Top-down incremental builds
  - Real GUI from tested at build N forms the test harness for application software in build N+1
  - Systematic derivation of tests from requirements, UIS and FTLS
  - Collected statement and branch coverage
  - Additional test scenarios to fill coverage gaps

Formal Methods

- **FSPM, FTLS and some supplementary designs are expressed in Z.**
- **Process design is in CSP, and model-checked using the FDR tool.**
- **SPARK can be seen as a formal programming language**

Formal Methods - Successes

- **Formalisation leads to early discovery of ambiguity and inconsistency.**
- **FTLS was a contractual baseline in the project.**
  - **No debate over a *fault* (we pay for it!) or a *change* (they pay for it!)**
  - **A strong commercial success.**
- **Model checking of CSP found significant design errors which were fixed prior to coding.**
- **Concurrent and distributed code ran first time.**
  - **Simple translation from CSP to Win32 Named Pipes and Ada tasks.**

Formal Methods - Limitations

- **Not all design elements have appropriate formal notations**
  - **What is a “formal architecture” anyway?**
- **Tool support still needs work in some areas**
  - **Model checking stressed available computing resources**
  - **Tool support for Z remains rudimentary**
- **Customers perceive FM as difficult**

SPARK

- **SPARK is a programming language, design approach, and static analysis technology designed for high-integrity systems.**
- **The language is an annotated subset of Ada95.**
- **A strong track record in the safety-critical industry, although, ironically, its roots are in the security community.**

SPARK - Design Goals

- **Logical Soundness**
  - No ambiguities
- **Simplicity of formal description**
  - A formal descriptions of SPARK's static and dynamic semantics were constructed some years ago.
- **Expressive power**
- **Security**
  - All language rule violations are detectable statically.
- **Verifiability**
  - Formal proof of correctness is achievable. Tool support exists and is used.

SPARK and Static Analysis

- **Rule of thumb - if you want someone to use a static analysis tool, it must be as fast as (or faster) than the compiler!**
- **SPARK is entirely *unambiguous*.**
  - **So analysis is *both* efficient and deep.**
  - **E.g. complete information-flow analysis of SPARK is decidable in polynomial time/space.**
- **SPARK facilitates *constructive* static analysis.**

SPARK and Secure Systems

- **SPARK has some unique properties that make is appropriate for the development of secure systems:**
  - **Complete program-wide data- and information-flow analysis**
  - **Verification-condition generation and theorem proving allow proof of**
    - **Partial correctness**
    - **Invariant properties**
    - **Freedom from runtime exceptions (e.g. no buffer overflows!)**

SPARK and Secure Systems (2)

- **SPARK be compiled with *no* supporting run-time library - useful if evaluation of such COTS components is a problem.**
  - **GCC compiles SPARK in this fashion.**
- **SPARK is (as far as we know) the *only* general-purpose programming language that meets the requirements of Common Criteria.**
  - **…and ITSEC, and Def. Stan. 00-55, and CENELEC 50128…**

What's wrong with SPARK?

- **It's Ada: despite technical strength, Ada remains misunderstood.**
  - **GCC might change this…**
- **It's not "hot" or "fashionable".**
- **It's (relatively) unknown outside Europe. Why?**
- **You're not using it!**

The CA Development - Results

- **1 year after delivery, 4 defects were found in 100,000 lines of code - 0.04 defects per kloc.**
  - **These were, of course, corrected under our warranty.**
- **Productivity was 28 lines of code per day, taking all project phases into account.**
  - **This compares favourably with other high-integrity projects.**
- **Total effort: 3571 person days.**

Results - Distribution of Effort

Activity	*Effort (%)*
User Requirements	2
Specification and architecture	25
Design and code	14
Test	34
Fault fixing	6
Project management	10
Training	3
Design authority	3
Development- and Target-environment	3

Conclusions

- **Three main themes:**
- **Successful use of COTS through careful architectural design and separation.**
- **Practical, large-scale use of formal methods. Both a technical and commercial success.**
- **Use of SPARK (mixed with other languages) to build a highly available, robust system.**

Resources

- **The paper: IEEE Software, Jan/Feb 2002.**
  - **We have reprints here.**
- **SPARK: www.sparkada.com**
- **MULTOS: www.multos.com**
- **Us:**
  - **rod.chapman@praxis-cs.co.uk**
  - **anthony.hall@praxis-cs.co.uk**

Questions?