



## **SPARK** and how it addresses security issues

Rod Chapman  
Praxis High Integrity Systems



### **Praxis HIS**

- specialists in the engineering of high-integrity and safety-critical, software-intensive systems
- delivers services by:
  - provision of tools such as the SPARK Examiner
  - outsourcing complete projects
  - capability enhancement
  - providing key consultancy
- 100+ technical staff and growing
- autonomous but part of the Altran engineering consultancy group
- founded specifically to put **engineering** into **software engineering**
  - first software house in the world to achieve ISO 9001 certification

Copyright © Praxis HIS - 2006

Slide 1



## Rod Chapman

- SPARK Team Leader at Praxis HIS
- Joined Praxis in 1995, following MEng and DPhil at York.
- Engineering role on several key-note projects, such as LM C130J, SHOLIS, and MULTOS CA.
- Now mainly doing technical R&D, training, customer support, sales and marketing of SPARK and Correctness-by-Construction
  - (Still a techie though...honest...)

Copyright © Praxis HIS - 2006

Slide 2



## Agenda

- What is SPARK?
- What can we do with it?
- Brief project examples

Copyright © Praxis HIS - 2006

Slide 3



## Agenda

- What is SPARK?
- What can we do with it?
- Brief project examples

Copyright © Praxis HIS - 2006

Slide 4



## SPARK

- An annotated language (based on Ada):
  - completely **unambiguous**
  - **free** from undefined and implementation-dependent language
  - **all** rule violations are detectable
  - formally defined
- SPARK language definition is open and widely available
- SPARK can be used as a
  - **programming** language or
  - precise, executable **design language** from which other languages can be generated
- Supported by analysis tools (including proof)

Copyright © Praxis HIS - 2006

Slide 5



## SPARK supports static verification

- The language is designed to provide *constructive* static analyses that are:
  - Deep  
(tells you something useful...)
  - Sound  
(with no false-negatives...)
  - Fast  
(tells you it *now*...)
  - Complete  
(with as few false-positives as possible...)
  - Modular  
(and works on incomplete programs.)

Copyright © Praxis HIS - 2006

Slide 6



## SPARK supports static verification

- Therefore analysis can **ensure**:
  - freedom from language misuse
  - freedom from data and information flow errors
  - freedom from run-time errors
  - specified safety/security properties are guaranteed
- That's why the analysis tool is called the SPARK **Examiner**

Copyright © Praxis HIS - 2006

Slide 7



## Why Annotations?

- Annotations strengthen specifications
  - providing **design-by-contract** facilities
- Allows analysis without access to procedure-bodies
  - which can be done **early** during development
  - **before** programs are complete or compilable
- Erroneous constructs are efficiently detected

Copyright © Praxis HIS - 2006

Slide 8



## What SPARK is **NOT**

- SPARK is *not*...
  - “just a subset” of Ada...
  - A “code scanning” or “bug finding” style static analysis tool...
  - Suitable for retrospective use on existing code...

Copyright © Praxis HIS - 2006

Slide 9



## Why (lack of) ambiguity is crucial

- The Standard definitions of all common unsubsetted programming languages are *ambiguous*.
  - E.g. unspecified and undefined behaviours in C
  - E.g. implementation-dependent and implementation-defined behaviours in Ada.
- The standards are important, because *that's what the compilers implement*.
- Ambiguity is terrible curse from the point of view of a verification tool, since it impacts soundness and completeness.
- Here is a small example:

Copyright © Praxis HIS - 2006

Slide 10



## #include “nasty test case”

```
#include "stdio.h"
static int d;

int f(int x)
{
    d = 5;
    return (x + 1);
}

int main (int argc, char **argv)
{
    int y;
    int a[4] = {1, 2, 3, 4};

    d = 2;
    y = a[d] + f (5); /* Evaluation order dependency! */
    printf ("Value of y is %d\n", y);
    return 0;
}
```

Copyright © Praxis HIS - 2006

Slide 11



## #include “nasty test case”

- What does this program mean?
- If left-to-right evaluation order, then
 

```
Value of y is 9
```
- If right-to-left, then there’s a buffer overflow, so behaviour is undefined.
  - GNAT Pro 3.16a (gcc 2.8.1):
 

```
Value of y is 4198647
```
  - Microsoft Visual C 6.0:
 

```
Value of y is 4198748
```
- Even knowing which compiler you are using doesn’t help!
- What should a static analysis tool do?

Copyright © Praxis HIS - 2006

Slide 12



## An example (detection of erroneous constructs)

```
procedure Inc (X : in out Integer);
--# global in out Callcount;
```

detection of function side-effect

```
function AddOne (X : Integer)
  return Integer is
  XLocal : Integer := X;
begin
  Inc (Xlocal);
  return XLocal;
end AddOne;
```

detection of aliasing

```
Inc (CallCount);
```

Copyright © Praxis HIS - 2006

Slide 13



## Agenda

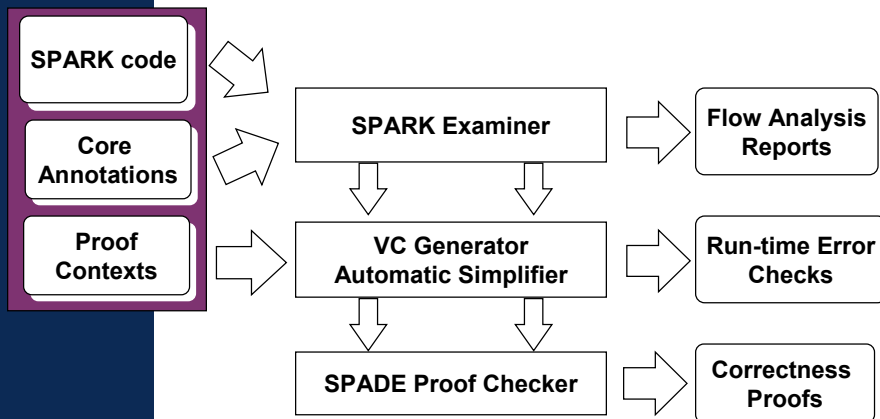
- What is SPARK?
- What can we do with it?
- Brief project examples

Copyright © Praxis HIS - 2006

Slide 14

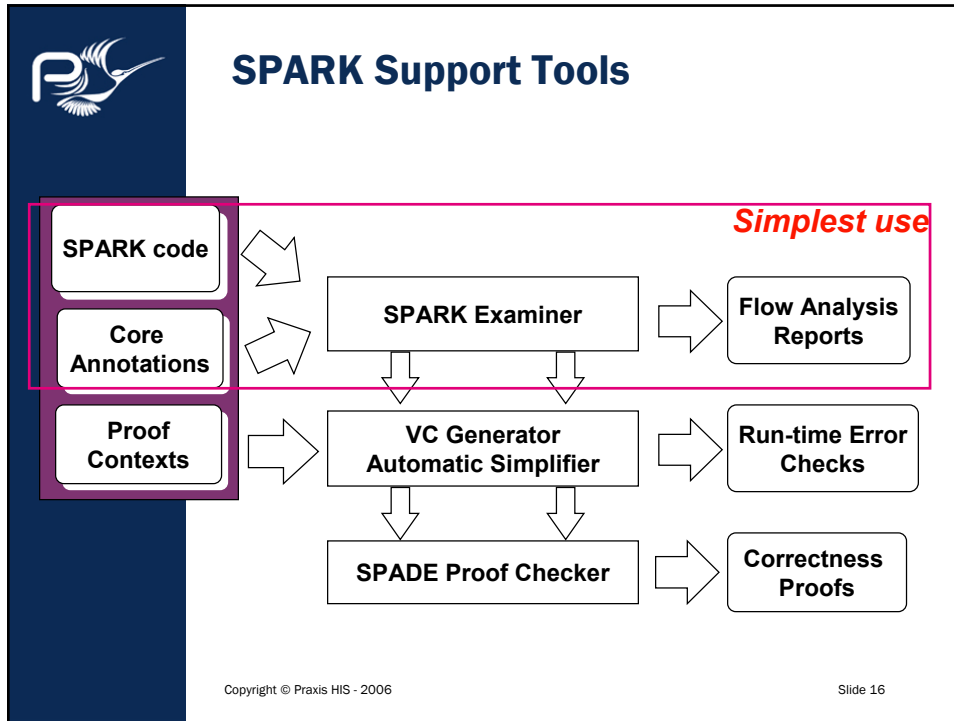


## SPARK Support Tools



Copyright © Praxis HIS - 2006

Slide 15



### Example: Data Flow Analysis

Example: an Aircraft Warning System

A specification for this function might be:

mon_Event	con_Bell
Warning	True
Caution	True
Advisory	False

An (incorrect) implementation might be:

```

type Alert is (Warning, Caution, Advisory);

function RingBell(Event : Alert) return Boolean
is
  Result : Boolean;
begin
  if Event = Warning then
    Result := True;
  elsif Event = Advisory then
    Result := False;
  end if;
  return Result;
end RingBell;
  
```

Copyright © Praxis HIS - 2006 Slide 17



## Example Contd. – Test Results

- There is a high probability that this code would pass through testing
- In the case of **Event=Warning** the code correctly returns **True**
- In the case of **Event=Advisory** the code correctly returns **False**
- In the case of **Caution** it returns an **undefined** value picked up from memory; however, there is a very high probability that this random value will be **non zero** and will be interpreted as **True** which is the expected test result

Copyright © Praxis HIS - 2006

Slide 18



## Example Contd. – Examiner Output

```

13  function RingBell(Event : Alert) return Boolean
14  is
15      Result : Boolean;
16  begin
17      if Event = Warning then
18          Result := True;
19      elsif Event = Advisory then
20          Result := False;
21      end if;
22      return Result;
    ^1
??? ( 1) Warning : Expression contains reference(s)
        to variable Result, which may be undefined.

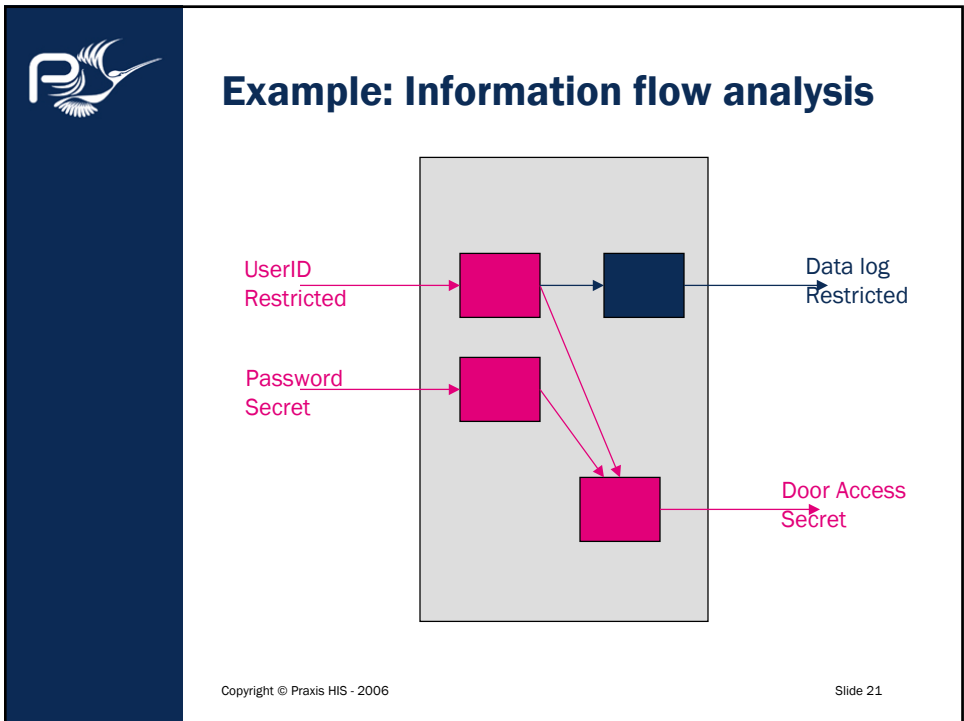
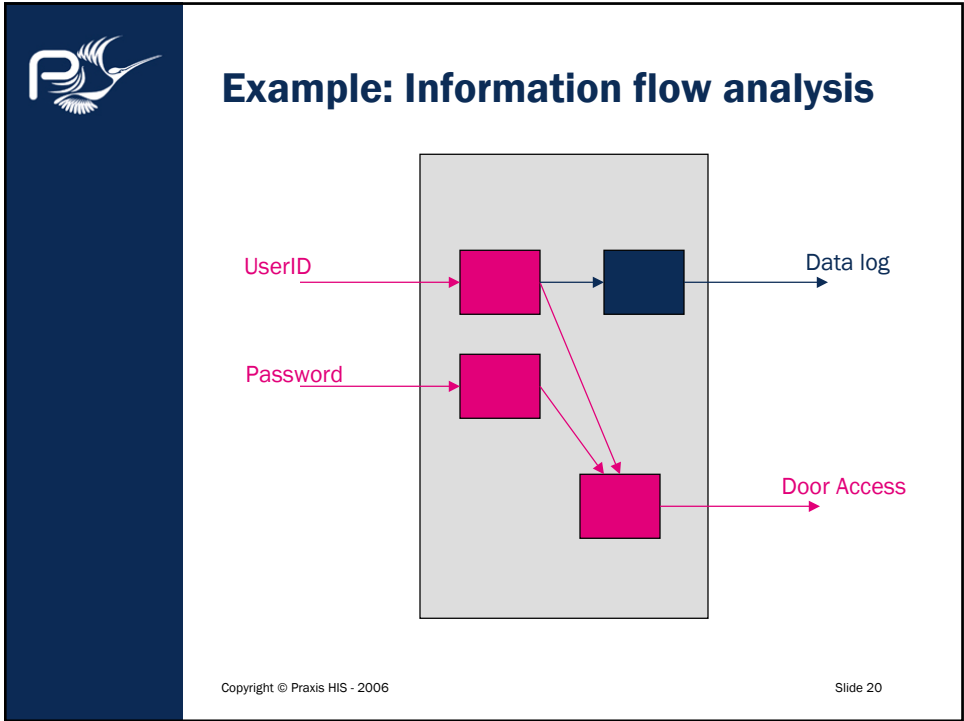
23  end RingBell;

??? ( 2) Warning : The undefined initial value of
        Result may be used in the derivation of the
        function value.

```

Copyright © Praxis HIS - 2006

Slide 19



**Information flow violation**

The diagram illustrates an information flow violation within a system boundary. On the left, two inputs are shown: "UserID Restricted" and "Password Secret". These inputs flow into a system represented by a grey box containing three pink rectangular components. Arrows indicate the flow of information: from the top-left component to the top-right component, from the bottom-left component to the top-right component, and from both the top-right and bottom-left components to the bottom-right component. On the right side of the system, two outputs are shown: "Access log Restricted" and "Door Access Secret". A pink callout box with the text "!!!Restricted output derived from Secret input" points to the "Access log Restricted" output, indicating that restricted information is being leaked through a non-restricted output channel.

Copyright © Praxis HIS - 2006 Slide 22

**Example: run-time error proof**

```

type T is range 0 .. 100;

procedure Inc (X : in out T)
--# derives X from X;
is
begin
  X := X + 1;
end Inc;
    
```

Unsimplified verification condition

```

procedure_inc_1.
H1: true .
H2: x >= t__first .
H3: x <= t__last .
  ->
C1: x + 1 >= t__first .
C2: x + 1 <= t__last .
    
```

Copyright © Praxis HIS - 2006 Slide 23



## Example: run-time error proof

```

type T is range 0 .. 100;

procedure Inc (X : in out T)
--# derives X from X;
is
begin
  X := X + 1;
end Inc;
    
```

### Simplified verification condition

```

procedure_inc_1.
H1:  x >= 0 .
H2:  x <= 100 .
->
C1:  x <= 99 .
    
```

Can't be proved - problem!

Copyright © Praxis HIS - 2006

Slide 24



## Solutions

```

type T is range 0 .. 100;

procedure Inc (X : in out T)
--# derives X from X;
is
begin
  if X < T'Last then
    X := X + 1;
  end if;
end Inc;
    
```

Defensive programming

Logical guard

```

type T is range 0 .. 100;

procedure Inc (X : in out T)
--# derives X from X;
--# pre X < T'Last;
is
begin
  X := X + 1;
end Inc;
    
```

Copyright © Praxis HIS - 2006

Slide 25



## Proof of other properties

- In addition to the proof of the absence of such “run-time errors”, SPARK also allows for proof of:
  - Partial correctness with respect to some specification
  - Invariant properties
- The latter can be safety- and/or security-properties...
  - Basically, anything that can be expressed as a first-order assertion in Hoare-logic!

Copyright © Praxis HIS - 2006

Slide 26



## Agenda

- What is SPARK?
- What can we do with it?
- Brief project examples

Copyright © Praxis HIS - 2006

Slide 27



Copyright © Praxis HIS - 2006

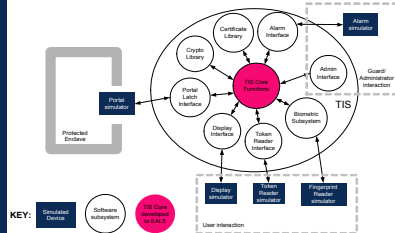
Slide 28

## MULTOS Certification Authority

- digital certificates and crypto keys for smart cards
- ITSEC Level E6
- high-availability (6 months+)
- 100kloc of SPARK, C++, SQL, Ada95 and C
- 28 loc/day, 0.04 defects/kloc, 1 year warranty

## NATO C3 Comms

- secure “filter” for routing and “downgrading” of classified material
- security certification evidence natural by-product of development process



Copyright © Praxis HIS - 2006

Slide 29

## Secure Biometrics

- Technical demo for the NSA
- Common Criteria EAL 5+
- Z, SPARK
- Proof of security properties
- 10kloc at 38 loc/day
- zero defects 20 months after delivery
- cheaper than original system

## Janus

- High-grade, programmable cryptographic engine
- by Rockwell Collins, USA
- SPARK selected to “verify security requirements ... in constrained budget and schedule”



## Conclusions

- A **precise** programming language, designed for analysis completely changes the way we build software
- Emphasis on error **prevention** rather than error **detection**
- Replace “seeking suspicious constructs” with “**prove system has desired properties**”
- Modifies engineers’ behaviour towards rigour and discipline
- We call it “**Correctness by Construction**”
- It is **better** and **cheaper**.

Copyright © Praxis HIS - 2006

Slide 30



## Contacts and Questions

Praxis High Integrity Systems

20 Manvers Street

Bath BA1 1PX

[www.praxis-his.com](http://www.praxis-his.com)

[www.sparkada.com](http://www.sparkada.com)

[sparkinfo@praxis-his.com](mailto:sparkinfo@praxis-his.com)

Copyright © Praxis HIS - 2006

Slide 31



## Resources

- Cook, David. *Evolution of Programming Languages and Why a Language Is Not Enough to Solve Our Problems*. Crosstalk Dec 99. pp 7-12 (<http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1999/12/cook.asp>)
- Amey, Peter. *Correctness by Construction - Better Can Also be Cheaper*. Crosstalk March 2002 pp 24 -28. ([http://www.praxis-his.com/pdfs/c\\_by\\_c\\_better\\_cheaper.pdf](http://www.praxis-his.com/pdfs/c_by_c_better_cheaper.pdf))
- ISO/IEC JTC1/SC22/WG9. *Programming Languages - Guide for the Use of the Ada Programming Language in High Integrity Systems*. ([www.dkuug.dk/jtc1/sc22/wg9/n359.pdf](http://www.dkuug.dk/jtc1/sc22/wg9/n359.pdf))
- German, Andy, *Software Static Code Analysis Lessons Learned*. Crosstalk Nov 2003. pp 13-17. (<http://www.stsc.hill.af.mil/crosstalk/2003/11/0311German.pdf>)
- Hall, Anthony and Chapman, Roderick: *"Correctness By Construction: Developing a Commercial Secure System"*, IEEE Software, Jan/Feb 2002, pp18-25 ([http://www.praxis-his.com/pdfs/c\\_by\\_c\\_secure\\_system.pdf](http://www.praxis-his.com/pdfs/c_by_c_secure_system.pdf))
- King, Steve; Hammond, Jonathan; Chapman, Rod and Pryor, Andy: *"Is Proof More Cost Effective Than Testing?"*, IEEE Transactions on Software Engineering, Volume 26 Number 8 ([http://www.praxis-his.com/pdfs/cost\\_effective\\_proof.pdf](http://www.praxis-his.com/pdfs/cost_effective_proof.pdf))

Copyright © Praxis HIS - 2006

Slide 32



## Resources (contd.)

- Butler, Ricky W., and George B. Finelli, eds. *"The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software."* IEEE Transactions on Software Engineering 19(1): 3-12. (<http://shemesh.larc.nasa.gov/paper-nonq/nonq-paper.pdf>)
- Littlewood & Strigini *"Validation of Ultrahigh Dependability for Software-based Systems"*.. CACM Nov 1993 ([http://www.csr.city.ac.uk/people/lorenzo.strigini/ls.papers/CACMnov93\\_limits/CACMnov93.pdf](http://www.csr.city.ac.uk/people/lorenzo.strigini/ls.papers/CACMnov93_limits/CACMnov93.pdf))
- Amey, Peter. *"A Language for Systems not Just Software"*. ACM SigAda 2001. ([http://www.praxis-his.com/pdfs/systems\\_not\\_just\\_sw.pdf](http://www.praxis-his.com/pdfs/systems_not_just_sw.pdf))
- Chapman, Rod., Amey, Peter. *"Industrial Strength Exception Freedom"*. Proceedings of ACM SigAda 2002. ([http://www.praxis-his.com/pdfs/Industrial\\_strength.pdf](http://www.praxis-his.com/pdfs/Industrial_strength.pdf))
- Chapman, Rod; Hilton, Adrian: *"Enforcing Security and Safety Models with an Information Flow Analysis Tool"*. Proceedings of ACM SIGAda 2004 ([http://www.praxis-his.com/sparkada/pdfs/infoflow\\_paper.pdf](http://www.praxis-his.com/sparkada/pdfs/infoflow_paper.pdf))
- Peter Amey, Rod Chapman, Neil White: *"Smart Certification Of Mixed Criticality Systems"*. Ada Europe 2005 ([http://www.praxis-his.com/sparkada/pdfs/Smart\\_Certification.pdf](http://www.praxis-his.com/sparkada/pdfs/Smart_Certification.pdf))
- Janet Barnes, Rod Chapman: *"Engineering the Tokeneer Enclave Protection Software"*. Proceedings of IEEE ISSSE 2006

Copyright © Praxis HIS - 2006

Slide 33



## Resources (contd.)

- Amey, Peter,. and White, Neil. "*High Integrity Ada in a UML and C World*". Lecture Notes in Computer Science 3063  
A. Llamosi, A. Strohmeier (Eds.): Reliable Software Technologies – Ada-Europe 2004 9th Ada-Europe International Conference, La Palma de Mallorca, June 2004, pp. 225-236. ([http://www.praxis-his.com/sparkada/pdfs/ada\\_uml\\_and\\_c.pdf](http://www.praxis-his.com/sparkada/pdfs/ada_uml_and_c.pdf))
- See also [www.sparkada.com](http://www.sparkada.com)