

SRG

Software Reliability Group

**State of the art and the practice of
computational systems
validation and verification**

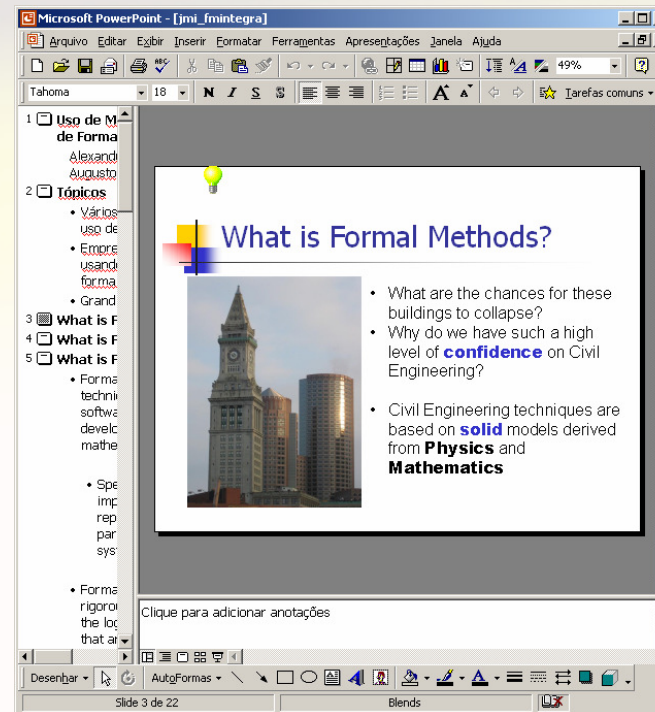
Augusto Sampaio & Alexandre Mota



Agenda

- Motivation
- Formal Methods
- Validation and verification
 - Scenarios
 - Examples
 - Industrial applications
 - Grand challenges
- Summary

Motivation



- Which artifact is more likely to collapse?
- What are the foundations of the two disciplines?
- What about their maturity?

What is a Formal Method?

- A system **development method** that ...
 - allows one to precisely define a problem and
 - progress towards an implementation
 - proven correct** with respect to the given specification

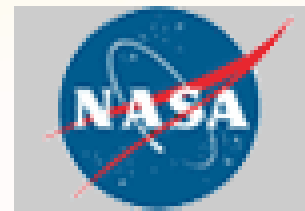
Formal Methods Use

- Comprises **techniques** and **tools** for software and hardware development based on **mathematical logic**
 - **Specifications, designs and implementations** are represented in **logical systems**
- **Verification and validation (V&V)** achieved through **deduction** in the logic

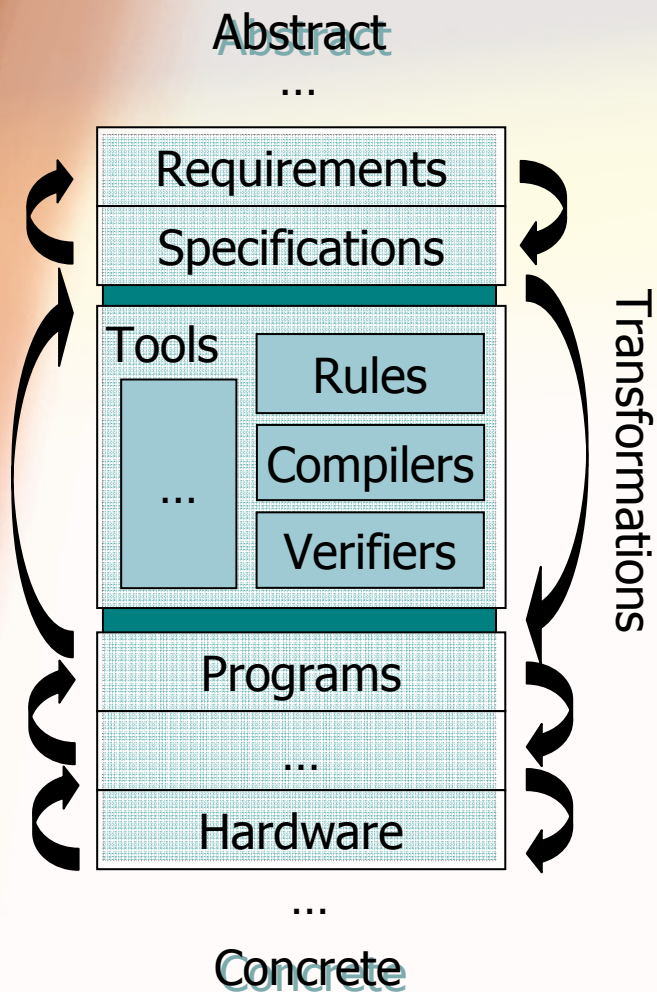
Formal Methods and Testing

- Complementary V&V techniques
 - **Testing** tend to be of more practical appeal
 - but ... cannot guarantee full **correctness**
 - **Formal methods** allow full analysis (e.g. model checking) in principle
 - but ... apply to models (rather than to physical implementations) and require more expertise

Formal Methods in Industry



The Ideal V&V Scenario



Different presentations...

- Natural Language
- Specification language
- Programming languages and their auxiliary tools
- Executable code
- Virtual Machines
- Logic gates

but the same meaning.

- Precise semantics required
- Transformations must preserve behaviour

Syntax

Semantics

Tool support and training!

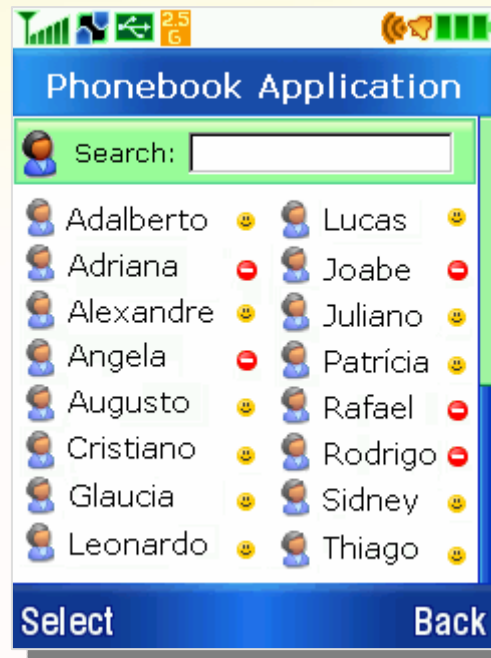
The ideal V&V scenario: Correct code by construction!

Specification



Code

Example: Phonebook Application



Example: Phonebook Application

- Operations:
 - Create New Contact:
 - Takes a contact name and a phone number and, if the memory is not full, adds them to the contact list
 - Search Contact:
 - Returns the list of contacts that match a given string `s`

Refinement Calculus

- System behaviour captured by logical specifications
- Laws (rules) applied to transform a specification into a program
- Each law must be proved to preserve semantics

Example: just a small flavour Phonebook Application

■ Specification: Create New Contact

$c, n, msg: [true,$ \leftarrow precondition *true* – this operation is defined for all states of the system

case there is memory space \leftarrow $(n < MemSize \wedge$
 $c[n] = (newName, newNumber)$
 $\wedge \dots \textit{rest of } c \textit{ preserved } \dots$
 $\wedge n = n_0 + 1$
 $\wedge msg = \textit{“Success”})$

case there is **not** memory space $\leftarrow \vee (n \geq MemSize \wedge c = c_0 \wedge n = n_0 \wedge msg = \textit{“Memory full”})]$

Refining Create New Contact

- Law 4.1 – If_then_else

$$w:[true, (b \wedge p) \vee (\neg b \wedge q)]$$
$$\subseteq$$
$$\text{if } b \text{ then } p \text{ else } q \text{ fi}$$

Refining Create New Contact

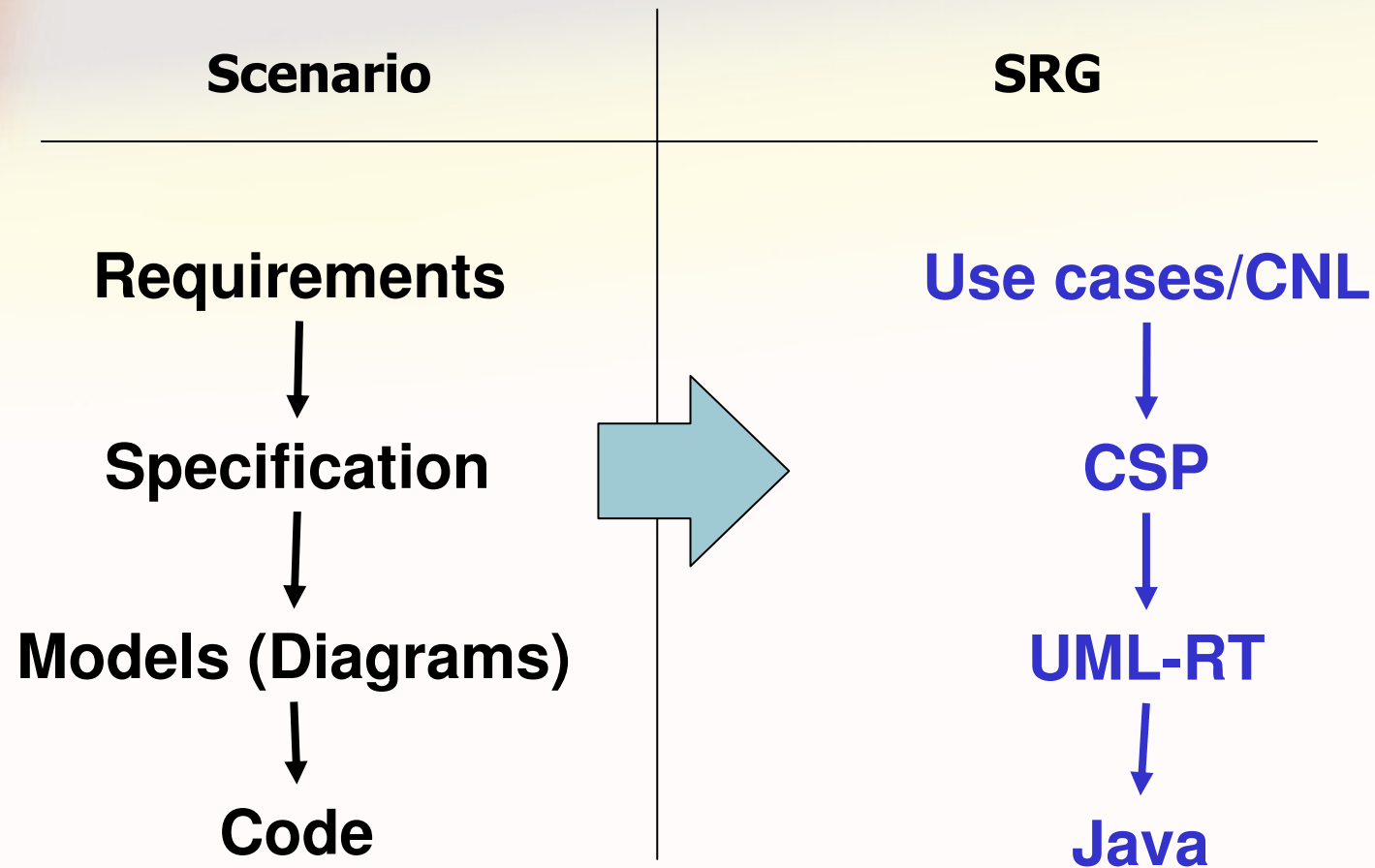
- Applying Law 4.1 – If_then_else

```
if n < MemSize then  
    c[n] := (newName, newNumber);  
    msg := "Saved";  
    n := n+1;  
  
else  
    msg := "Memory full";  
  
fi
```

Challenges in practice

- The languages must have **formal semantics**
 - **Link with informal requirements** not addressed
- The laws cannot be applied fully automatically
 - Interaction with a **theorem prover** is necessary
- **Data refinement**
 - Some types in the specification have no correspondence in the implementation
 - Change of representation is not fully automatic

Scenario: Model-driven development



Use cases: from CNL to CSP

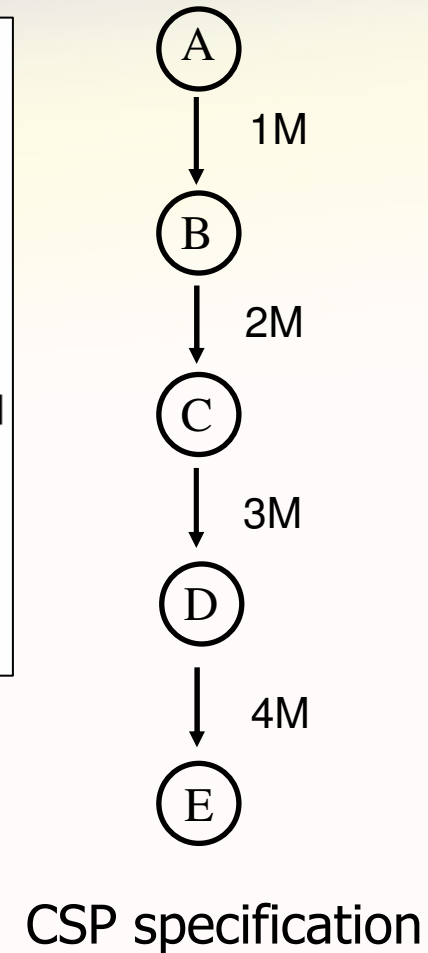
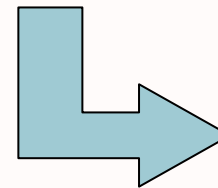
Main Flow

Description: Create a new contact
 From Step: START
 To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation. [TRS_11111_101]	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

CNL representation

CNL
processing



Use cases: from CNL to CSP

Main Flow

From Steps: [Main Flow] [OLE D]

Step ID	Actor Action	System State	System Response
CE1	Center [action]	Condition [condition]	Response "Hot Message" folder is displayed after "Tab" "Hot Message" folder is displayed before "Voice Mail" response [step]
CE2	Go to "Tab" [action]	Condition [condition]	Response All inbox Messages are displayed [response]
CE3	Scroll to a non-Class 2 message [action]	Condition [condition]	Response Message is highlighted [response]
CE4	Go to CSM [action]	Condition [condition]	Response Move to Hot Messages option is displayed [response]

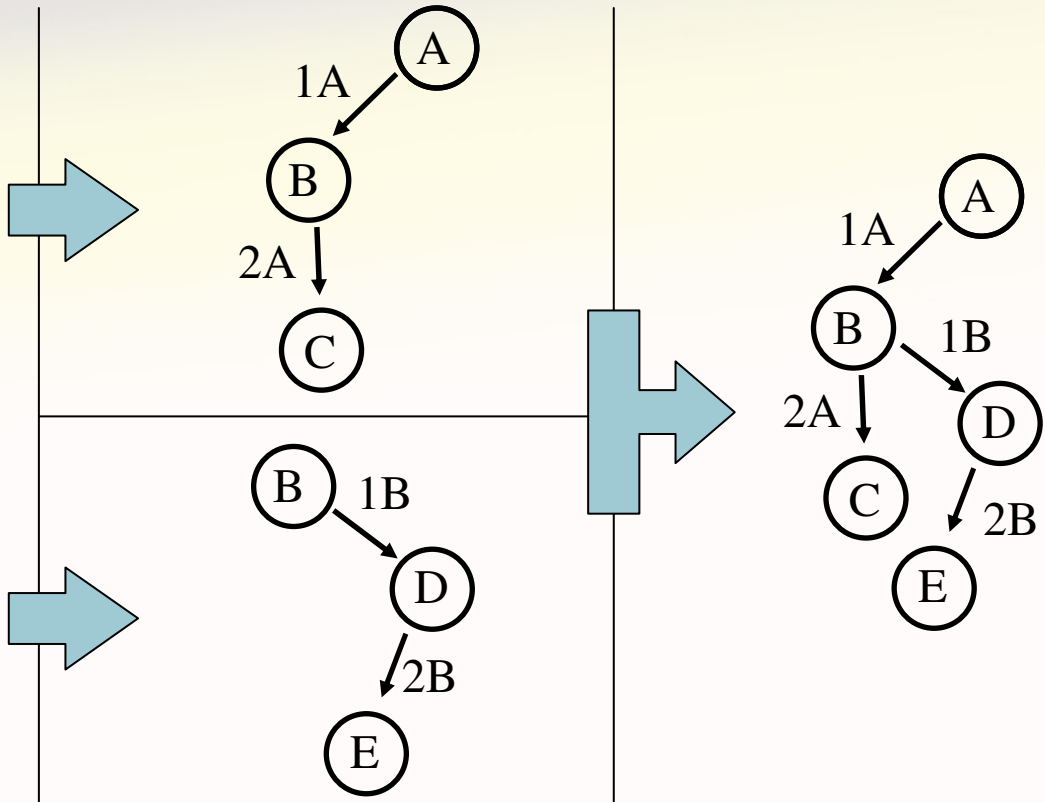
Main Flow

Main Flow

From Steps: [Main Flow] [OLE D]

Step ID	Actor Action	System State	System Response
CE1	Center [action]	Condition [condition]	Response "Hot Message" folder is displayed after "Tab" "Hot Message" folder is displayed before "Voice Mail" response [step]
CE2	Go to "Tab" [action]	Condition [condition]	Response All inbox Messages are displayed [response]
CE3	Scroll to a non-Class 2 message [action]	Condition [condition]	Response Message is highlighted [response]
CE4	Go to CSM [action]	Condition [condition]	Response Move to Hot Messages option is displayed [response]

Alternative Flow

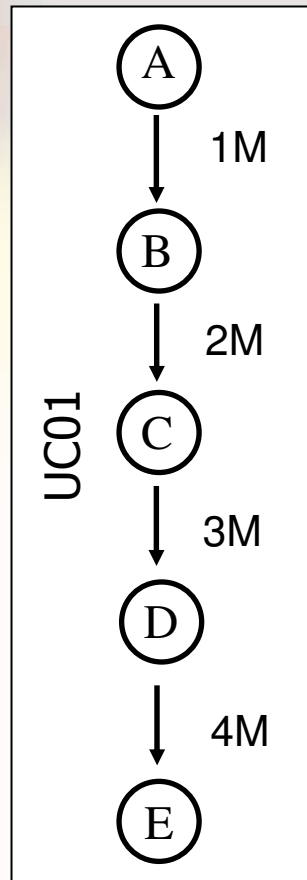


CNL representation

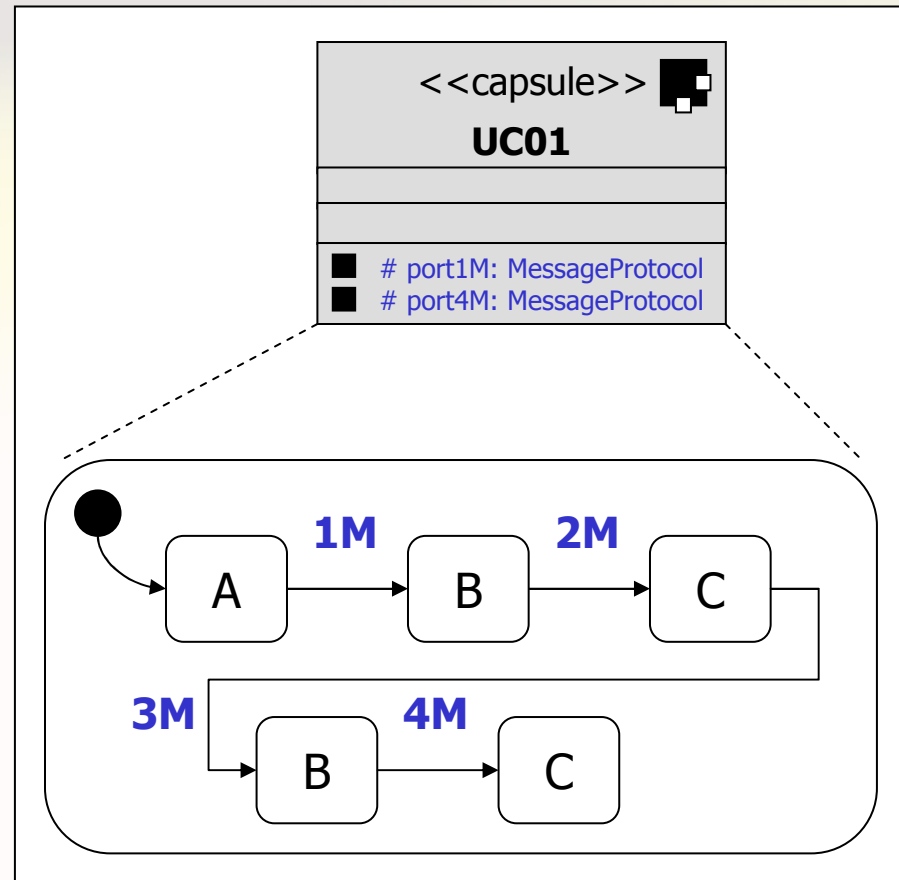
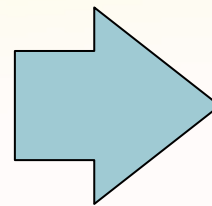
CSP processes

CSP specification

CSP to UML-RT mapping



CSP specification



UML-RT model

UML-RT to code

- Pragmatically
 - Rational Rose RealTime
 - Framework allows developing the entire application using capsules, protocols and ports and **Generate code** or **Build** the application
 - ***RTJava, RTC++ and RTC***

Critical view of the scenario

Use cases/CNL



CSP



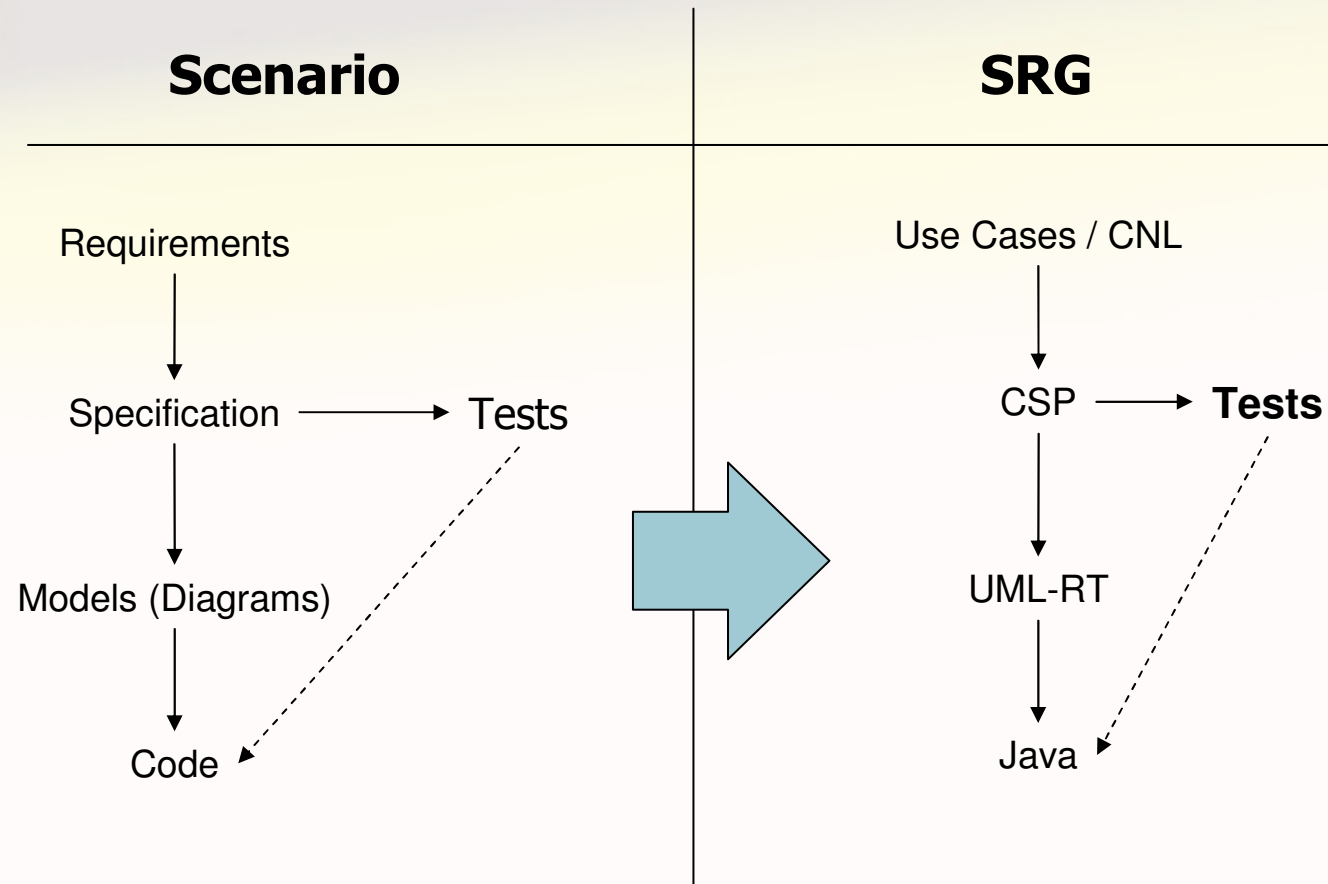
UML-RT



Java

- Feasible in practice, but ...
 - does not identify errors in requirements
 - does not guarantee correctness of generated code

Incorporating **model based testing** (verification)



TaRGeT Test Generation

Feature 11111 - My Phonebook

UC 01 - Creating a New Contact

Description

This use case describes the creation of a new contact.

Main Flow

Description: Create a new contact

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation. [TRS_11111_101]	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

Exception Flows

Description: There is no enough memory.

From Step: 3M

To Step: END

Step Id	User Action	System State	System Response
1A	Confirm the contact creation.	There is no enough phone memory.	A dialog is displayed informing that there is no enough memory. [TRS_111166_103]

Case Description	Procedure	Expected Results
TC_1		
Requirements:		
Initial Conditions:		
Test Procedure:		

TaRGeT Test Generation

Feature 11111 - My Phonebook

UC 01 - Creating a New Contact

Description

This use case describes the creation of a new contact.

Main Flow

Description: Create a new contact

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation. [TRS_11111_101]	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

Exception Flows

Description: There is no enough memory.

From Step: 3M

To Step: END

Step Id	User Action	System State	System Response
1A	Confirm the contact creation.	There is no enough phone memory.	A dialog is displayed informing that there is no enough memory. [TRS_111166_103]

Case Description	Procedure	Expected Results
TC_1		
Requirements:		
Initial Conditions:		
Test Procedure:		

TaRGeT Test Generation

Feature 11111 - My Phonebook

UC 01 - Creating a New Contact

Description

This use case describes the creation of a new contact.

Main Flow

Description: Create a new contact

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation. [TRS_11111_101]	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

Exception Flows

Description: There is no enough memory.

From Step: 3M

To Step: END

Step Id	User Action	System State	System Response
1A	Confirm the contact creation.	There is no enough phone memory.	A dialog is displayed informing that there is no enough memory. [TRS_111166_103]

Case Description	Procedure	Expected Results
TC_1		
Requirements:		
Initial Conditions:		
Test Procedure:		
1	Start My Phonebook application.	My Phonebook application menu is displayed.
2	Select the New Contact option.	The New Contact form is displayed.
3	Type the contact name and the phone number.	The new contact form is filled.
4	Confirm the contact creation.	A dialog is displayed informing that there is no enough memory.

TaRGeT Test Generation

Feature 11111 - My Phonebook

UC 01 - Creating a New Contact

Description

This use case describes the creation of a new contact.

Main Flow

Description: Create a new contact

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation. [TRS_11111_101]	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

Exception Flows

Description: There is no enough memory.

From Step: 3M

To Step: END

Step Id	User Action	System State	System Response
1A	Confirm the contact creation.	There is no enough phone memory.	A dialog is displayed informing that there is no enough memory. [TRS_111166_103]

Case Description	Procedure	Expected Results
TC_1		
Requirements:	TRS_111166_103	
Initial Conditions:		
Test Procedure:		
1	Start My Phonebook application.	My Phonebook application menu is displayed.
2	Select the New Contact option.	The New Contact form is displayed.
3	Type the contact name and the phone number.	The new contact form is filled.
4	Confirm the contact creation.	A dialog is displayed informing that there is no enough memory.

TaRGeT Test Generation

Feature 11111 - My Phonebook

UC 01 - Creating a New Contact

Description

This use case describes the creation of a new contact.

Main Flow

Description: Create a new contact

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation. [TRS_11111_101]	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

Exception Flows

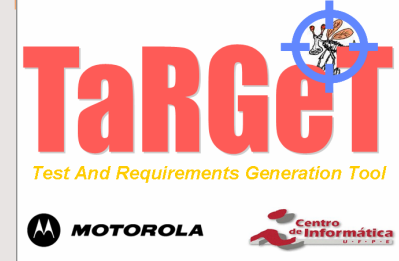
Description: There is no enough memory.

From Step: 3M

To Step: END

Step Id	User Action	System State	System Response
1A	Confirm the contact creation.	There is no enough phone memory.	A dialog is displayed informing that there is no enough memory. [TRS_111166_103]

Case Description	Procedure	Expected Results
TC_1		
Requirements:	TRS_111166_103	
Initial Conditions:	My Phonebook application is installed in the phone. There is no enough phone memory.	
Test Procedure:		
1	Start My Phonebook application.	My Phonebook application menu is displayed.
2	Select the New Contact option.	The New Contact form is displayed.
3	Type the contact name and the phone number.	The new contact form is filled.
4	Confirm the contact creation.	A dialog is displayed informing that there is no enough memory.



The TaRGeT Tool

- Test and Requirements Generation Tool
- Integrates the researches from BTC-RD project
 - Reads structured use case documents
 - Generates
 - Internal model (CSP or LTS)
 - test suites (complete and guided)
 - traceability information
 - Tool in constant evolution

TaRGeT Workspace

The screenshot shows the TaRGeT Requirements workspace. The main window displays the details for use case UC_111. The left pane shows a tree view of use cases, and the right pane shows the details for the selected use case. The bottom pane shows the artifacts generated for this use case.

UC_111 - Generating Test Suites with Test Purposes

Description:

This use case describes how the test suites are generated using test purposes within TaRGeT.

Related requirement(s)

Requirements Codes
FR_TARGET_0155
FR_TARGET_0160
FR_TARGET_0151
FR_TARGET_0145

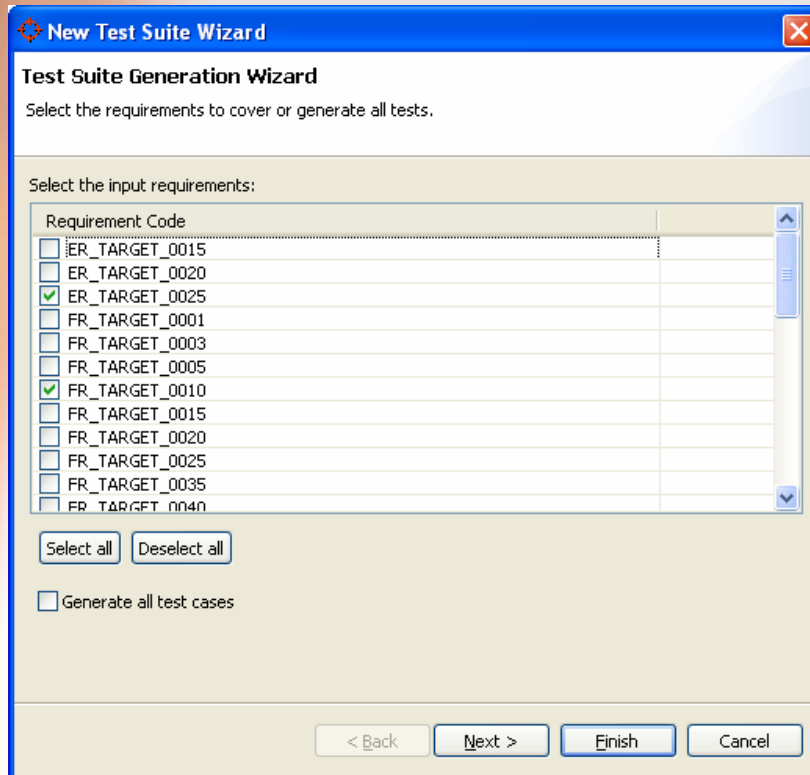
Main Flow

Description: Generate test cases using one test purpose composed of one step.
 From Steps: UC_110#2M
 To Steps: END

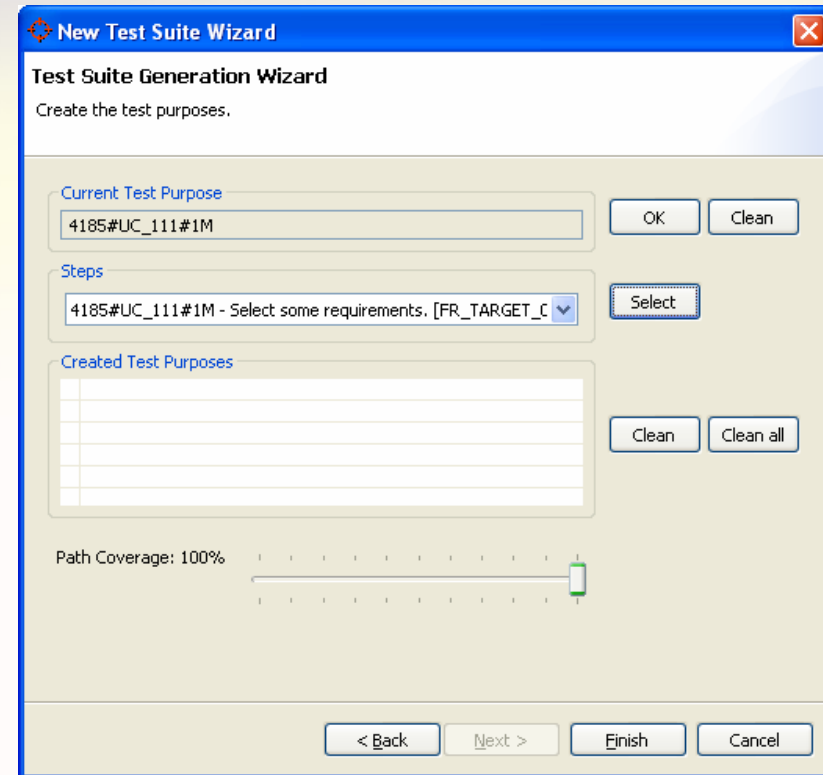
Artifacts

- Features
 - ZBR05-DES-06-4185.doc
- TestCases
 - NewTestSuite1.xls

Test Generation Options



Test generation from selected requirements



Test generation from test purposes

Generated Artifacts

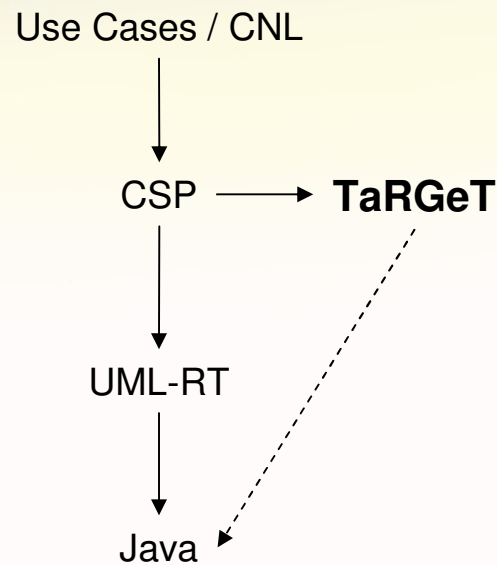
3	na	Man	TC_3	Generate test cases by requirements. Cancel the generation.	
			Use Cases:	4185#UC_110	
			Requirements:	FR_TARGET_0140, FR_TARGET_0145	
			Setup:		
			Initial Conditions:	The TaRGeT is already started up. A project is opened and some use case documentation is already imported. No error is listed in the "Error" list. The imported use cases have more than one step which refers to distinct requirements.	
			Notes:	Test case auto-generated by TaRGeT system.	
			Test Procedure (Step Number):		
			1	Choose "Tools" option in the menu bar.	A drop down menu is displayed. "Test Generation" option is available.
			2	Choose "Generate Test Suite" option in the drop down menu.	The "Test Generation" window is displayed. The "Requirements List" is displayed, showing all requirements referred in the valid imported documents. The "Finish" button is disabled.
			3	Select some requirements (not all).	"Finish" button is enabled.
			4	Click on "Cancel" button.	The focus goes back to the work area.
			Final Conditions:		
			Cleanup:		

Test suite design for manual test execution

Requirements Traceability Matrix - System Test	
Requirement	System Test Case
REQ - ER_TARGET_0015	TES (TC 46)
REQ - ER_TARGET_0020	TES (TC 46, TC 47, TC 60)
REQ - ER_TARGET_0025	TES (TC 46, TC 47)
REQ - FR_TARGET_0001	TES (TC 33, TC 34, TC 35, TC 36,
REQ - FR_TARGET_0003	TES (TC 28, TC 30, TC 38, TC 39, TC 40, TC 41, TC 42, TC 43,
REQ - FR_TARGET_0005	TES (TC 26, TC 27, TC 31, TC 74,
REQ - FR_TARGET_0010	TES (TC 1, TC 2, TC 5, TC 7, TC 8, TC 54, TC 56, TC 58, TC 59, TC 67, TC 78, TC 79,
REQ - FR_TARGET_0015	TES (TC 26, TC 27, TC 31, TC 41, TC 44, TC 78, TC 79, TC 81,
REQ - FR_TARGET_0020	TES (TC 48, TC 49, TC 50, TC 51, TC 52, TC 53, TC 54, TC 55,
REQ - FR_TARGET_0025	TES (TC 33, TC 34, TC 52, TC 53, TC 54, TC 55, TC 56, TC 71,
REQ - FR_TARGET_0035	TES (TC 69, TC 70)
REQ - FR_TARGET_0040	TES (TC 61, TC 62, TC 63, TC 64,
REQ - FR_TARGET_0100	TES (TC 78, TC 79, TC 80, TC 81, TC 82, TC 83, TC 84, TC 85, TC 86, TC 87, TC 88, TC 89,
REQ - FR_TARGET_0110	TES (TC 78, TC 79, TC 81, TC 82, TC 85, TC 87, TC 88, TC 90,
REQ - FR_TARGET_0120	TES (TC 26, TC 27, TC 31, TC 41, TC 81, TC 82, TC 88, TC 90,
REQ - FR_TARGET_0125	TES (TC 26, TC 27, TC 31, TC 81,

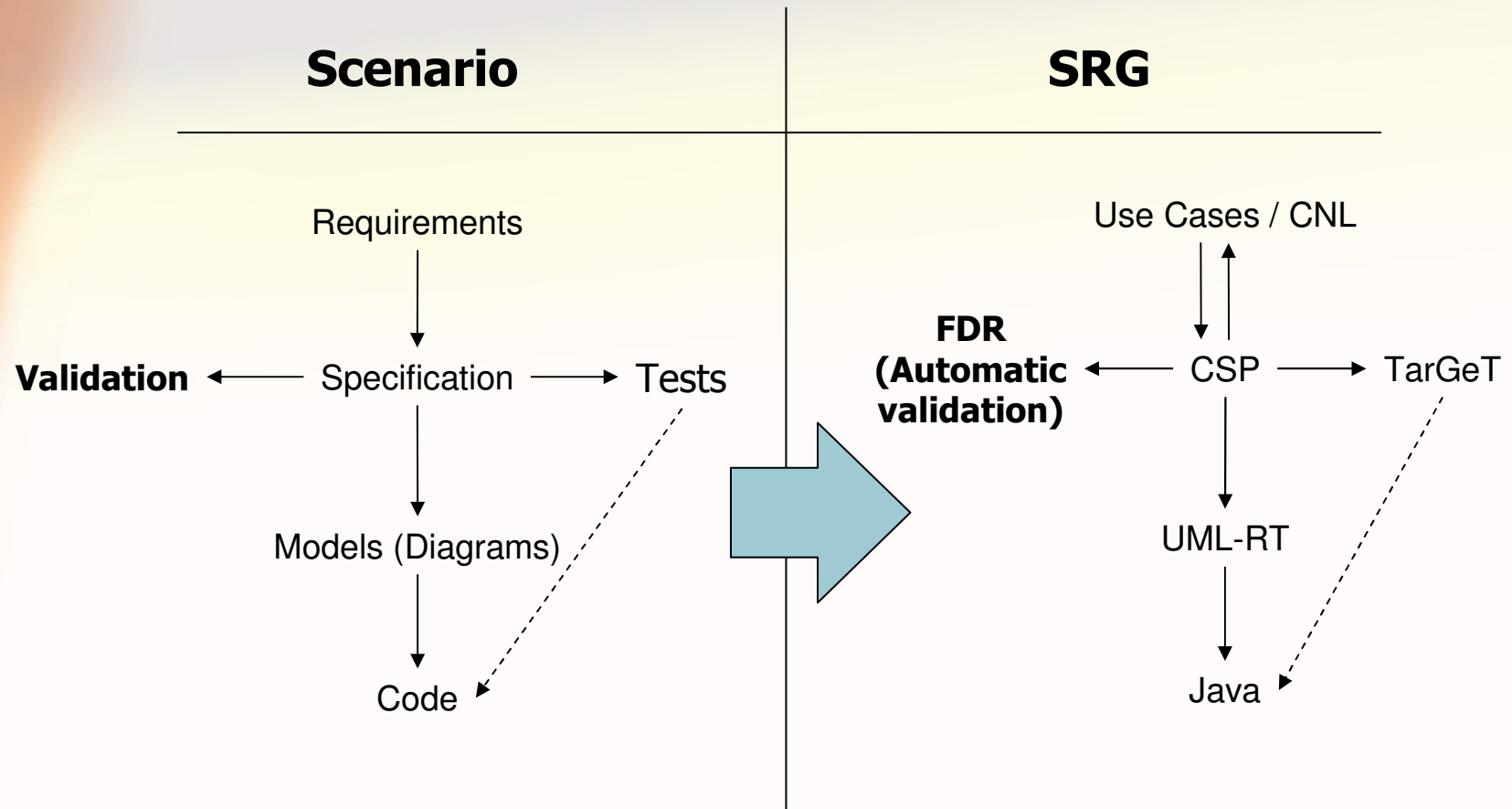
Traceability information (UC x Reqs, UC x TCs, Reqs x TCs)

Critical view of the scenario



- This scenario alone can not guarantee code was complete verified
 - *“Tests can only find errors but not prove their absence”*
 - What about errors in the use case specification?

Incorporating model checking (automatic validation)



FDR: Refinement Checking

FDR 2.82 interface showing the Livelock checking results. The window title is "FDR 2.82". The menu bar includes File, Assert, Process, Options, Interrupt, FormalSystems, and Help. The toolbar includes Refinement, Deadlock, Livelock, Determinism, and Evaluate. The Livelock section is active, showing an Implementation field with a dropdown arrow and a "Check" button. Below the Implementation field, there are "Add" and "Clear" buttons. The results pane shows the following checks:

- ✓ System deadlock free [F]
- ✓ System01 deadlock free [F]
- ✓ System livelock free
- ✓ System01 livelock free

At the bottom, the session path is: DR2 session: /export/home/wcb065/checker-beta4.linux6/UC-speclmp.cs

FDR 2.82 interface showing the Refinement checking results. The window title is "FDR 2.82". The menu bar includes File, Assert, Process, Options, Interrupt, FormalSystems, and Help. The toolbar includes Refinement, Deadlock, Livelock, Determinism, and Evaluate. The Refinement section is active, showing Specification and Implementation fields with dropdown arrows, a Model dropdown set to "Failures-divergence", and "Check", "Add", and "Clear" buttons. The results pane shows the following checks:

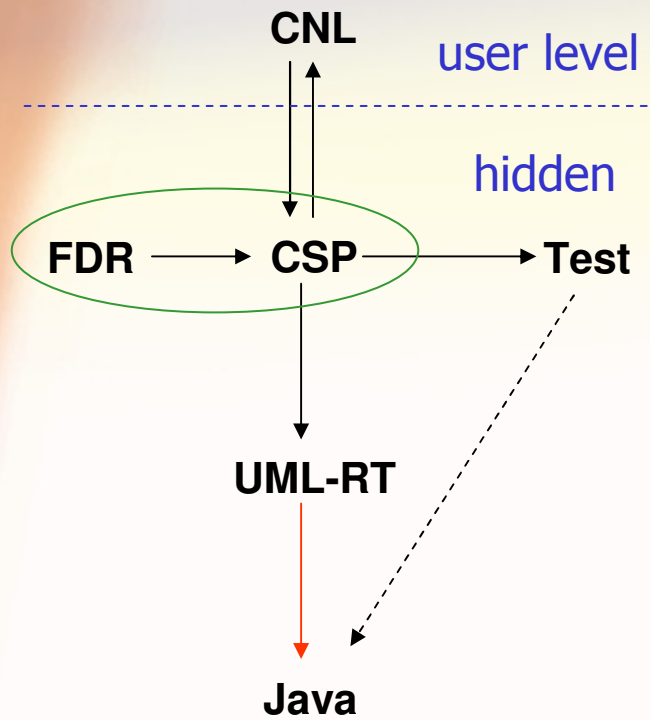
- ✓ SysSpec [T= SysImpl]
- ✓ SysImpl [T= SysSpec]
- ✓ SysSpec [FD= SysImpl]
- ✓ SysImpl [FD= SysSpec]

Below the results, there are two scrollable panes. The first pane contains:

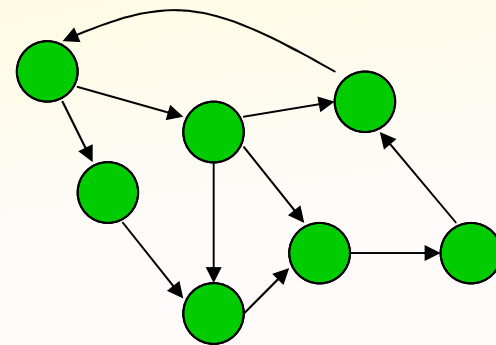
- CHAOS(-)
- P1
- P1_Hiding
- P2

At the bottom, the session path is: FDR2 session: /export/home/wcb065/checker-beta4.linux6/UC-speclmp.csp

Automatic validation



Model Checking



\models

prop

model

satisfies

property

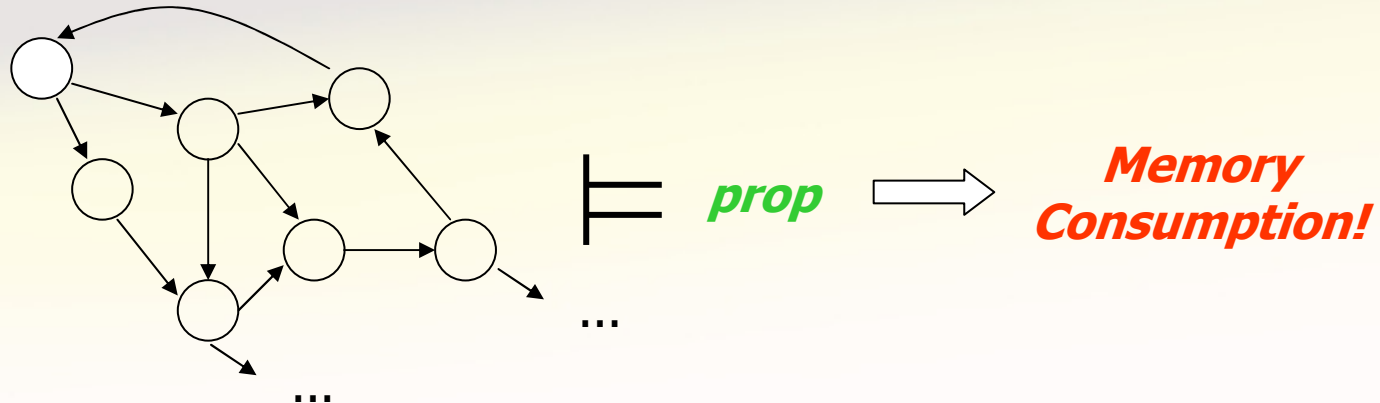
Phonebook

\models

Deadlock-freedom

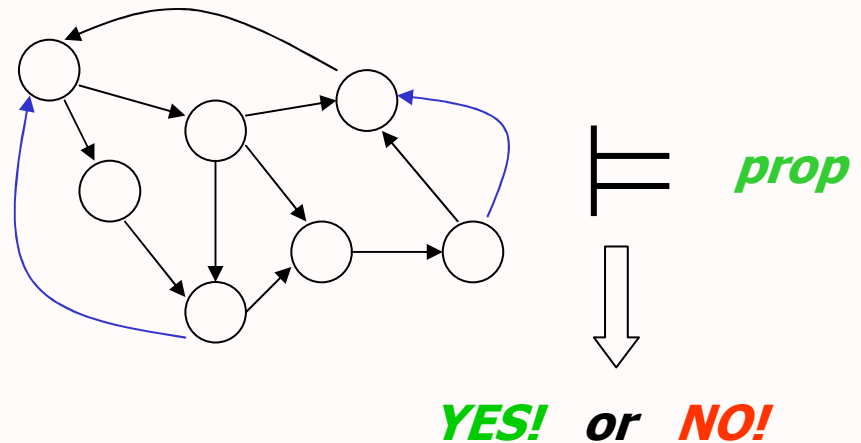
YES!

Automatic validation: Dealing with State Explosion

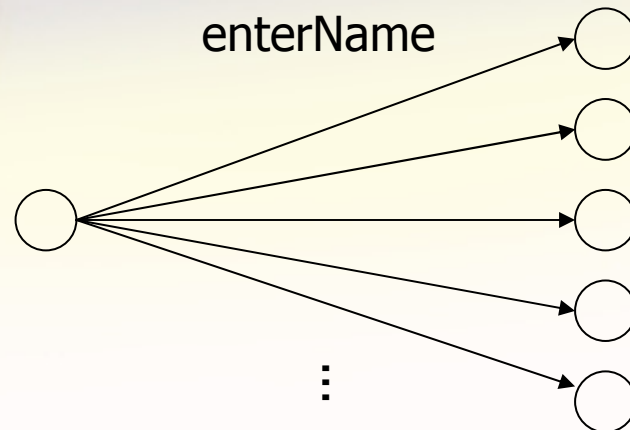


■ Auxiliary techniques:

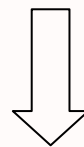
- Data independence
- Data Abstraction
- Process Refinement



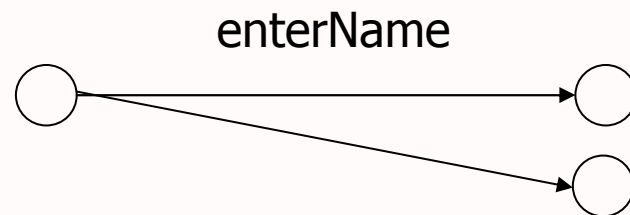
Example: Search a contact



An infinite (or a large) number of names are available

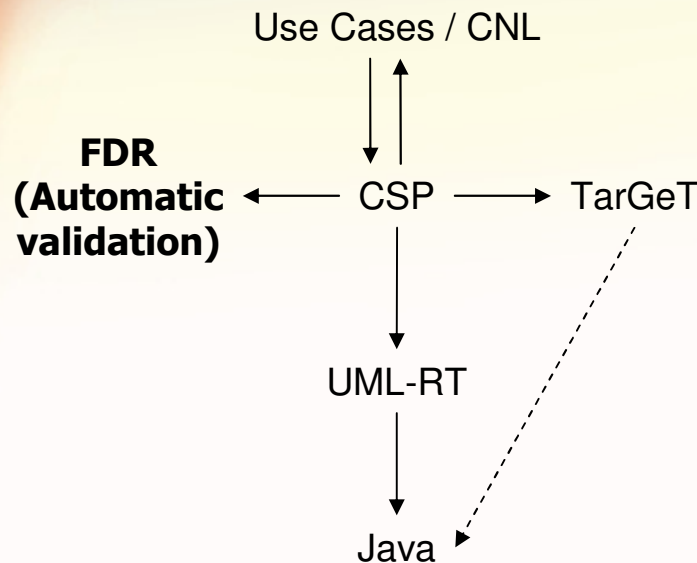


abstraction techniques



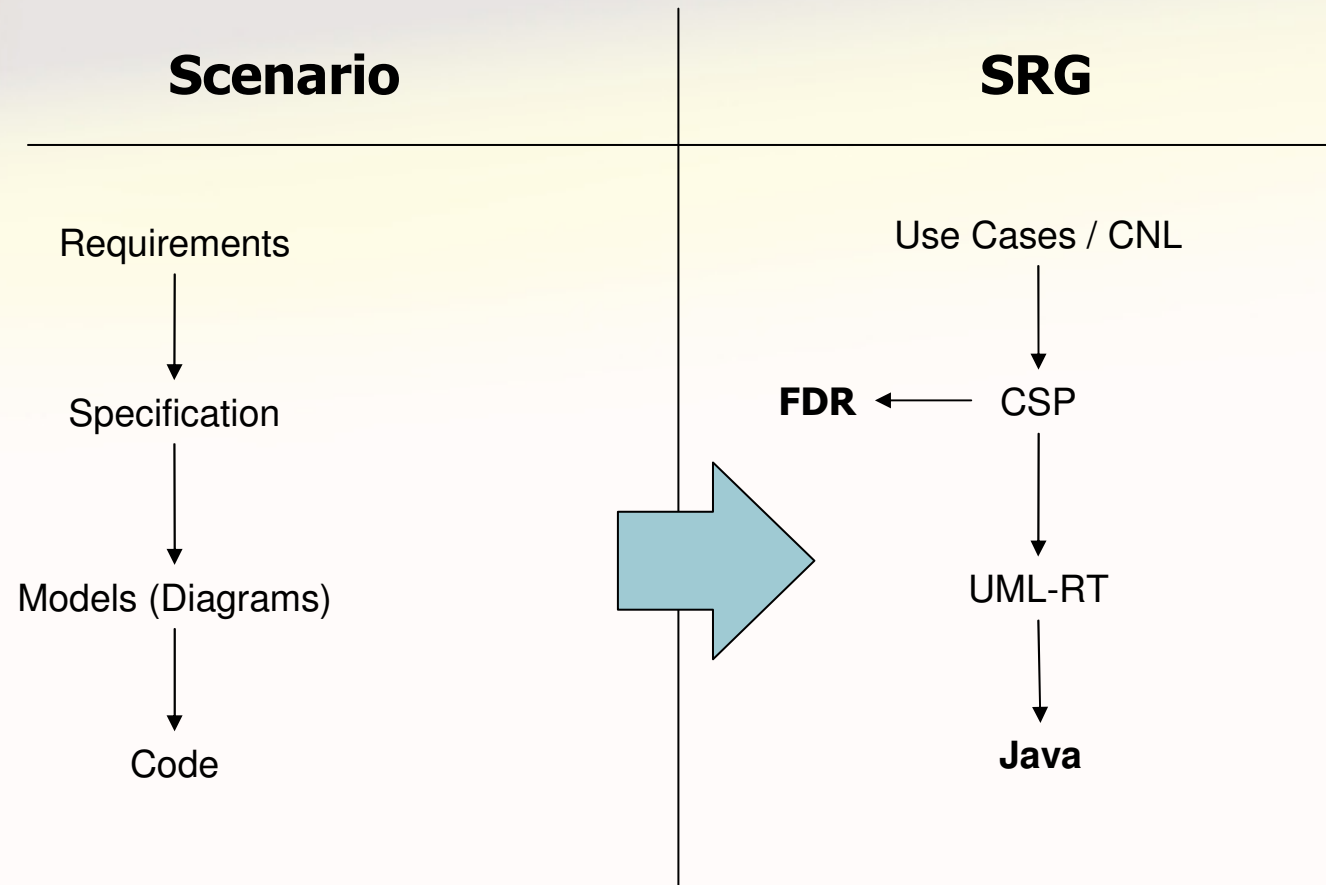
A smaller but sufficient set of names
Are available

Critical view of the scenario



- Automatic validation (full analysis)
- In practice, State explosion
- Mechanise abstraction strategies is a challenge
- Does not guarantee correctness of code

Analysing Code: Software Model Checking



Software Model Checking

- Abstraction level analysis
 - High-Level Approach
 - Very close to requirements
 - Low-Level Approach
 - Very close to design and implementation

High-Level Approach

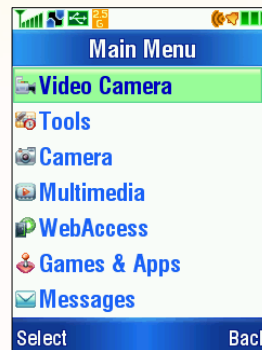
- Obtain models exercising GUI provided behavior
- Models comparable to system-based tests and requirements (FDR)
- BxT prototype
 - Automatic tool that extracts behavioral models from mobile phones [Motorola project]

High-Level Approach

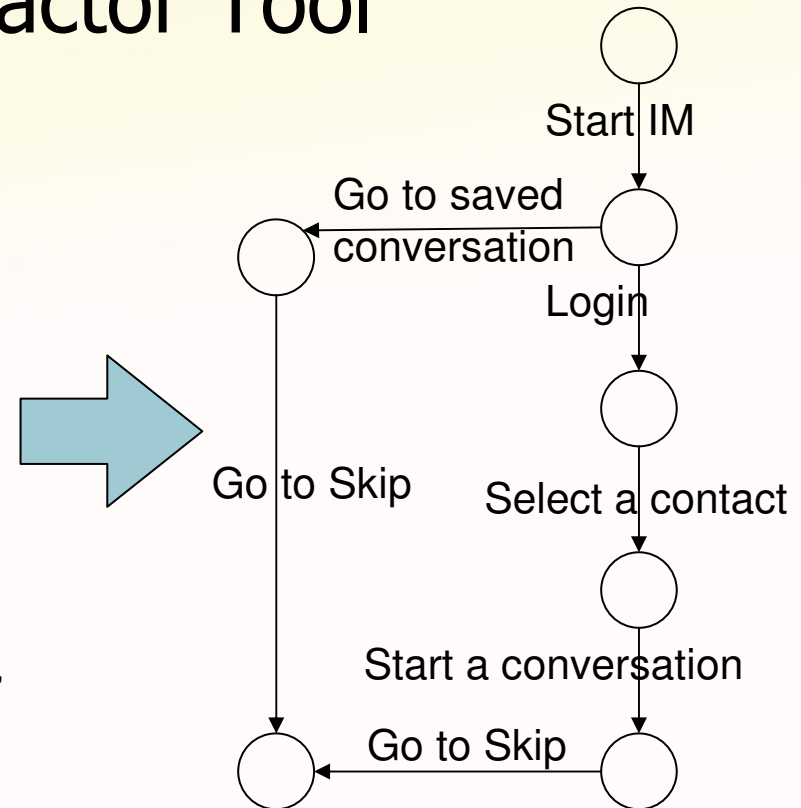
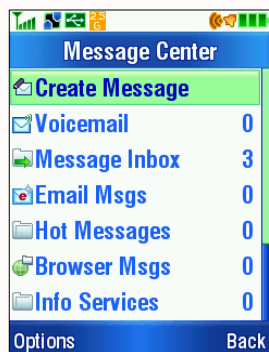
- BXT: Behavior eXtractor Tool



Goto "Main Menu"



Select "Messages"



Low-Level Approach

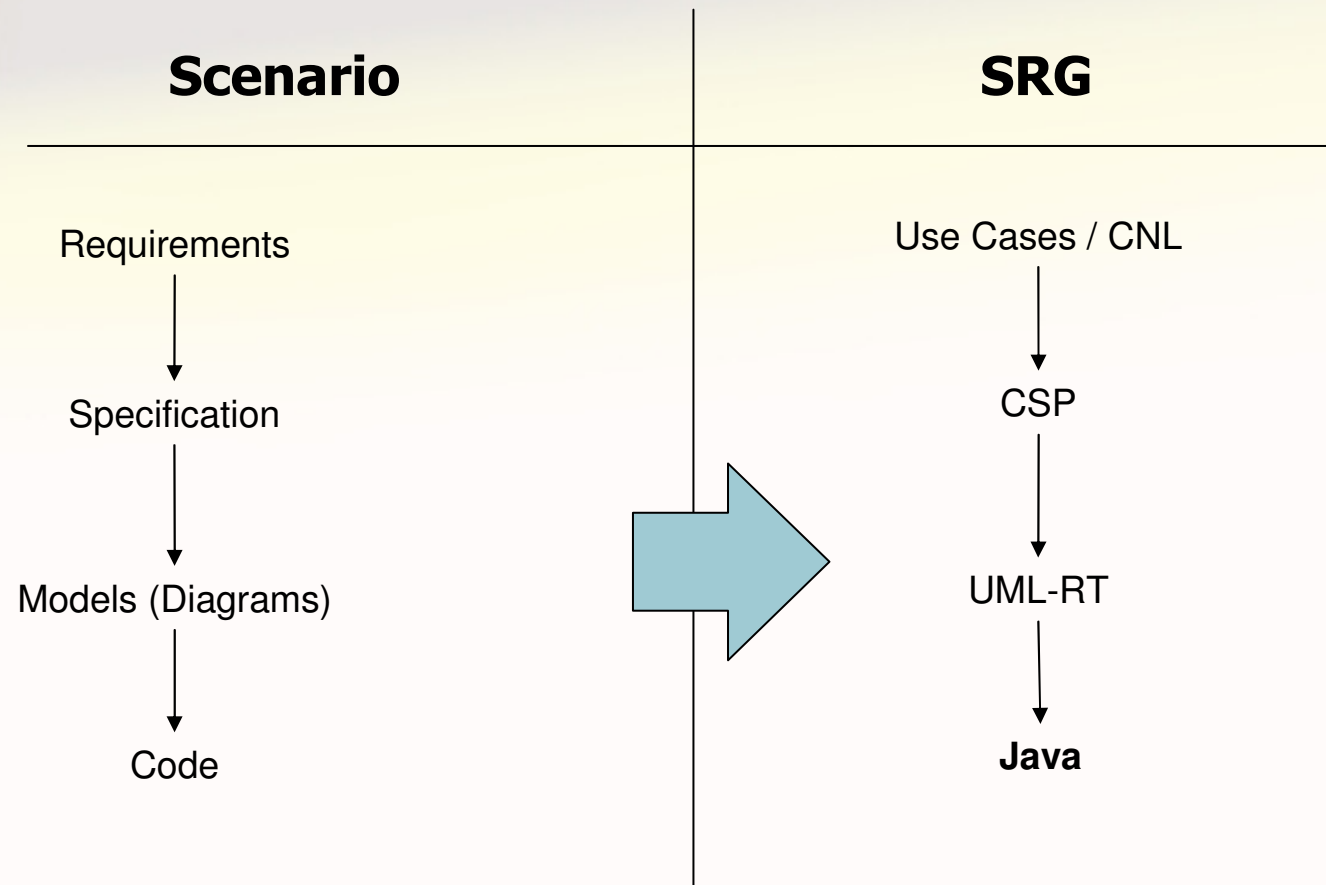
- Java PathFinder (JPF), BANDERA
 - Obtain formal models from code (state changes)
 - Deal with Java bytecode programs (Specific JVM)
 - Explore all potential execution paths to find property violations
 - Design and implementation based properties
- Limitations
 - Code size [~ 10 KLOC or ~ 30 KLOC for specific applications]
 - Limited support for some Java API`s [java.net, java.io]

Low-Level Approach

- JPF default properties:
 - NoDeadlocked
 - NoUncaughtExceptions
 - NoAssertionViolated
 - More Properties can be implemented

```
===== error #1  
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty  
java.lang.NoSuchMethodException: calling javax.swing.JPanel.<init>(Ljava/awt/LayoutManager;)V  
  at PBookGUI.<init>(PBookGUI.java:21)  
  at PBookGUI.main(PBookGUI.java:47)
```

Analysing Code: Static and Dynamic Analysis



Static Program Analysis

- Performed without executing programs
 - In most cases, the source code is used
- Static analysers:
 - Esc/Java (Houdini)
 - Spec# static analyser

ESC/Java (Houdini)

- **ESC/Java = Extended Static Checker for Java**
 - Programming tool that attempts to find common run-time errors in **Java programs**
 - Based on the Java Modeling language (**JML**)
 - It is a specification language for Java programs
 - Specifications are added as annotation comments
- **Houdini**
 - Annotation assistant for ESC/Java
- **Tested on 40 KLOC**
 - It has found a variety of defects

ESC/Java (Houdini) Example


```
class Rational {
  //@ invariant denom != 0;
  int num, denom;
  //@ requires d != 0;

  Rational(int n, int d) {
    num = n;
    denom = d;
  }
  double getDouble() {
    return ((double)num)/denom;
  }
  public static void main(String[] a) {
    int n = readInt(), d = readInt();
    if( d == 0 ) return;
    Rational r = new Rational(d, n);
    print( r.getDouble() );
  }
  ...
}
```

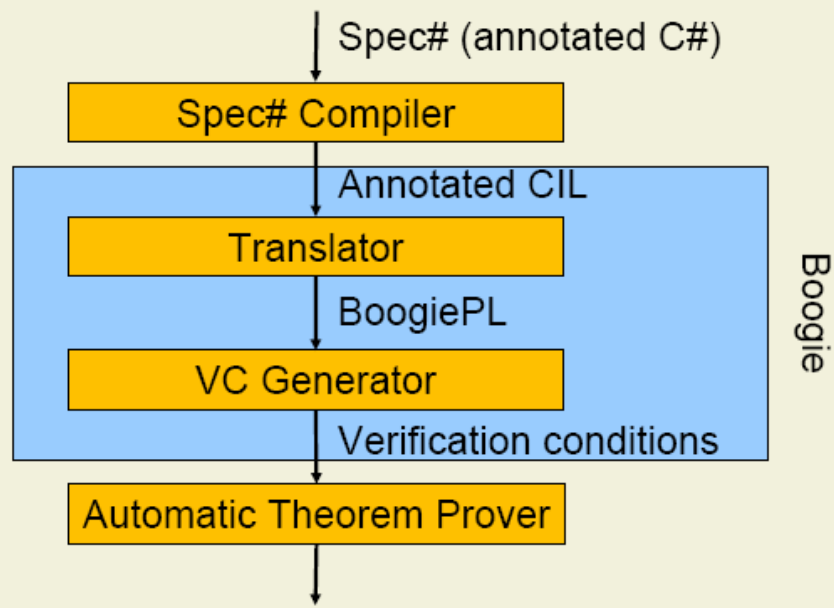
Warning: invariant possibly not established

Warning: possible division by zero

Warning: precondition possibly not established



Spec# - BoogiePL



Picture by Peter Mueller (LASER2007)

- Extension of C# with annotations
 - Invariants
 - Method contracts
 - Fields and type annotations
- Automatic verification (BoogiePL)
 - Type checking
 - Verification condition generation (VC)
 - Automatic theorem proving (ATP)
- Software construction
 - Guided by contracts (annotations)
 - Specifications and implementations are checked, but the specification can also be wrongly defined.

Spec# in Visual Studio

The screenshot shows a Visual Studio window titled 'Chunker - Microsoft Visual Studio'. The code editor displays the following C# code for a class named 'Chunker':

```

public class Chunker
{
    string! src;
    int n; // the number of characters returned so far
    invariant 0 <= n && n <= src.Length;
    public readonly int ChunkSize;
    invariant 0 < ChunkSize;

    public virtual string! NextChunk()
    ensures result.Length == ChunkSize;
    {
        string s = src.Substring(n, ChunkSize);
    }

    public Chunker(string! source, int chunkSize)
    requires 0 < chunkSize;
}
    
```

Annotations in the code include `string!`, `int n;`, `invariant`, `public readonly int`, `invariant`, `public virtual string!`, `ensures`, `string s =`, `requires`, and `int chunkSize`.

Callouts on the left side of the image point to specific parts of the code:

- Source string: points to `string! src;`
- Auxiliary counter: points to `int n;`
- Size: points to `invariant 0 < ChunkSize;`
- Return the next token: points to `public virtual string! NextChunk()`
- Implementation: points to the body of the `NextChunk()` method.
- Constructor: points to `public Chunker(string! source, int chunkSize)`

Annotations on the right side of the code editor:

- Chunker is a kind of StringTokenizer**: A blue text annotation above the class definition.
- Errors are caught on program construction, but libraries must be previously annotated.**: A blue text annotation next to the `NextChunk()` method.
- Specifications can also be wrong!**: A blue text annotation next to the constructor.

A yellow callout box highlights an error in the `NextChunk()` method body:

```

Call of string.Substring(int startIndex, int length), unsatisfied precondition:
requires startIndex + length <= this.Length;
    
```

The status bar at the bottom shows 'Ready', 'Ln 12', 'Col 20', 'Ch 20', and 'INS'.

Dynamic Program Analysis

- Performed by executing programs
 - In most cases, the executable code is used
- Used to infer properties
 - Property stability is related to the number of different program executions
- Dynamic Analysers
 - [Daikon](#)

Dynamic Program Analysis

- **Daikon** exercises a program using test cases and then provides heuristic inferred invariants

Adding multiple contacts to the Phonebook

Size of Phonebook here limited to 3.

```
=====
PBook.addListing(java.lang.String, java.lang.String)::ENTER
=====
PBook.addListing(java.lang.String, java.lang.String)::EXIT
this.tableDB == orig(this.tableDB)
this.str == orig(this.str)
this.contact == orig(this.contact)
this.currentSize one of { 1, 2, 3 }
return.toString one of { "Contact Added.", "Phone Book is full." }
name.toString == orig(name.toString)
phoneNumber.toString == orig(phoneNumber.toString)
=====
```

Dynamic Program Analysis

Methods used inside the Phonebook application

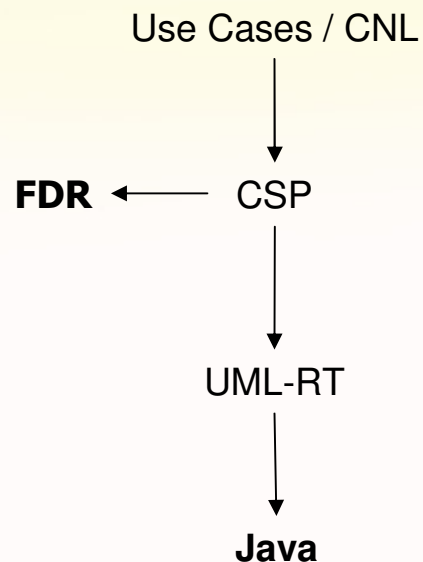
```

=====
PBook.isFull():::ENTER
=====
PBook.isFull():::EXIT
this.tableDB == orig(this.tableDB)
this.str == orig(this.str)
this.currentSize == orig(this.currentSize)
this.contact == orig(this.contact)
(return == true) <==> (PBook.DEFAULT_CAPACITY == this.currentSize)
(this.currentSize one of { 0, 1, 2 }) <==> (return == false)
=====
PBook.isFull():::EXIT;condition="return == true"
PBook.DEFAULT_CAPACITY == this.currentSize
PBook.DEFAULT_CAPACITY == orig(this.currentSize)
return == true
=====
PBook.isFull():::EXIT;condition="not(return == true)"
this.currentSize one of { 0, 1, 2 }
return == false
=====

```

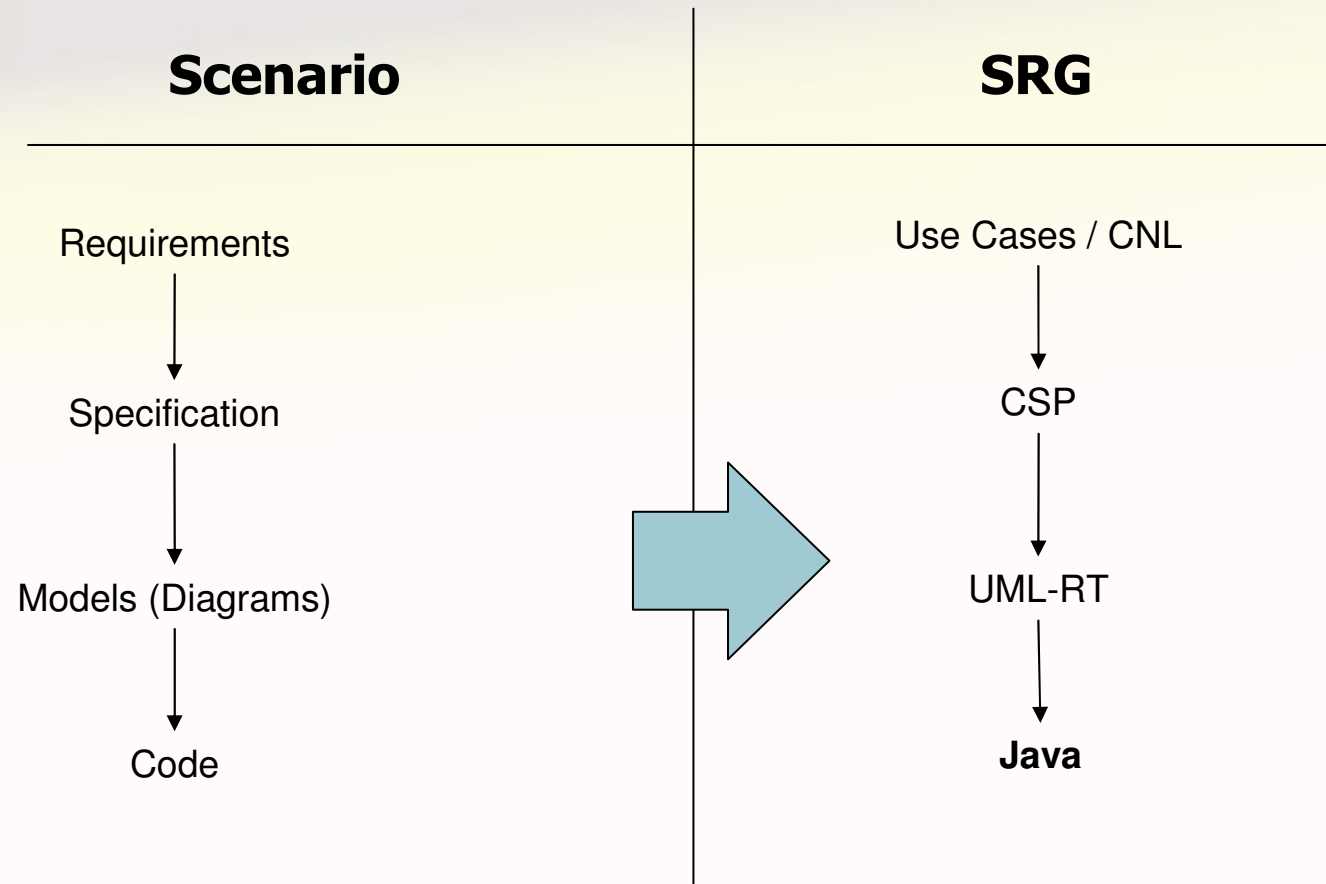
**Invariants can
be used to
assert logical
properties,
theorems, etc.**

Critical view of the scenario



- For specific properties, approaches are promising
- In practice, State explosion
- Scalability is still a problem
- Does not combine components

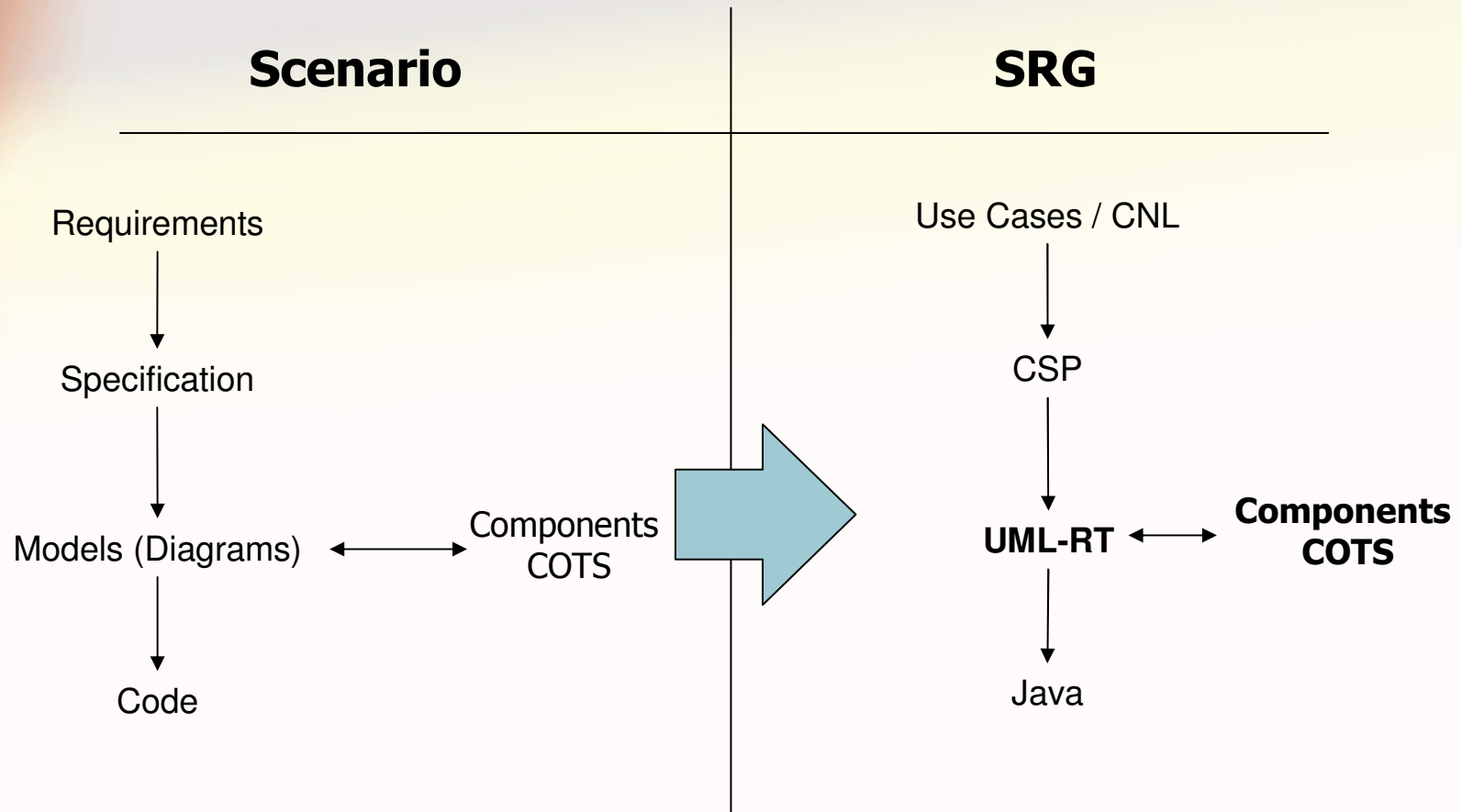
Introducing Non-Functional Requirements



Performance

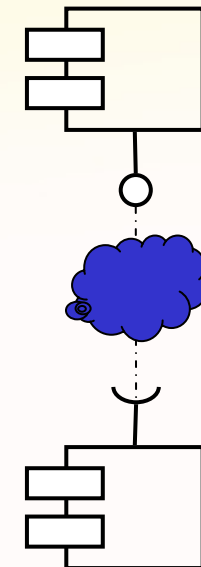
- Sequential programs do not take advantage of **multiprocessing** capabilities
- Introducing parallelism using transformation laws
 - Using **laws** increases reliability
- Resulting code executes independent commands in **different threads**
- **Performance** is directly related to the number of available processors

Considering Third Party Components



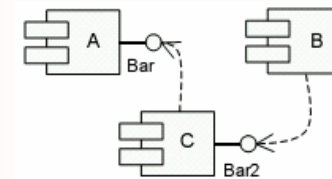
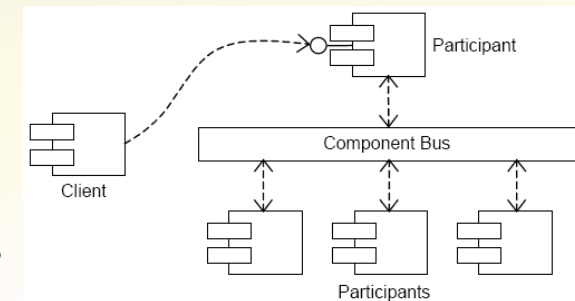
Integrated Elements

- Integrated elements deal with:
 - Heterogeneous
 - Orthogonal [**transparent**] services
 - Such as, transaction management, load balance, resource sharing, etc.
 - Common design criteria
 - **Quality Assurance**
 - Preservation of quality attributes (properties)

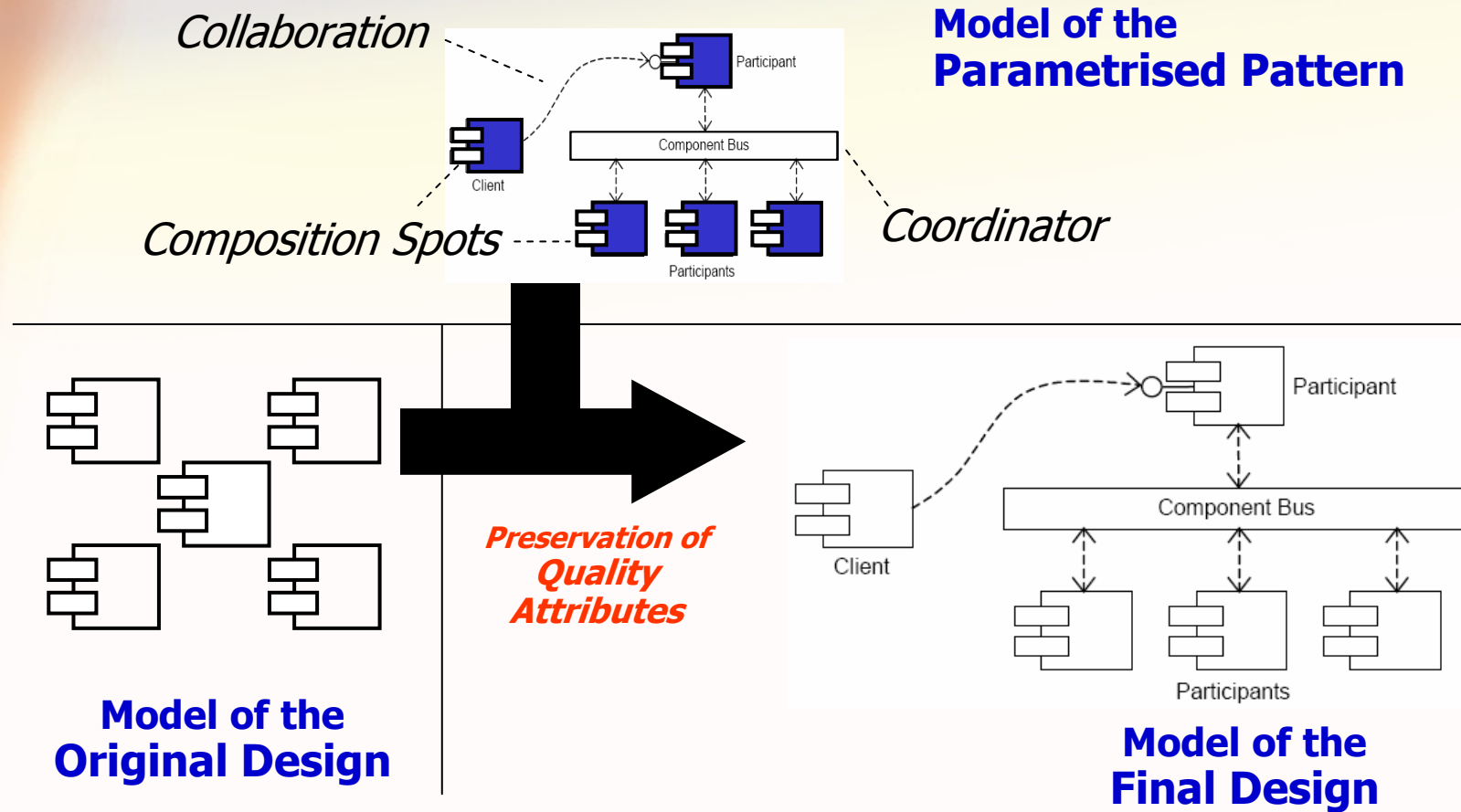


Integrated Elements Design

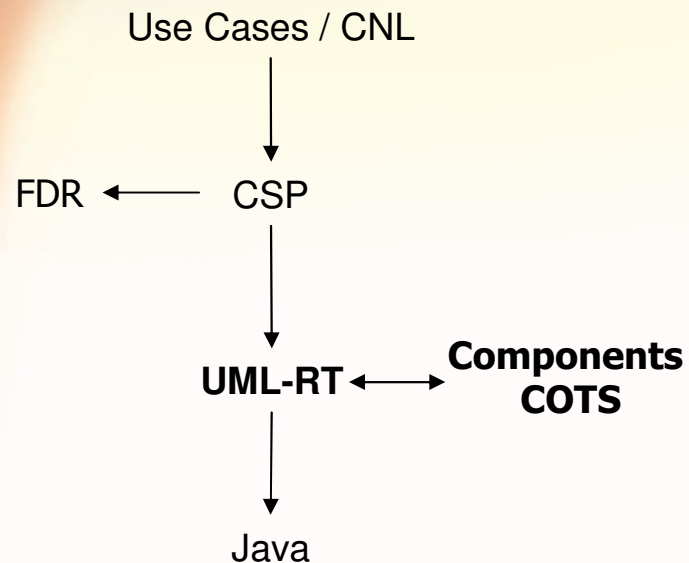
- Reused in different architectures
 - Ex.: Connectors Wrappers, coordinators, etc.
 - Parametrised by components contracts
 - Not fully specified until their final composition with components
- Design guided by domain specific patterns
 - Component interaction, fault tolerant, real-time, distributed systems



Component Integration Guided by design patterns

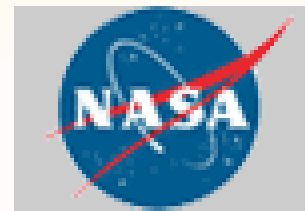


Critical view of the scenario



- Parallelism increases performance
- Integration increases productivity
- Properties depend on specific domains
- Reuse is difficult due to parameterisation

Formal Methods in Industry



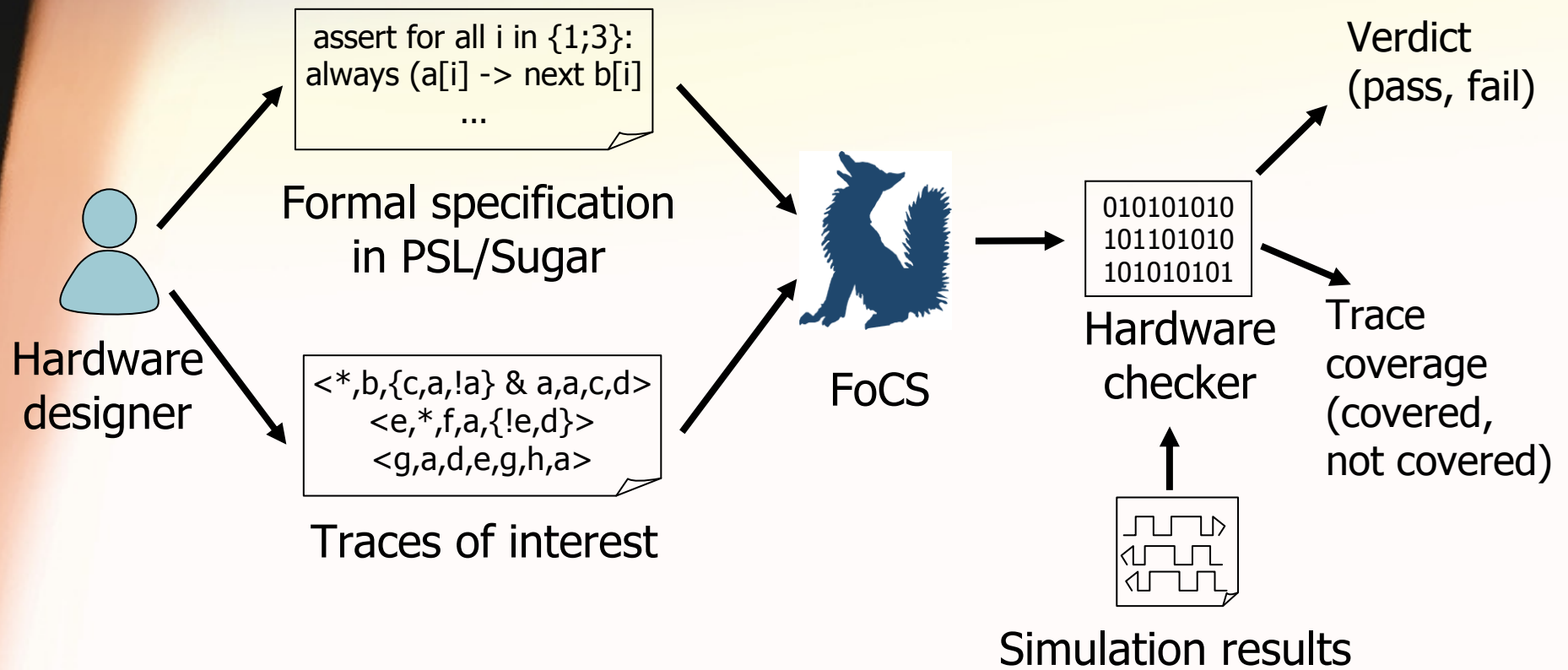
MOTOROLA



Microsoft

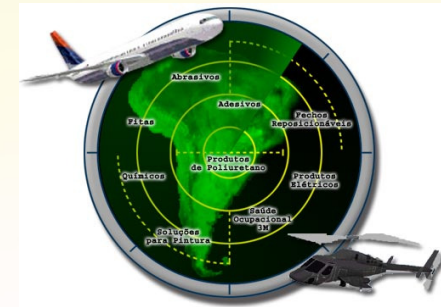
- Project Terminator
 - Automatically proving program termination
 - Used on windows device drivers (35 KLOC)
 - Current Research
 - Investigating concurrent program termination
- Project SLAM
 - Static analysis of C-based device drivers
 - Find errors with respect to API usage rules
 - Analyzed 126 drivers: 53 bugs found!
 - Average size 12 KLOC (largest: 130 KLOC)

IBM



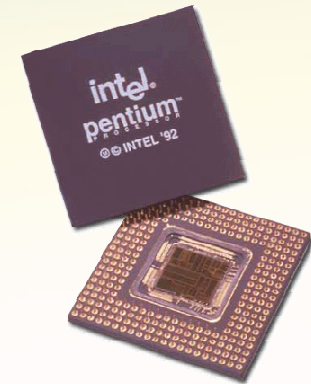
NASA

- Formal analysis of air traffic management systems
- Formal verification of safety-critical flight systems
 - Found a bug related to the starvation of a critical real-time flight calculation
- Research
 - Formalizing float-point and real number calculations, integral calculus, etc. in theorem provers



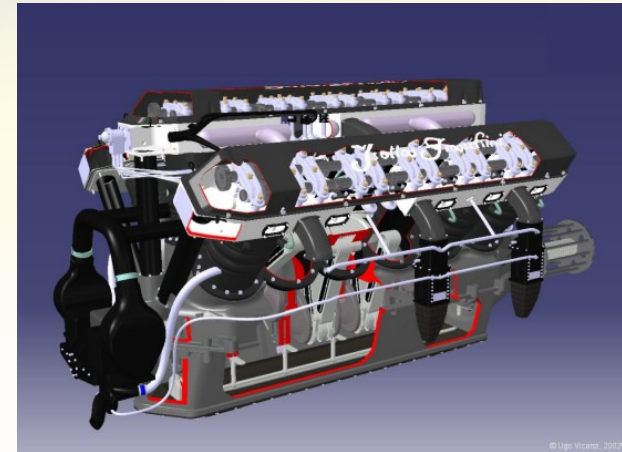
Intel

- (Mainly after the \$500 million Pentium bug in 1993...)
- Floating point verification
- Automation of elementary number-theoretic proofs
- Verification of nonlinear real formulas
- Functional language for hardware design and theorem proving



BMW Rolls-Royce

- Aero Engines
 - Formal methods were used to obtain the certification of the Electronic Control Engine
- The language Z was used to build a language interpreter



Bell Labs

- The formal methods group developed the following tools:
 - Delta-X: specification of integrity constraints and generation of constraint-checking code
 - Orion: static analysis of C and C++ code
 - VeriSoft: software model checking and systematic testing

Airbus

- SCADA Suite
 - Used to develop most of the A380 and A400M critical on-board software and for the secondary flying command system of the A340-500/600 aircraft, in operational use since August 2002
 - Significant decrease of coding errors. Up to 70% of the code has been automatically generated



Embraer

- The Pratt & Whitney PW617F engine is driven by SCADE-generated code
- Research goal: From model V to Y



Embraer Phenom 100

Motorola

- Precise natural language requirement documentation
- Automatic generation of
 - UML diagrams
 - Natural language tests
 - TAF tests
 - TAF implementations

Verified Software Grand Challenge

- Main goals are:
 - establish a unified theory of program construction and analysis;
 - build a comprehensive and integrated suite of tools that support verification activities, including specification, validation, test-case generation, program refinement, program analysis, program verification, and runtime checking; and
 - collect a repository of formal specifications and verified codes

Summary

- Ideal scenario is still unrealistic in practice
- But *ad hoc* testing is no longer an alternative
- Integration of formal approaches to V&V with systematic testing seems promising
- Industrial applications provide practical evidence
- Non-functional requirements addressed with heuristics