Correctness-by-Construction Engineering – How can we build better software?

Ina Schaefer

31 May 2023, ACM Tech Talk
Software Failures are expensive…

• Ariane 5 in July 1996 – Self Destruction due to overflow…
• Reason: Erroneous Reuse of Ariane 4 Software
• Damage: 370 Million US$
… Software Failures are also deadly.

Lufthansa-Flight 2904 (A320) from Frankfurt Main to Warsaw crashed on landing at WAW on 14 September 1993.

Damage:
- 2 dead
- 62 injured

Mariusz Siecinski - http://www.airliners.net/photo/Lufthansa/Airbus-A320-211/0265541/L/
Ensuring software correctness

- Small fraction of SW is correctness-critical, but then it really matters.
- Usually correctness is ensured by post-hoc testing (and sometimes post-hoc verification).
- But that is usually expensive.
- And: Software can’t be tested into correctness.

“We search for bugs…”

“Program testing can be used to show the presence of bugs, but never to show their absence!”

Edsger Dijkstra
Post-hoc Verification vs. Correct-by-Construction

Post-hoc Verification:
- Implementation
- Specification
- Verification

Correctness-by-Construction (CbC):
- Specification
- Refinement Step
- Check Side Conditions
- Correct Implementation

Specification Refinement Step Check Side Conditions Correct Implementation
Why CbC?

- Think first rather than hacking things into correctness
- Errors detected earlier
- Reduce proof complexity

CbC in other Engineering Disciplines

- Correctness-by-Construction is very common, e.g., in mechanical engineering, civil engineering, electrical engineering, chip design etc.

- No sense in building a full thing and start testing it…

- For example, **CAD tools**:
  - Component-based engineering from components with known properties
  - Standard libraries of building blocks used by drag-and-drop
  - Tools respect component properties and restrictions on composition

Example: FreeCAD
Industrial Uptake – Success Stories

- **Case: Large semi-conductor manufacturing company**
  - Mix of software for robotics (motion planning), metrology, mechatronics, optimization for semi-conductor manufacturing machines
  - High robustness, sensitivity, performance requirements
  - Physical and financial consequences to a software crash
  - Tens of millions of lines of C and C++ code
  - 100% faithful simulation software for optimization purposes
  - Simulation code’s kernel developed using CbC

- **Comparing CbC against traditional SE process:**
  - Factor 10 reduction in time to test-suite validation
  - Factor 3 reduction in size of team required
  - Cleaner and more maintainable code
  - Clearly requires some CbC-skills on-team
Correctness-by-Construction (CbC)

Edsger W. Dijkstra

"For a long time I have wanted to write a book somewhat along the lines of this one: on the one hand I knew that programs could have a compelling and deep logical beauty, on the other hand I was forced to admit that most programs are presented in a way fit for mechanical execution but, even if of any beauty at all, totally unfit for human appreciation."
CbC – General approach

1. Start with $\{P\} S \{Q\}$ where $S$ is an abstraction of code

2. Refine iteratively using refinement rules (and common sense)

3. Certain rules replace parts of abstract code with GCL commands

4. Each rule preserves correctness of $\{P\} S \{Q\}$

5. Eventually arrive at GCL code, $C$, such that $\{P\} C \{Q\}$ is true
Linear search

Determine which index $i$ in array $A$ contains $x$

Assumptions:

- At least one $i$ such that $A_i = x$
- No preferred $i$ if $x$ occurs multiply in $A$. 
Linear search – more formally

Formally:

- **Pre-condition:** \( \text{app}(A, x, k, \ell) \triangleq \exists i \in [k, \ell) : (A_i = x) \)

- **Post-condition:** \( A_i = x \)

- **Problem:** \( \{\text{app}(A, x, 0, A.\text{len})\} i : S \ \{(A_i = x)\} \)
Linear search – choosing (in)variant

Variant: \( i \)

\[
\text{inv: } = \neg \text{app}(A, x, i + 1, A.\text{len})
\]

\( A \)

\[
\begin{array}{c}
x \text{ must be here} \\
0 \\
\end{array} \quad \begin{array}{c}
\downarrow \\
i + 1 \\
\end{array} \quad \begin{array}{c}
x \text{ not here} \\
A.\text{len} \\
\end{array}
\]

- Strategic choice: sweep from right to left
- Suggests invariant and variant indicated above
Linear search – initialization

\{\text{app}(A, x, 0, A.\text{len})\} \ i : S \ \{(A_i = x)\}

\sqsubseteq \ \{\text{Strengthen postcon: inv} \land (A_i = x) \ \Rightarrow (A_i = x)\}

\text{app}(A, x, 0, A.\text{len}) \ i : S \ \{\text{inv} \land (A_i = x)\}

\sqsubseteq \ \{\text{Composition: Mid predicate is inv}\}

\text{app}(A, x, 0, A.\text{len}) \ i : \text{init}; \{\text{inv}\} S_2 \ \{\text{inv} \land (A_i = x)\}

\sqsubseteq \ \{\text{Assignment: app}(A, x, 0, A.\text{len}) \ \Rightarrow \neg\text{app}(A, x, i + 1, A.\text{len})[i \backslash A.\text{len} - 1]\}

\text{app}(A, x, 0, A.\text{len}) \ i : i := A.\text{len} - 1; \{\text{inv}\} S_2 \ \{\text{inv} \land (A_i = x)\}
Linear search – loop structure

\{\text{inv}\} S_2 \{\text{inv} \land (A_i = x)\}
\subseteq \{\text{Repetition: } \neg G \text{ is } (A_i = x) \} \{\text{inv}\}
\{\text{inv}\} \textbf{do } (A_i \neq x) \rightarrow B \textbf{ od } \{\text{inv} \land (A_i = x)\}
\subseteq \{\text{Insert implicit pre- and post con for } B \} \{\text{inv}\}
\textbf{do } (A_i \neq x) \rightarrow \\
\quad \{(A_i \neq x) \land \text{inv}\} \ B \{\text{inv}\}
\textbf{ od}
\{\text{inv} \land (A_i = x)\}
Linear search – loop body

\[
\{(A_i \neq x) \land \text{inv}\} B \{\text{inv}\}
\]

\[\sqsubset \{\text{Variant bounds strengthen postcon: inv} \land (0 \leq i < i_0)\}\]

\[
\{(A_i \neq x) \land \text{inv}\} B \{\text{inv} \land (0 \leq i < i_0)\}
\]

\[\sqsubset \{\text{Assignment: inv} \land (A_i \neq x) \Rightarrow (\text{inv} \land (0 \leq i < i_0))[i \setminus i - 1]\}\]

\[
\{\text{inv} \land (A_i \neq x)\} \quad i := i - 1 \quad \{\text{inv} \land (0 \leq i < i_0)\}
\]
Basic CbC-Refinement Rules

\[ \{P\} S \{Q\} \text{ can be refined to } \]

**Skip:** \( \{P\} \text{ skip } \{Q\} \iff P \implies Q \) \hspace{1cm} (1)

**Assignment:** \( \{P\} x := E \{Q\} \iff P \implies Q[x := E] \) \hspace{1cm} (2)

**Composition:** \( \{P\} S1 ; S2 \{Q\} \iff \text{ there is } M \text{ s.t. } \{P\} S1 \{M\} \text{ and } \{M\} S2 \{Q\} \) \hspace{1cm} (3)

**Selection:** \( \{P\} \text{ if } G \text{ then } S1 \text{ else } S2 \{Q\} \text{ fi } \iff \)
\[ \{P \land G\} S1 \{Q\} \text{ and } \{P \land \neg G\} S2 \{Q\} \]
\hspace{1cm} (4)

**Repetition:** \( \{P\} \text{ do } G \rightarrow S \text{ od } \{Q\} \iff \text{ there is invariant } I \text{ and variant } V \text{ s.t.} \)
\[ (P \implies I) \text{ and } (I \land \neg G \implies Q) \text{ and } \{I \land G\} S \{I\} \]
\[ \text{ and } \{I \land G \land V = V_0\} S \{I \land 0 \leq V < V_0\} \]
CorC – Tool Support for CbC

- IDE for Correctness-by-Construction
- Textual and graphical editor
  - Meta-model with EMF
  - Interchangeable
- KeY used to verify the refinements
- Available at https://github.com/KIT-TVA/CorC

WebCorC (https://www.isf.cs.tu-bs.de/WebCorC/)
CbC – Benefits and Applicability

- CbC generally results in well-structured code.
- CbC allows the taxonomization of algorithmic families.
- CbC saves testing effort and can be used to bootstrap post-hoc verification.
- CbC allows faster time to market.

- But: Tool Support is needed!!
The CorC Ecosystem

- CorC 2.0 – OO-Programs-by-Construction
- ArchiCorC – Scalable CbC for Architectures
- VarCorC – CbC-Construction of SPLs
- IFbC - Information-Flow-By-Construction

CorC 2.0 – The Idea

Single algorithms

OOP

CorC

Java
CorC 2.0

**Classes**: Fields, Methods, and Class Invariants
Inheritance and Interfaces

**Synchronisation** between Java Classes and CbC-Diagrams

**Change Tracking**

---

**CorC-Project**

**ClassA.java**
- `ClassA`
- `ClassA.cbcclass`
- `MethodA1.cbc`
- `MethodA2.cbc`

**ClassB.java**
- `ClassB`
- `ClassB.cbcclass`
- `MethodB1.cbc`

---

**Change Tracking**

**Import and Export**

**Synchronisation**

**Change Tracking**

---

**Classes**

- `public int a`
- `Class invariants
  a >= 0`

**Pre**: `<prenew>`
**Post**: `<postnew>`

---

**Public static void methodB1()**
CorC 2.0 - Roundtrip Engineering

- Single methods or whole Java classes
- Construction in CorC
- Application of CbC refinement rules
- Java to CorC
- CorC to Java
- Generation of correct Java code to new or original class
Component-based Software Engineering

*Decompose large software into coherent and modular sub-units!*

- **Atomic Components**
  - Independent module
  - Mapping to implementation artifacts (i.e., behavioral model)

- **Composite Components**
  - Aggregation of components

- **Interfaces**
  - Interaction between components
  - Provided/Required Interfaces
ArchicorC – Scaling CbC

Contract-based Interface Description

```java
interface OrderEntry {
    /\[...\]
    method add {
        contract {
            requires: "Person.isSolvent(d)";
            ensures: "OrderAbleItem.isAvailable(id) => res";
        }
        in { UInt32 id } 
        out { Boolean res }
    }
    /\[...\]
}
```

CorC Implementation

1. Contract-based Interface Description
2. Composition for \( F \)
3. Side conditions of refinement rules
4. Code Generation

Compatibility and consistency checking

Correct-by-Construction

Correct-by-Construction Java Code

Feature-Oriented Programming

Produkte:
[BankAccount]
[BankAccount, DailyLimit]
[BankAccount, Interest]
[BankAccount, Transaction]
[BankAccount, DailyLimit, Interest]
[BankAccount, DailyLimit, Transaction]
[BankAccount, Interest, Transaction]
[BankAccount, DailyLimit, Interest, Transaction]
**VarCorC: SPL-by-Construction**

### Feature model

- **Root**
  - **Base**
  - **Limited**
  - **Sorted**
    - **Increasing**
    - **Decreasing**

### Feature Composition Order:

- **Base**
- **Limited**
- **Sorted**
- **Increasing**
- **Decreasing**

### Legend:

- **Mandatory**
- **Optional**
- **Alternative**
- **Abstract**
- **Concrete**

---

### Verification

- **Feature-based CbC**
- **Product-based**
- **Family-based**

---

Secure and Correct Program Construction

Motivating Example:

- **Client** and **server** have different security levels
- **Client** should not read **server** data uncontrolled
  - Ensure confidentiality
- **Server** should not read **client** data uncontrolled
  - Ensure integrity
Information Flow Control

- **Non-interference**: Confidential data may not be deducible from public data
- **Direct Information Flow**: Assignments
- **Indirect Information Flow**: Usage of confidential data in loop-guards and if-conditions

**Lattice-based Information Flow Policies**

![Lattice Diagram]

Cosmic Top Secret
Secret
Confidential
Restricted
Unclassified
Untainted
Tainted
Language-based Security (Post-hoc)

```java
boolean low paymentAction := true;
if (paymentAction) {
    int high creditCard := getNumber();
    while (!valid(creditCard)) {
        creditCard = getNumber();
    }
    String low output := declassify(mask(creditCard));
} else {
    skip
}
```

Security Type Systems [cf. Sabelfeld, Myers, Volpano et al.]

1. \( \vdash \text{exp} : \text{high} \)
2. \( \vdash \text{h} \not\in \text{Vars(exp)} \)
3. \( \vdash \text{exp} : \text{low} \)
4. \( \vdash \text{skip} \)
5. \( \vdash \text{h} = \exp \)
6. \( \vdash \text{exp} : \text{low} \)
7. \( \vdash \text{ct} \vdash S_1 \) \( \vdash \text{ct} \vdash S_2 \)
8. \( \vdash \text{ct} \vdash \text{while exp do S} \)
9. \( \vdash \text{ct} \vdash \text{if exp then S_1 else S_2} \)

- Fig. 1. Refinement Rules in CbC [14]
- Fig. 2. Security Type System [19]

To discard programs which violate confidentiality as expressed by a security policy, a security type system can be introduced (cf. Figure 2) according to [19]. The type system assigns every variable and expression a security type. `E : t` means that expression `E` has security type `t`; in our case `t` can be either `high` or `low`. The type system uses a security context which is an environment variable tracking the current status of the program (high or low) to control implicit information flow. In a high context, no assignments to low variables may occur. The typing rules are depicted in Figure 2. The rules define that an expression `exp` can always have a high type (1), but can only have a low type if no high variables occur in the expression (2). A skip is always typeable (3), and every expression can be assigned to a high variable (4). If we want to assign an expression to a low variable, the expression must be low (5). A composition of two statements keeps the same context (6). Rules (7) and (8) are used to ensure that if the guard has a high context, the statements are typable in a high context.

Security Type Systems 

31.05.2023

32
IFbC – Information Flow-by-Construction

```java
static low imm void verifySignature(
    low mut Client client, low mut Email email) {
    low imm int pubkey = client.publicKey;
    high imm int privkey = email.emailSignKey;
    high imm boolean isVerified;
    if (isKeyPairValid(privkey, pubkey)) {
        isVerified = true;
    } else {
        isVerified = false;
    }
    email.IsSignatureVerified = isVerified;
}
```
The CorC Ecosystem

- Original CorC – Functionally Correct Procedural Programs
- CorC 2.0 – OO-Programs-by-Construction
- ArchiCorC – Scalable CbC for Architectures
- VarCorC – CbC-Construction of SPLs
- IFbC - Information-Flow-By-Construction
- IFbC-OO - Information-Flow-By-Construction for OO-Languages

Future Research Directions

Applying CbC to other Domains and Languages
• CbC for Parallel and Cloud-based Programming
• CbC for Data-flow Languages, such as Matlab/Simulink
• CbC for Domain-Specific Languages
• CbC for Quantum Computing

X-by-Construction: CbC for Other Types of Properties
• Information Flow Security
• Resource Usage: Memory, Energy, …

Prerequisite is CbC-Readiness:
• Formal Language Syntax
• Formal Specification Language
• Proof Rules for Specifications over Programs
• Refinement Rules for Programs and Specifications
Conclusions

- CbC should be taught more widely. (Against the bad image of Formal Methods!)
- CbC *lite* should be widely used, even in the presence of verification tools, or if verification is mandated by standards.
- Conceptual contributions to CbC for Other Domains and Properties are necessary.
- Tool Development for Broader Uptake of CbC in Large Scale Software Development
Research Portfolio

Automotive Software Engineering

Variability & Evolution

Quality Assurance & Testing
Re-Engineering & Analysis
Correctness-by-Construction
Quantum Software Engineering