Karlsruher Institut für Technologie

Forschungsgruppe für
Test, Validierung und Analyse
Software-intensiver Systeme

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

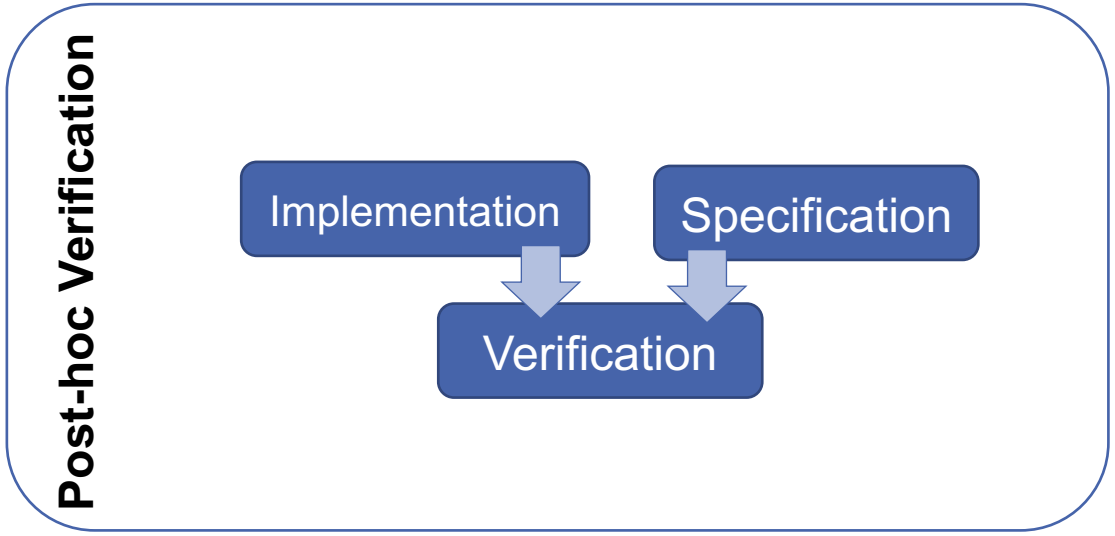
- Small fraction of SW is correctness-critical, but then it really matters.
- Usually correctness is ensured by post-hoc testing (and sometimes post-hoc verification).
- But that is usually expensive.
- And: Software can’t be tested into correctness.

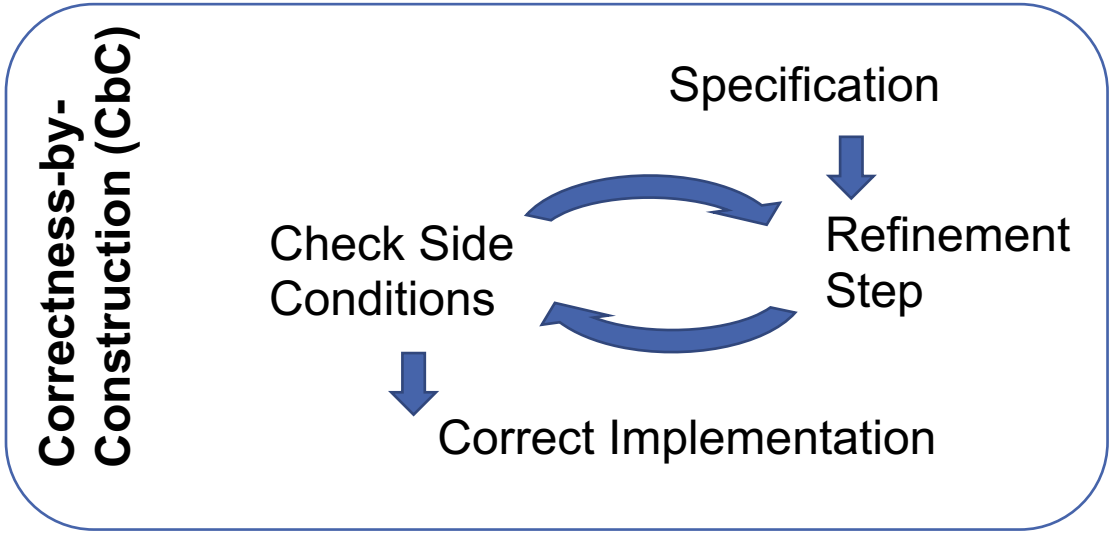
„Program testing can be used to show the presence of bugs, but never to show their absence!“

Edsger Dijkstra

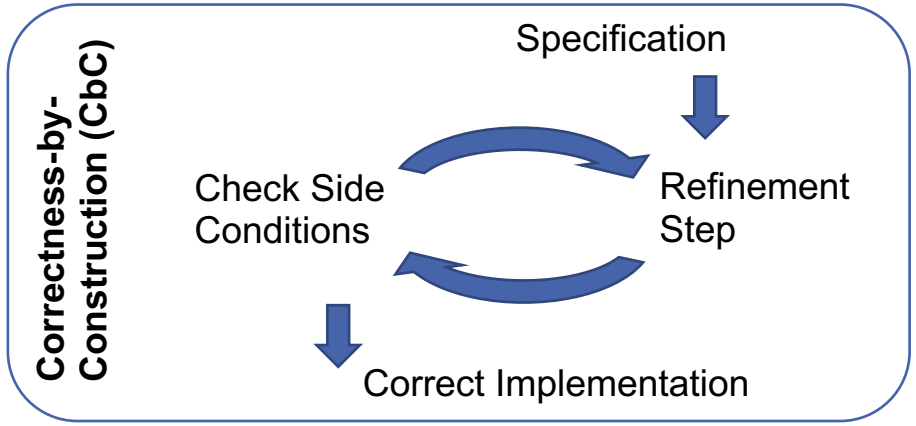
“We search for bugs…”

Post-hoc Verification vs. Correct-by-Construction

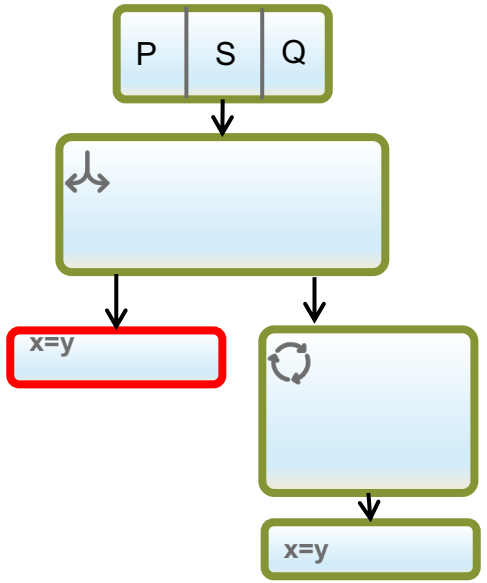


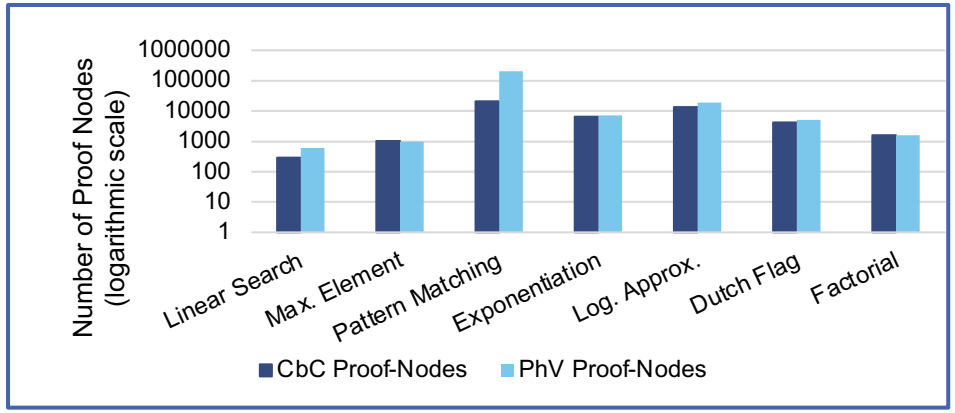


Why CbC?



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

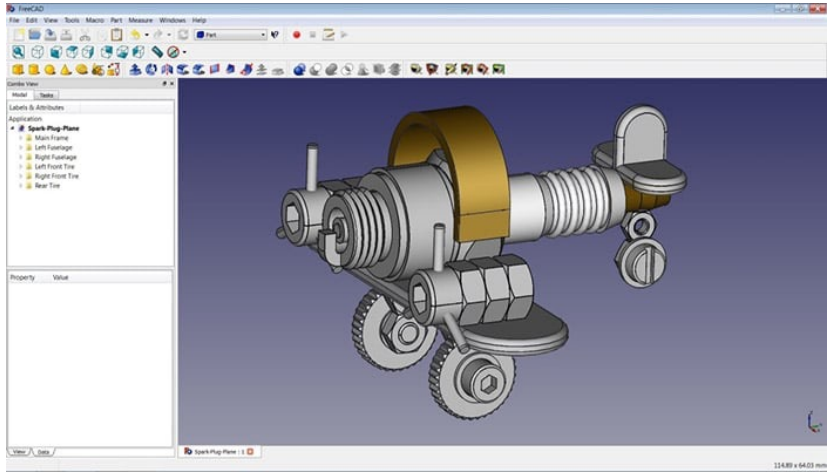




T. Runge, I. Schaefer, L. Cleophas, T. Thüm, D. G. Kourie, B. W. Watson: Tool Support for Correctness-by-Construction, Proc. of the International Conference on Fundamental Approaches to Software Engineering (FASE), Springer, 2019.

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

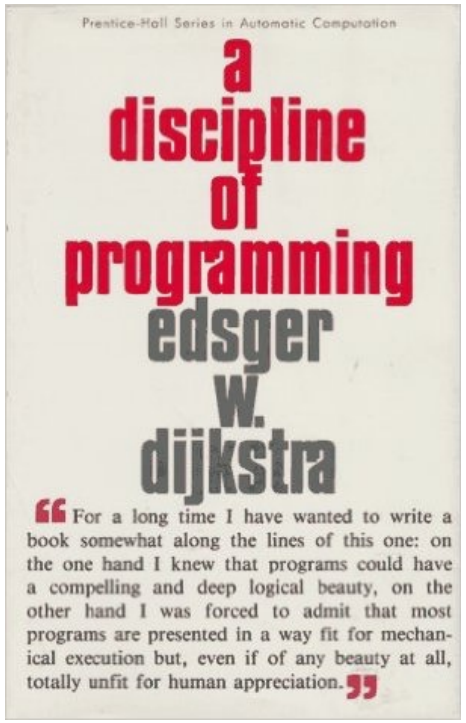
Industrial Uptake – Success Stories

- **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

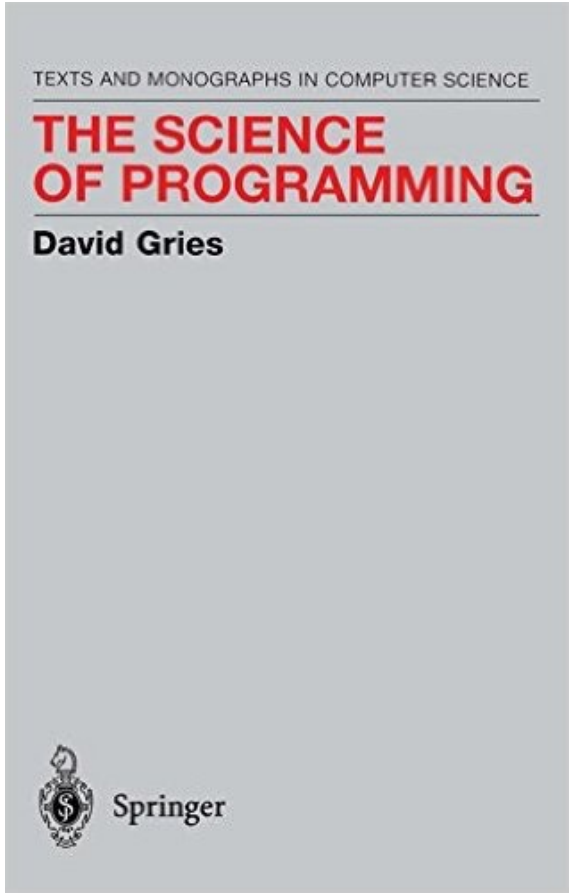
- **Comparing CbC against traditional SE process:**
  - Factor 10 reduction in time to test-suite validation
  - Factor 3 reduction in size of team required
  - Cleaner and more maintainable code
  - Clearly requires some CbC-skills on-team

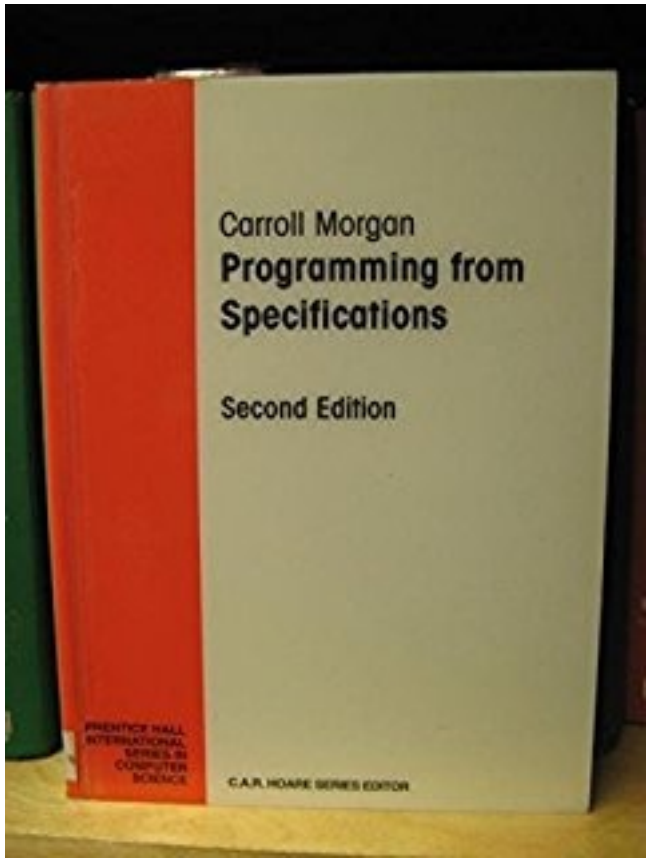
PPT Stock Photo

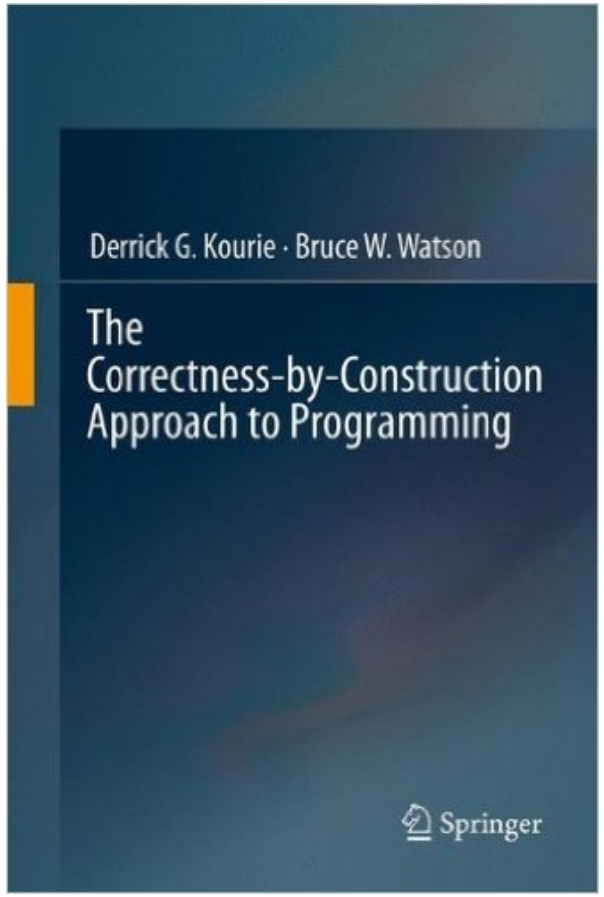
Correctness-by-Construction (CbC)



CbC Round 2+

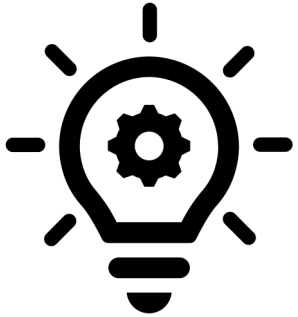

TEXTS AND MONOGRAPHS IN COMPUTER SCIENCE
THE SCIENCE OF PROGRAMMING
David Gries
Springer


Carroll Morgan
Programming from Specifications
Second Edition


Derrick G. Kourie · Bruce W. Watson
The Correctness-by-Construction Approach to Programming
Springer

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:

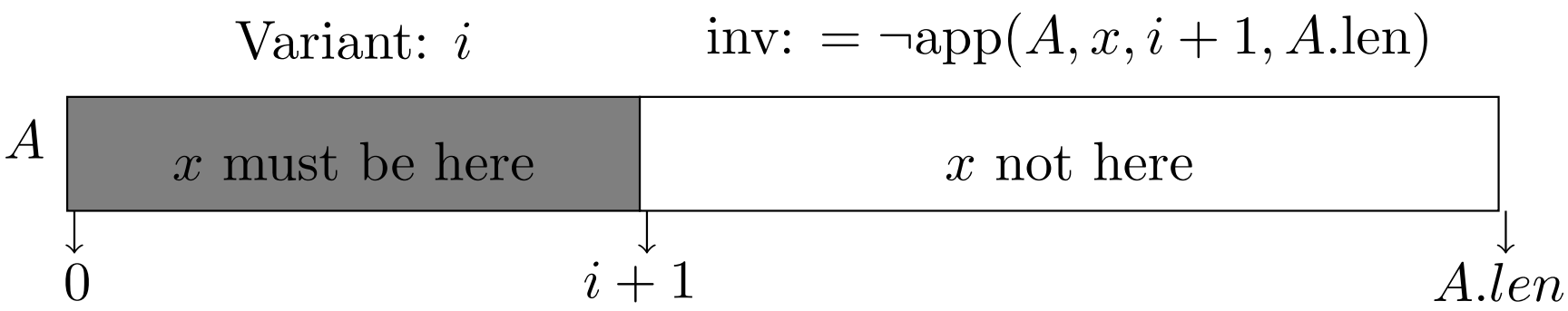
- At least one i such that $A_i = x$
- No preferred i if x occurs multiply in A.

Linear search – more formally

Formally:

- **Pre-condition:** $\mathrm{app}(A, x, k, \ell) \triangleq \exists i \in [k, \ell) : (A_i = x)$
- **Post-condition:** $A_i = x$
- **Problem:** $\{\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : S\ \{(A_i = x)\}$

Linear search – choosing (in)variant



- Strategic choice: sweep from right to left
- Suggests invariant and variant indicated above

Linear search – initialization

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

$$\sqsubseteq \quad \{\text{Strengthen postcon: } \mathrm{inv} \wedge (A_i = x) \Rrightarrow (A_i = x)\}$$

$$\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : S\ \{\mathrm{inv} \wedge (A_i = x)\}$$

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

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

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

$$\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : i := A.\mathrm{len} - 1; \{\mathrm{inv}\}\ S_2\ \{\mathrm{inv} \wedge (A_i = x)\}$$

Linear search – loop structure

$$\{\text{inv}\}\ S_2\ \{\text{inv} \wedge (A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Repetition: } \neg G \text{ is } (A_i = x)\ \}$$

$$\{\text{inv}\}\ \textbf{do}\ (A_i \neq x) \rightarrow B\ \textbf{od}\ \{\text{inv} \wedge (A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Insert implicit pre- and post con for } B\ \}$$

$$\{\text{inv}\}$$

$$\textbf{do}\ (A_i \neq x) \rightarrow$$

$$\{(A_i \neq x) \wedge inv\}\ B\ \{\text{inv}\}$$

$$\textbf{od}$$

$$\{\text{inv} \wedge (A_i = x)\}$$

Linear search – loop body

$$\{(A_i \neq x) \wedge \text{inv}\}\ B\ \{\text{inv}\}$$

$$\sqsubseteq \quad \{\text{Variant bounds strengthen postcon: } \text{inv} \wedge (0 \leq i < i_0)\}$$

$$\{(A_i \neq x) \wedge \text{inv}\}\ B\ \{\text{inv} \wedge (0 \leq i < i_0)\}$$

$$\sqsubseteq \quad \{\text{Assignment: } \text{inv} \wedge (A_i \neq x) \Rrightarrow (\text{inv} \wedge (0 \leq i < i_0)[i \backslash i-1]\ \}$$

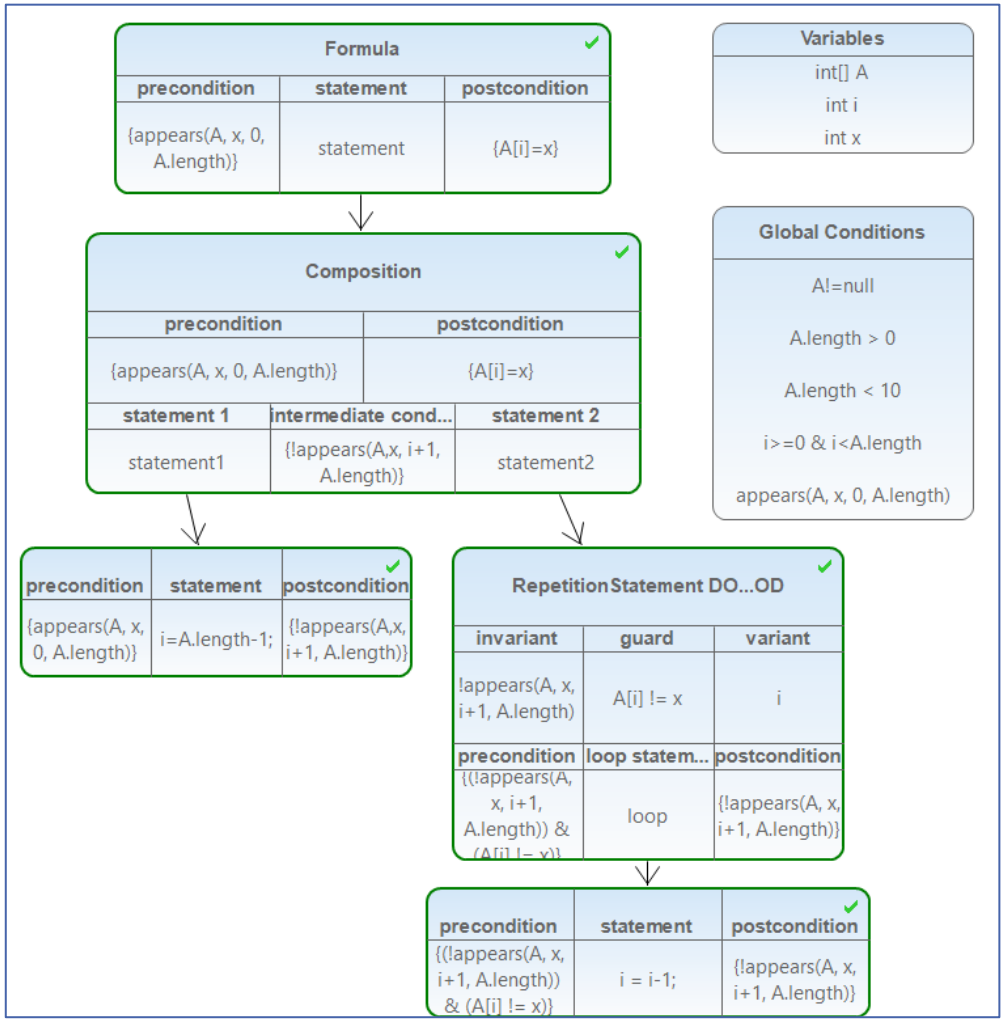
$$\{\text{inv} \wedge (A_i \neq x))\}\ i := i - 1\ \{\text{inv} \wedge (0 \leq i < i_0)\}$$

Basic CbC-Refinement Rules

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$	*can be refined to*	
Skip :	$\{\mathtt{P}\}$ *skip* $\{\mathtt{Q}\}$ *iff* $\mathtt{P}$ *implies* $\mathtt{Q}$	(1)
Assignment :	$\{\mathtt{P}\}\ x := E\ \{\mathtt{Q}\}$ *iff* $\mathtt{P}$ *implies* $\mathtt{Q}[\mathtt{x} := \mathtt{E}]$	(2)
Composition :	$\{\mathtt{P}\}\ \mathtt{S1}\,;\,\mathtt{S2}\ \{\mathtt{Q}\}$ *iff there is* $\mathtt{M}$ *s.t.* $\{\mathtt{P}\}\ \mathtt{S1}\ \{\mathtt{M}\}$ *and* $\{\mathtt{M}\}\ \mathtt{S2}\ \{\mathtt{Q}\}$	(3)
Selection :	$\{\mathtt{P}\}\ \mathbf{if}\ \mathtt{G}\ \mathbf{then}\ \mathtt{S1}\ \mathtt{else}\ \mathtt{S2}\ \{\mathtt{Q}\}\ \mathbf{fi}$ *iff*	(4)
	$\{\mathtt{P} \wedge \mathtt{G}\}\ \mathtt{S1}\ \{\mathtt{Q}\}$ *and* $\{\mathtt{P} \wedge \neg\mathtt{G}\}\ \mathtt{S2}\ \{\mathtt{Q}\}$	
Repetition :	$\{\mathtt{P}\}\ \mathbf{do}\ \mathtt{G} \rightarrow\ \mathtt{S}\ \mathbf{od}\ \{\mathtt{Q}\}$ *iff there is invariant* $\mathtt{I}$ *and variant* $\mathtt{V}$ *s.t.*	(5)
	$(\mathtt{P}$ *implies* $\mathtt{I})$ *and* $(\mathtt{I} \wedge \neg\mathtt{G}$ *implies* $\mathtt{Q})$ *and* $\{\mathtt{I} \wedge \mathtt{G}\}\ \mathtt{S}\ \{\mathtt{I}\}$	
	and $\{\mathtt{I} \wedge \mathtt{G} \wedge \mathtt{V} = \mathtt{V}_0\}\ \mathtt{S}\ \{\mathtt{I} \wedge 0 \leq \mathtt{V} < \mathtt{V}_0\}$	

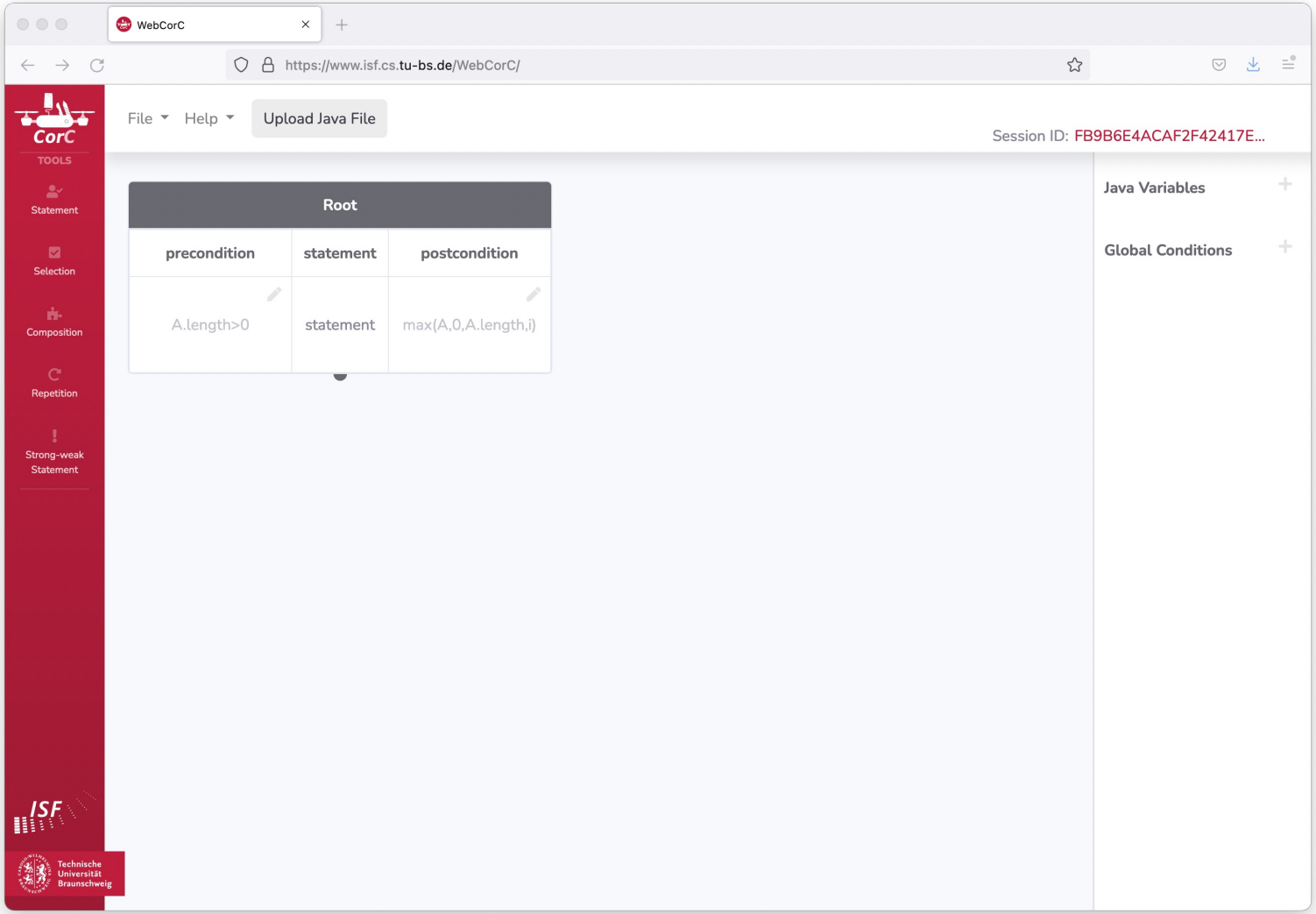
CorC – Tool Support for CbC

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



T. Runge, I. Schaefer, L. Cleophas, T. Thüm, D. G. Kourie, B. W. Watson: Tool Support for Correctness-by-Construction, Proc. of the International Conference on Fundamental Approaches to Software Engineering (FASE), Springer, 2019.

WebCorC (https://www.isf.cs.tu-bs.de/WebCorC/)

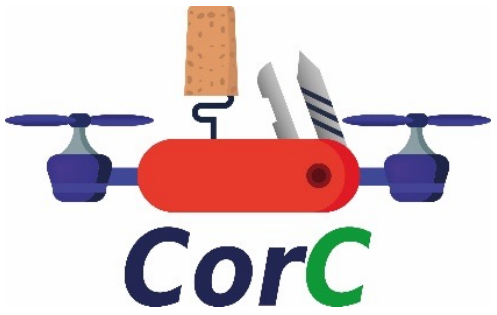


CbC – Benefits and Applicability

- 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!!**

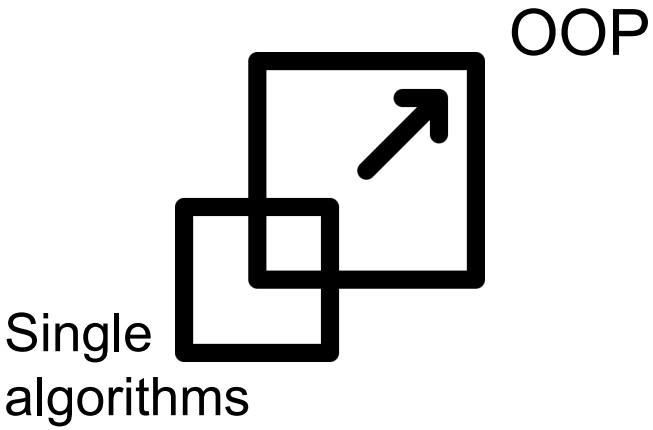
The CorC Ecosystem

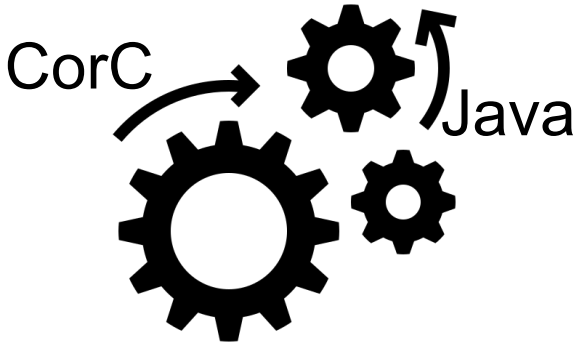
- CorC 2.0 – OO-Programs-by-Construction
- ArchiCorC – Scalable CbC for Architectures
- VarCorC – CbC-Construction of SPLs
- IFbC - Information-Flow-By-Construction



T. Bordis, L. Cleophas, A. Kittelmann, T. Runge, I. Schaefer, B.W. Watson: Re-CorC-ing KeY: Correct-by-Construction Software Development Based on KeY. The Logic of Software. A Tasting Menu of Formal Methods: 80-104, 2022.

CorC 2.0 – The Idea



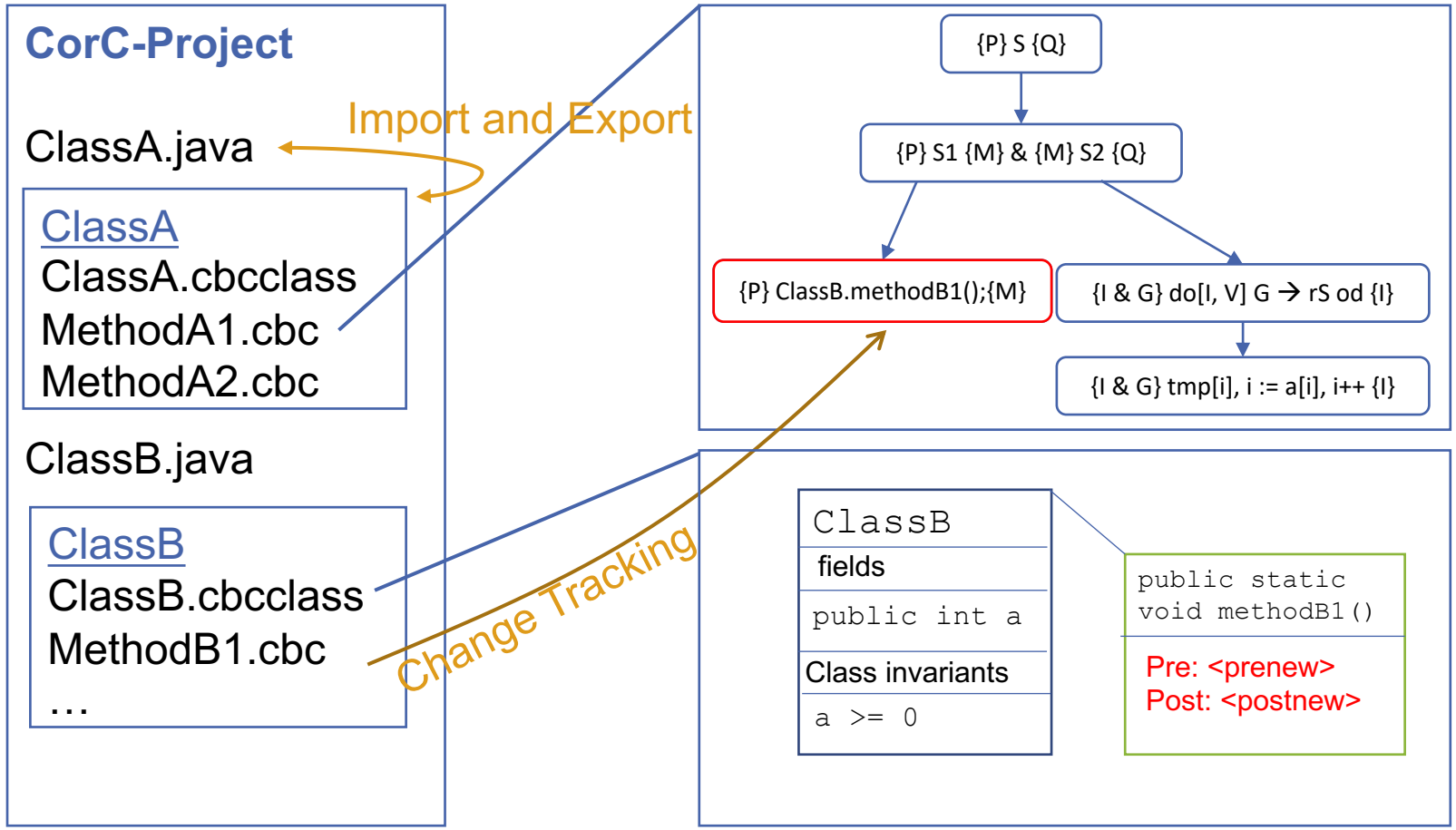


CorC 2.0

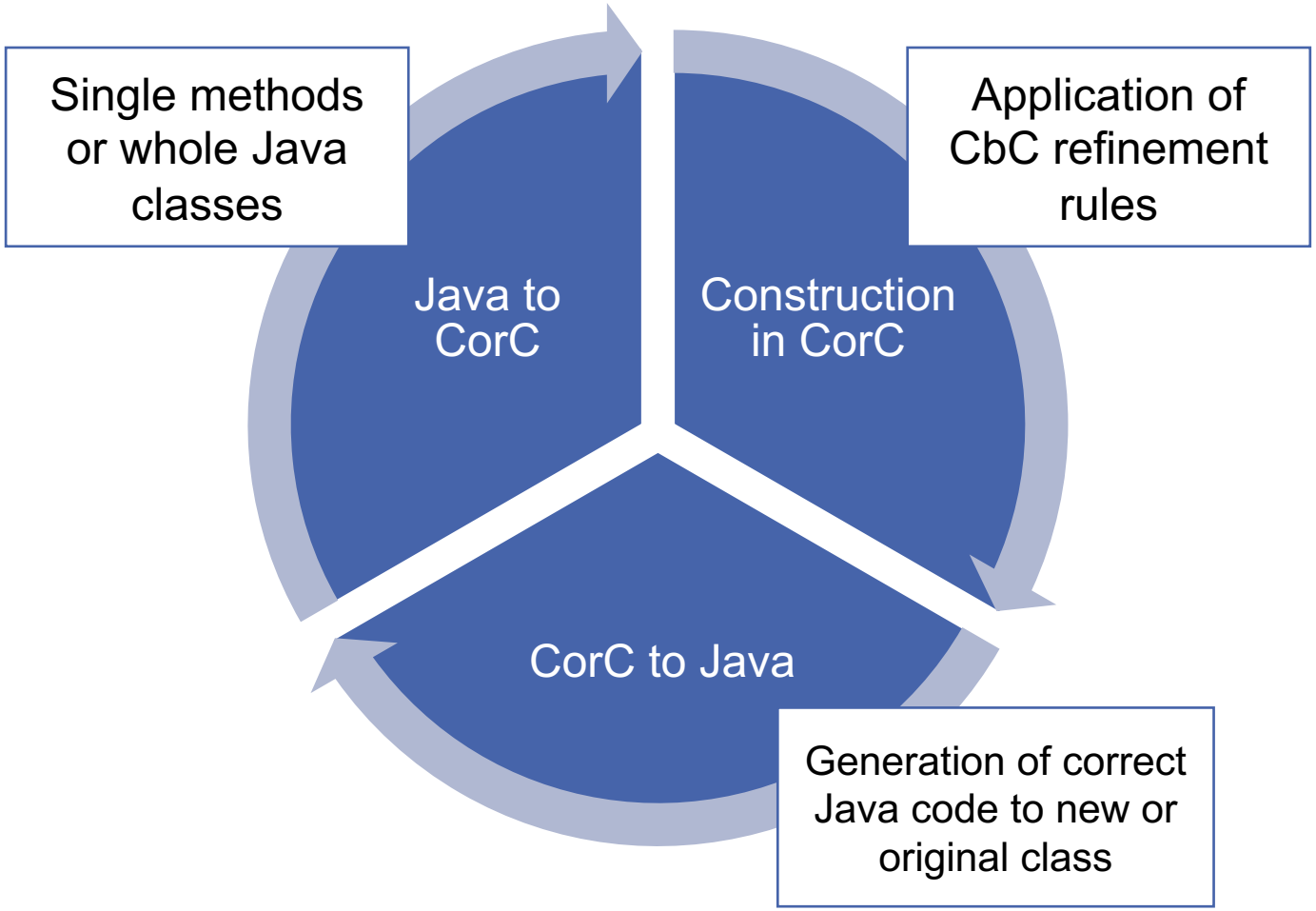
Classes: Fields, Methods, and Class Invariants
Inheritance and Interfaces

Synchronisation between Java Classes and CbC-Diagrams

Change Tracking



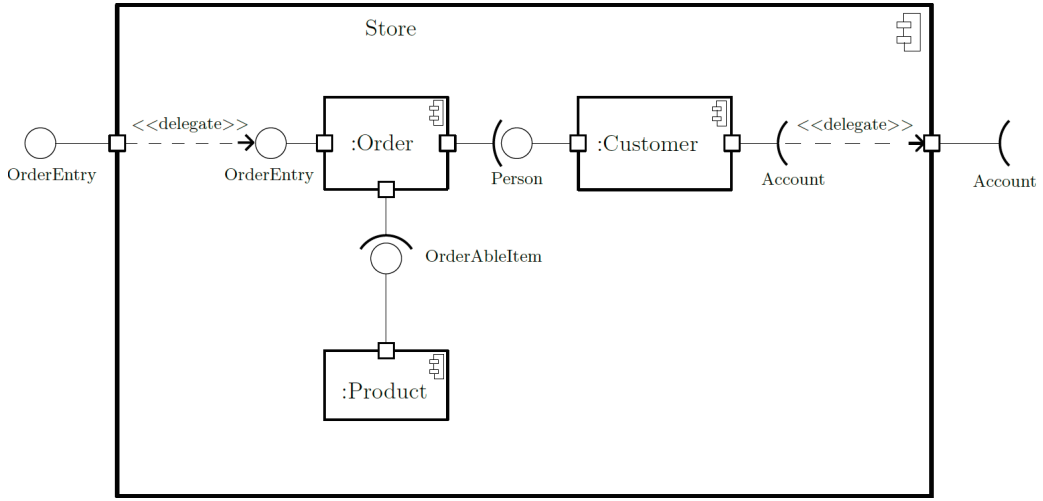
CorC 2.0 - Roundtrip Engineering



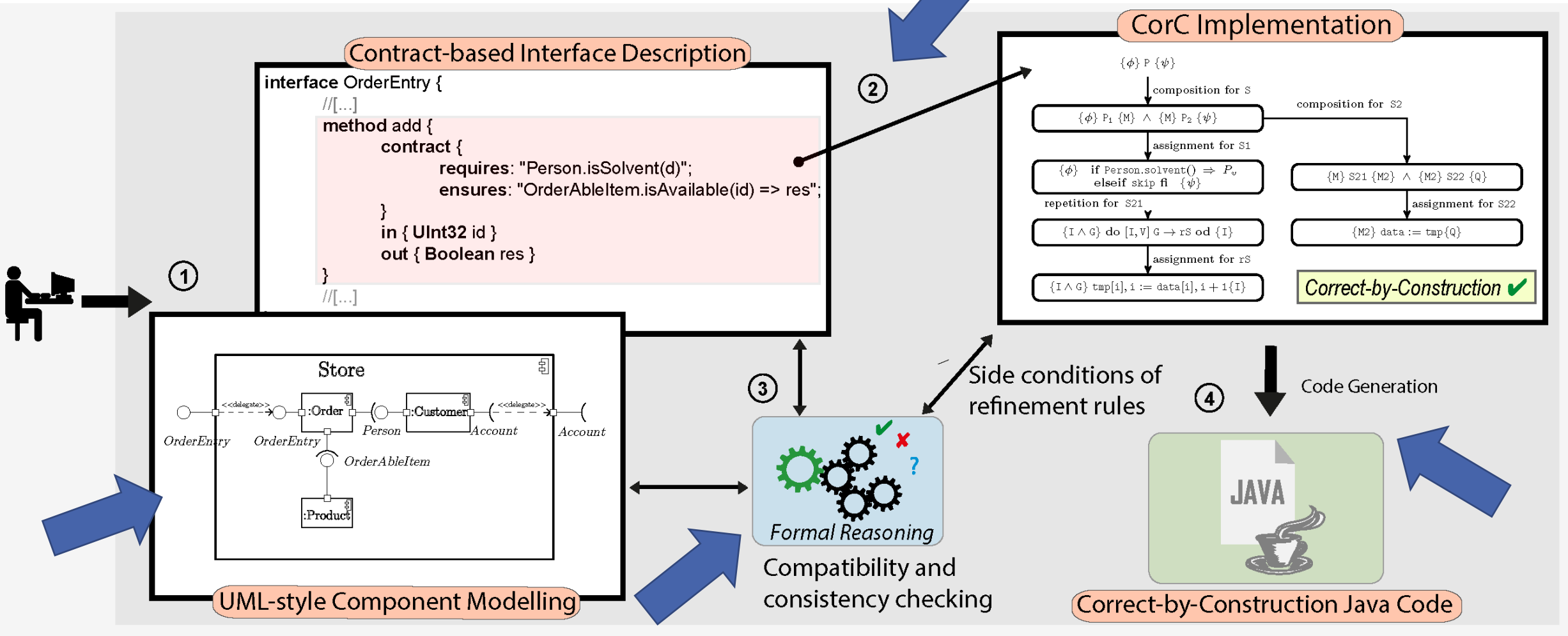
Component-based Software Engineering

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

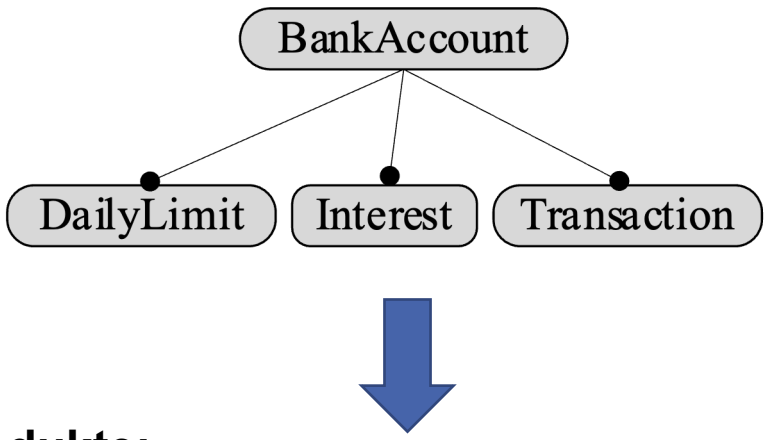


ArchiCorC – Scaling CbC

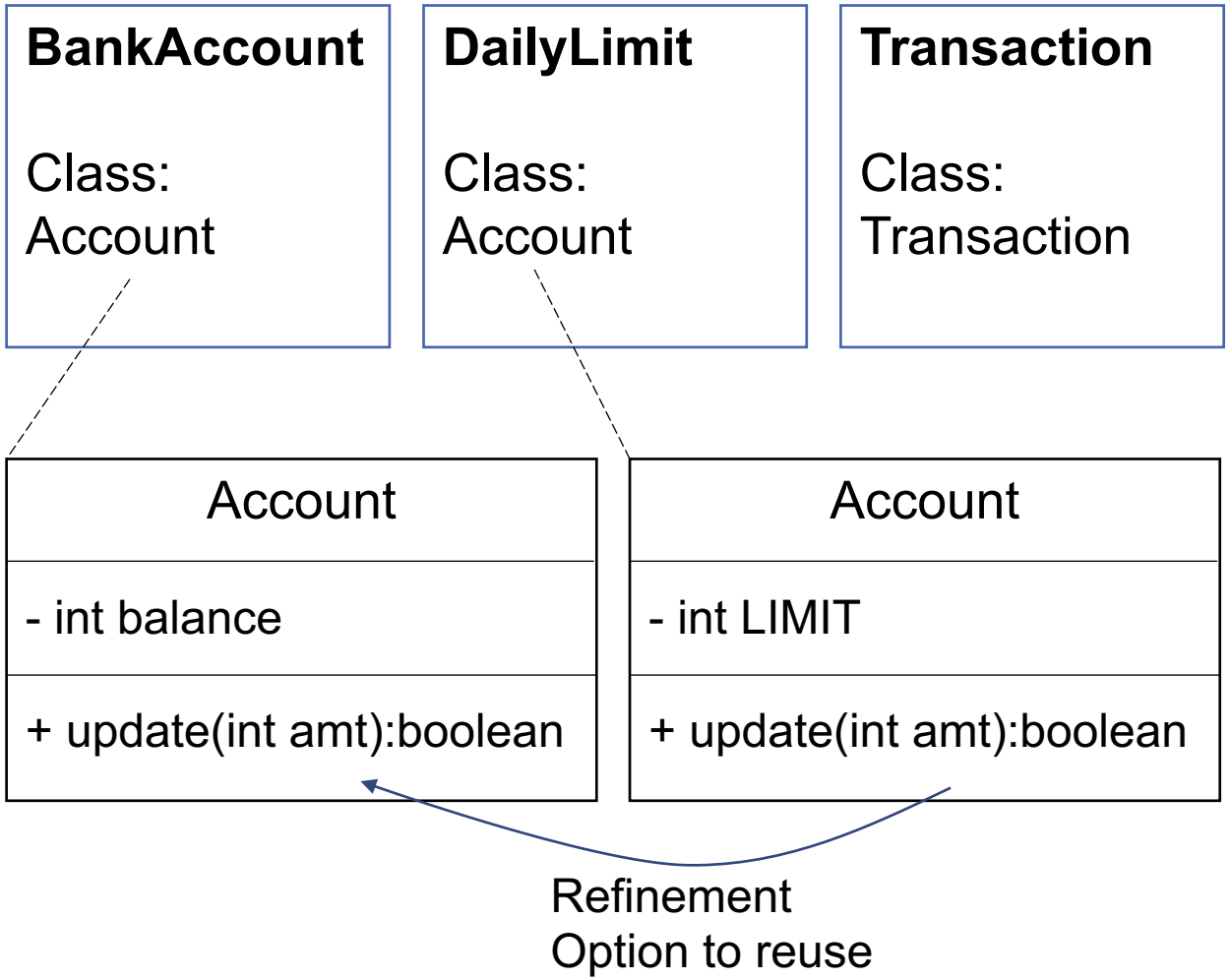


Alexander Knüppel, Tobias Runge, Ina Schaefer: Scaling Correctness-by-Construction. ISoLA (1) 2020: 187-207

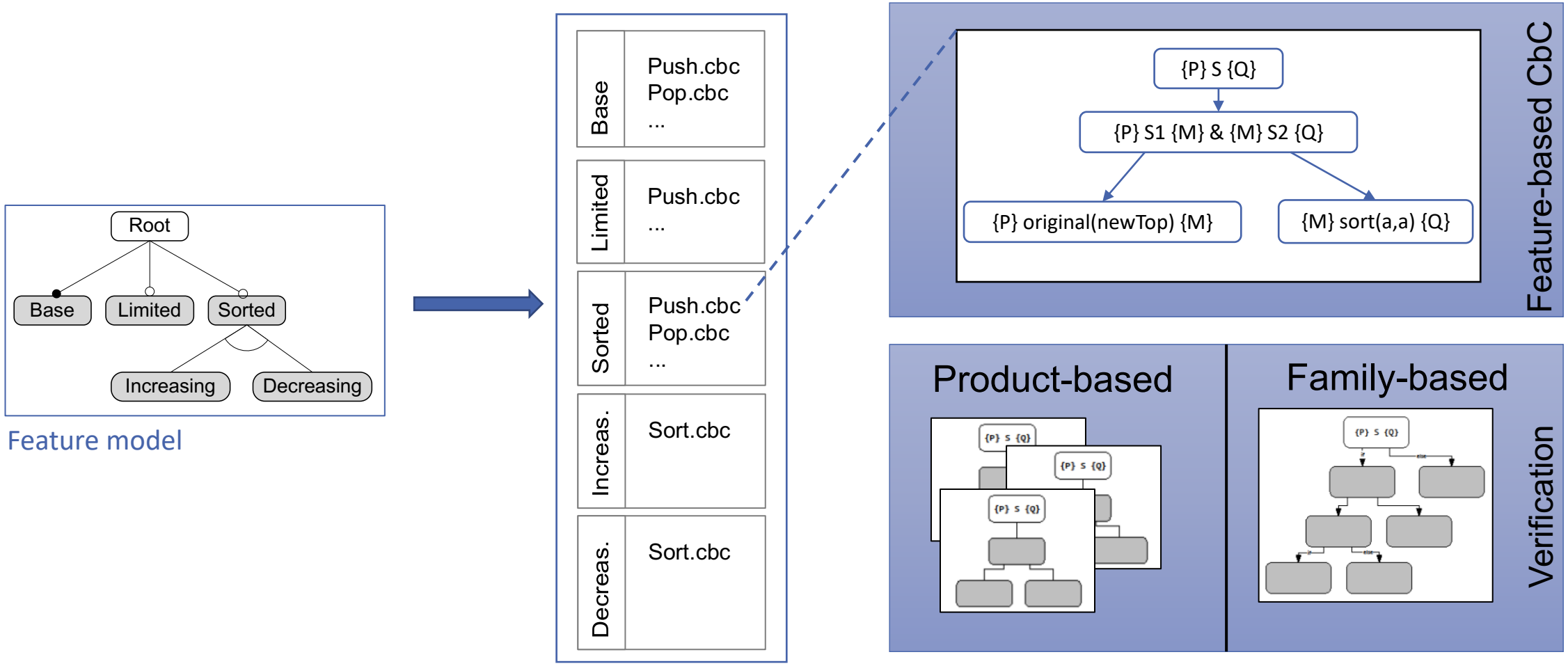
Feature-Oriented Programming



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

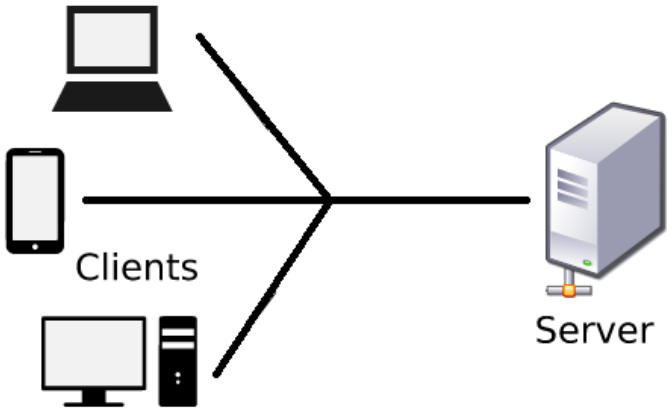


Tabea Bordis, Tobias Runge, David Schultz, Ina Schaefer: Family-Based and Product-Based Development of Correct-by-Construction Software Product Lines. J. Comput. Lang. 70: 101119 (2022)

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

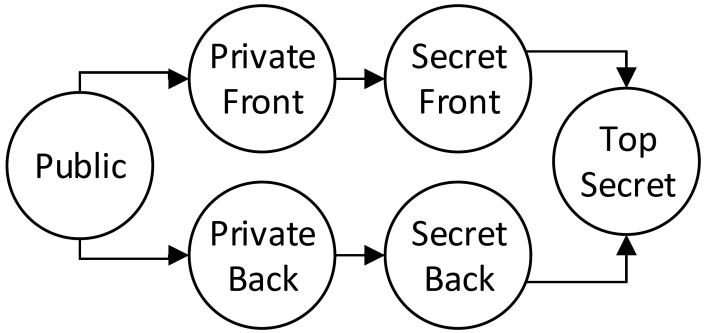


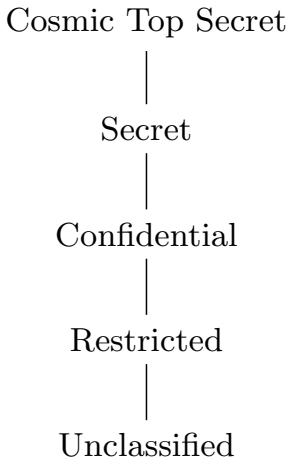
Information Flow Control

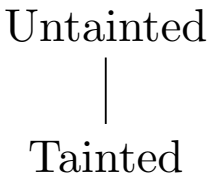
- **Non-interference:** Confidential data may not be deducible from public data

- **Direct Information Flow**: Assignments
- **Indirect Information Flow:** Usage of confidential data in loop-guards and if-conditions

Lattice-based Information Flow Policies







Language-based Security (Post-hoc)

```
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.]

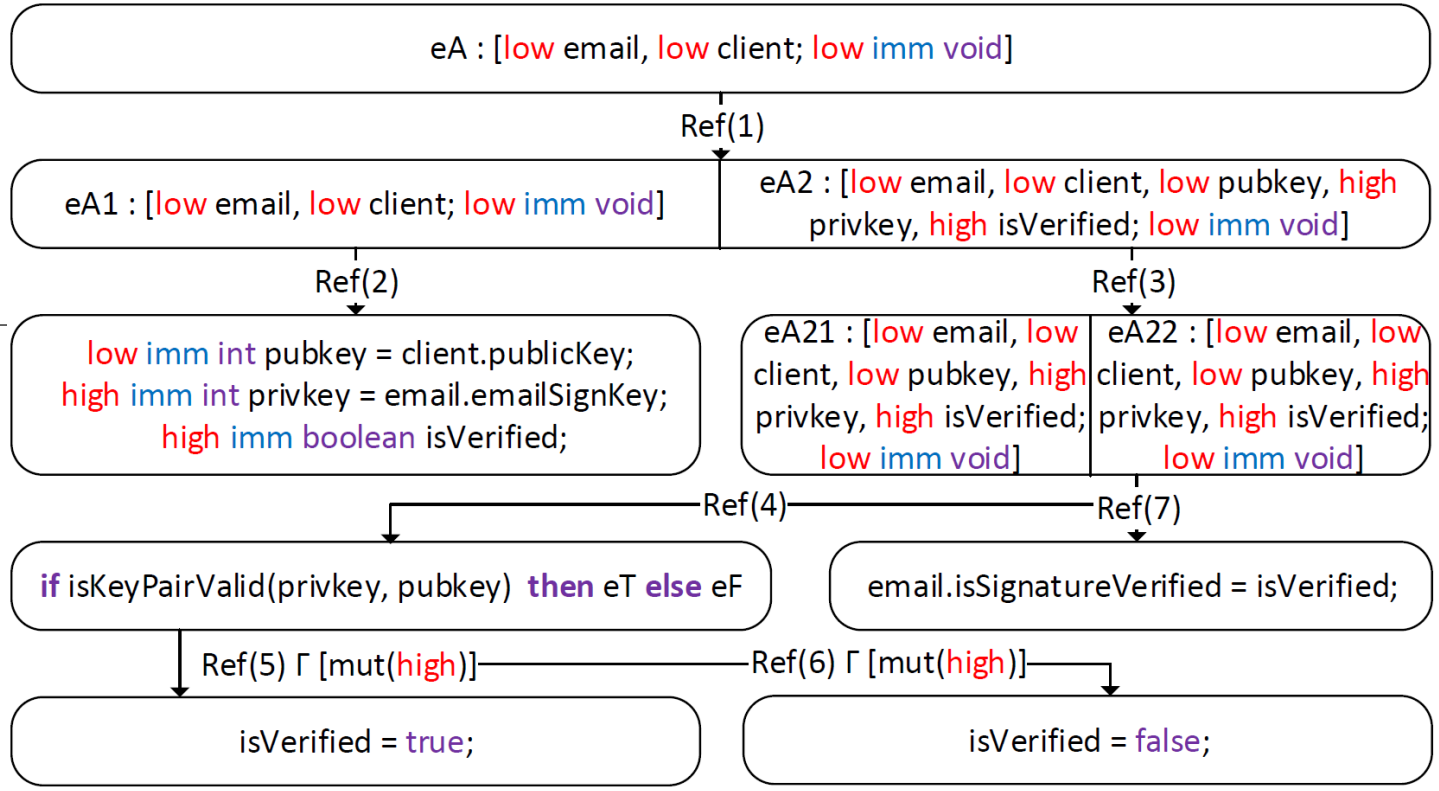
$$\vdash \texttt{exp} : \texttt{high} \qquad \frac{\texttt{h} \notin \texttt{Vars}(\texttt{exp})}{\vdash \texttt{exp} : \texttt{low}} \qquad [\texttt{ct}] \vdash \texttt{skip} \qquad [\texttt{ct}] \vdash \texttt{h} = \texttt{exp} \tag{1–4}$$

$$\frac{\texttt{exp} : \texttt{low}}{[\texttt{low}] \vdash \texttt{l} = \texttt{exp}} \qquad \frac{[\texttt{ct}] \vdash \texttt{S}_1 \quad [\texttt{ct}] \vdash \texttt{S}_2}{[\texttt{ct}] \vdash \texttt{S}_1; \texttt{S}_2} \qquad \frac{\vdash \texttt{exp} : \texttt{ct} \quad [\texttt{ct}] \vdash \texttt{S}}{[\texttt{ct}] \vdash \texttt{while exp do S}} \tag{5–7}$$

$$\frac{\vdash \texttt{exp} : \texttt{ct} \quad [\texttt{ct}] \vdash \texttt{S}_1 \quad [\texttt{ct}] \vdash \texttt{S}_2}{[\texttt{ct}] \vdash \texttt{if exp then S}_1 \texttt{ else S}_2} \tag{8}$$

IFbC – Information Flow-by-Construction

```
1  static low imm void verifySignature(
2    low mut Client client, low mut Email email) {
3    low imm int pubkey = client.publicKey;
4    high imm int privkey = email.emailSignKey;
5    high imm boolean isVerified;
6    if (isKeyPairValid(privkey, pubkey)) {
7      isVerified = true;
8    } else {
9      isVerified = false;
10   }
11   email.IsSignatureVerified = isVerified;
12 }
```



Tobias Runge, Alexander Kittelmann, Marco Servetto, Alex Potanin, Ina Schaefer: Information Flow Control-by-Construction for an Object-Oriented Language. SEFM 2022: 209-226

The CorC Ecosystem

https://github.com/KIT-TVA/CorC

- 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

T. Bordis, L. Cleophas, A. Kittelmann, T. Runge, I. Schaefer, B.W. Watson: Re-CorC-ing KeY: Correct-by-Construction Software Development Based on KeY. The Logic of Software. A Tasting Menu of Formal Methods: 80-104, 2022.

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

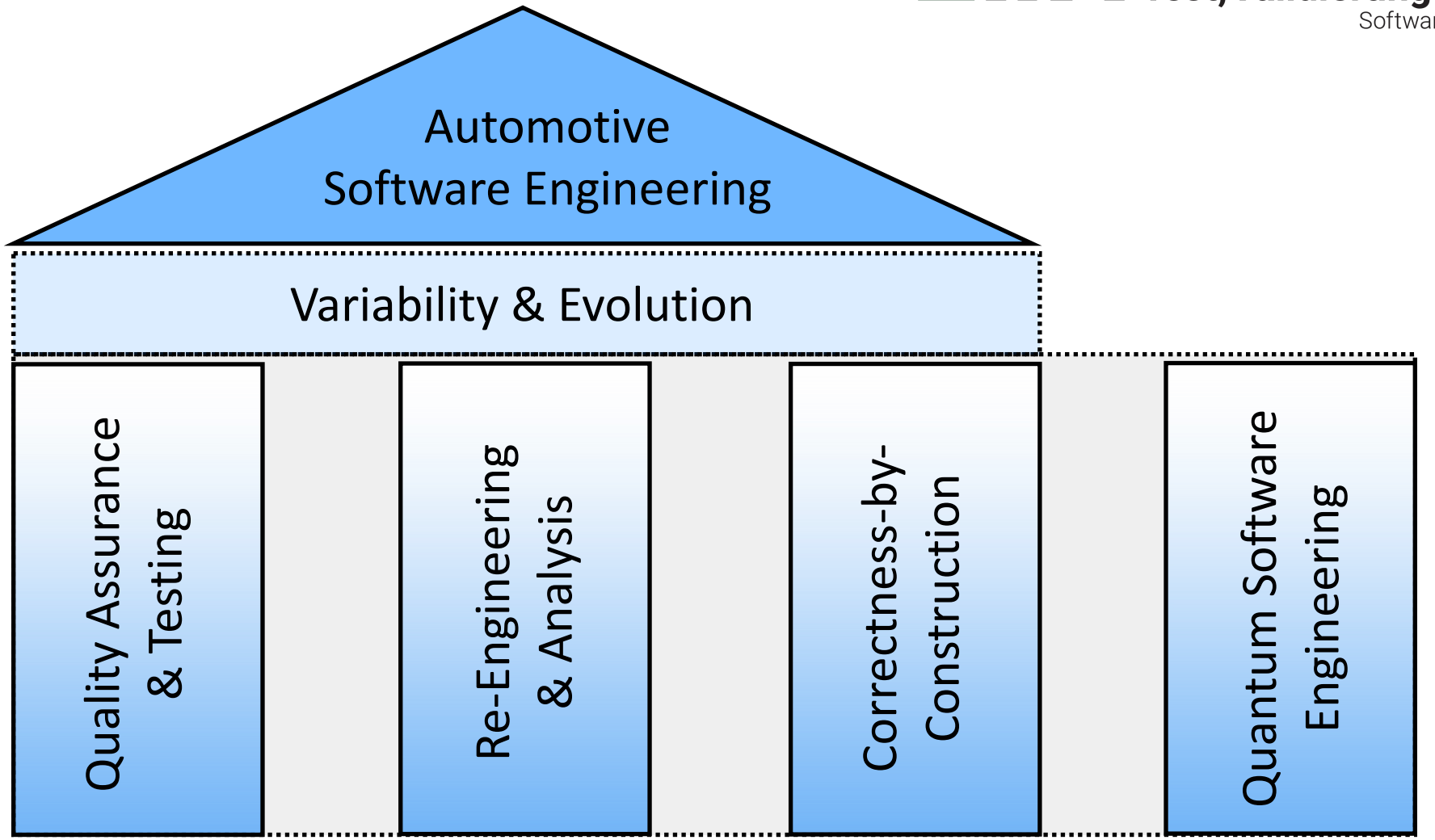
PPT Stock Photo

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

Forschungsgruppe für
Test, Validierung und Analyse
Software-intensiver Systeme



Test, Validierung, Analyse