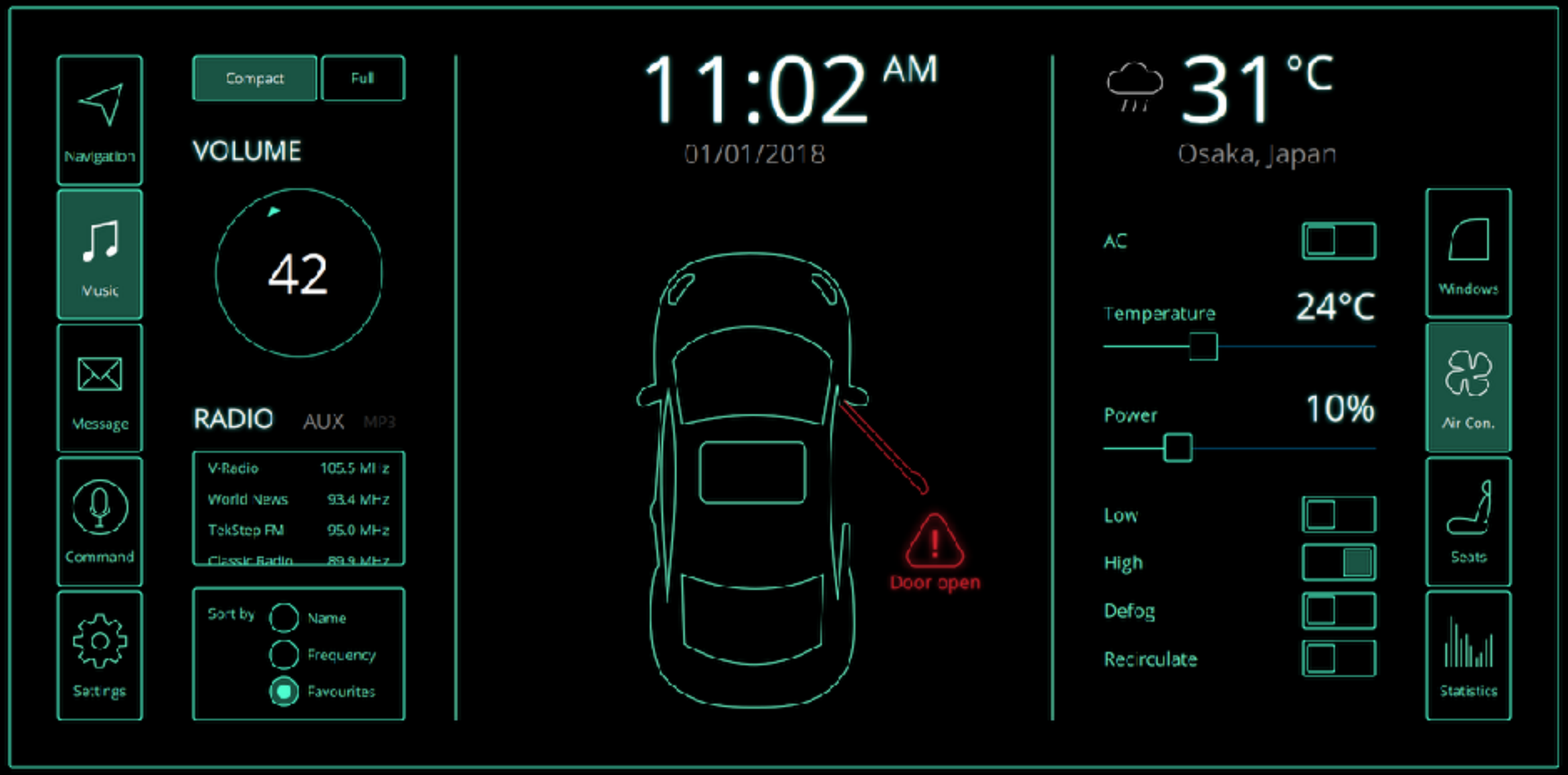


Bounded Model Checking of C++ Programs based on the Qt Cross-Platform Framework

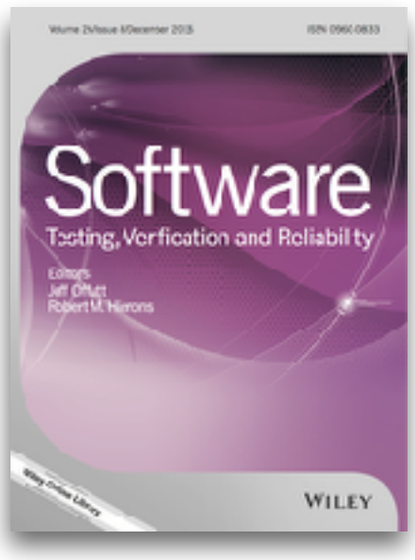
(Journal-First Abstract)

Felipe R. Monteiro
Mário A. P. Garcia
Lucas C. Cordeiro
Eddie B. de Lima Filho

33rd IEEE/ACM International Conference on Automated Software Engineering

Bounded Model Checking of C++ Programs based on the Qt Cross-Platform Framework

Felipe R. Monteiro, Mário A. P. Garcia, Lucas C. Cordeiro, and Eddie B. de Lima Filho



Federal University of Amazonas

Motivation

Why should we ensure software reliability?

Why should we ensure software reliability?

- Consumer electronic products must be as robust and bug-free as possible, given that even medium product-return rates tend to be unacceptable

"*Engineers reported the static analyser* ***Infer*** *was key to build a concurrent version of Facebook app to the Android platform.*"

- Peter O'Hearn, FLoC, 2018.

Why should we ensure software reliability?

- Consumer electronic products must be as robust and bug-free as possible, given that even medium product-return rates tend to be unacceptable

  - In 2014, Apple revealed a bug known as **Gotofail**, which was caused by a single misplaced “goto” command in the code

  - “*Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS*”

– Apple Inc., 2014.

Why should we ensure software reliability?

- Consumer electronic products must be as robust and bug-free as possible, given that even medium product-return rates tend to be unacceptable

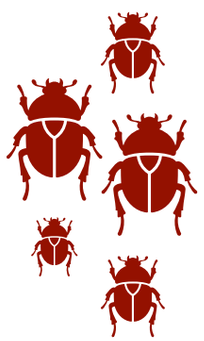
  - In 2014, Apple revealed a bug known as **Gotofail**, which was caused by a single misplaced “goto” command in the code

  - “*Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS*”

    – Apple Inc., 2014.

  - “*Mozilla browser has around **20,000** open bugs*”

    – Michail, Amir. ICSE, 2005.



Industry NEEDS Formal Verification

"*There has been a tremendous amount of valuable research in formal methods, but rarely have formal reasoning techniques been deployed as part of the development process of large industrial codebases.*"

facebook research

- Peter O'Hearn, FLoC, 2018.

amazon web services

"*Formal automated reasoning is one of the investments that AWS is making in order to facilitate continued simultaneous growth in both functionality and security.*"

- Byron Cook, FLoC, 2018.

Our main goals is to…

Extend the analysis power of model checkers through operational models to support linked libraries and frameworks

We demonstrate in this paper how to…

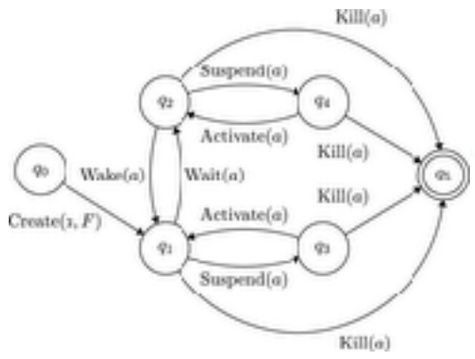
Apply model checking techniques to formally verify Qt-based applications

Background

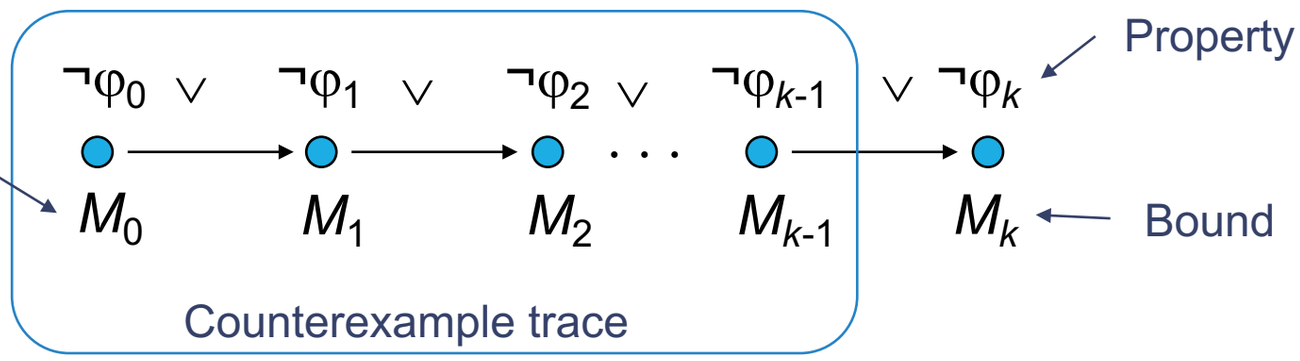
Model Checking

Bounded Model Checking

- Basic Idea: given a transition system *M*, check negation of a given property ϕ up to given depth *k*



Transition System

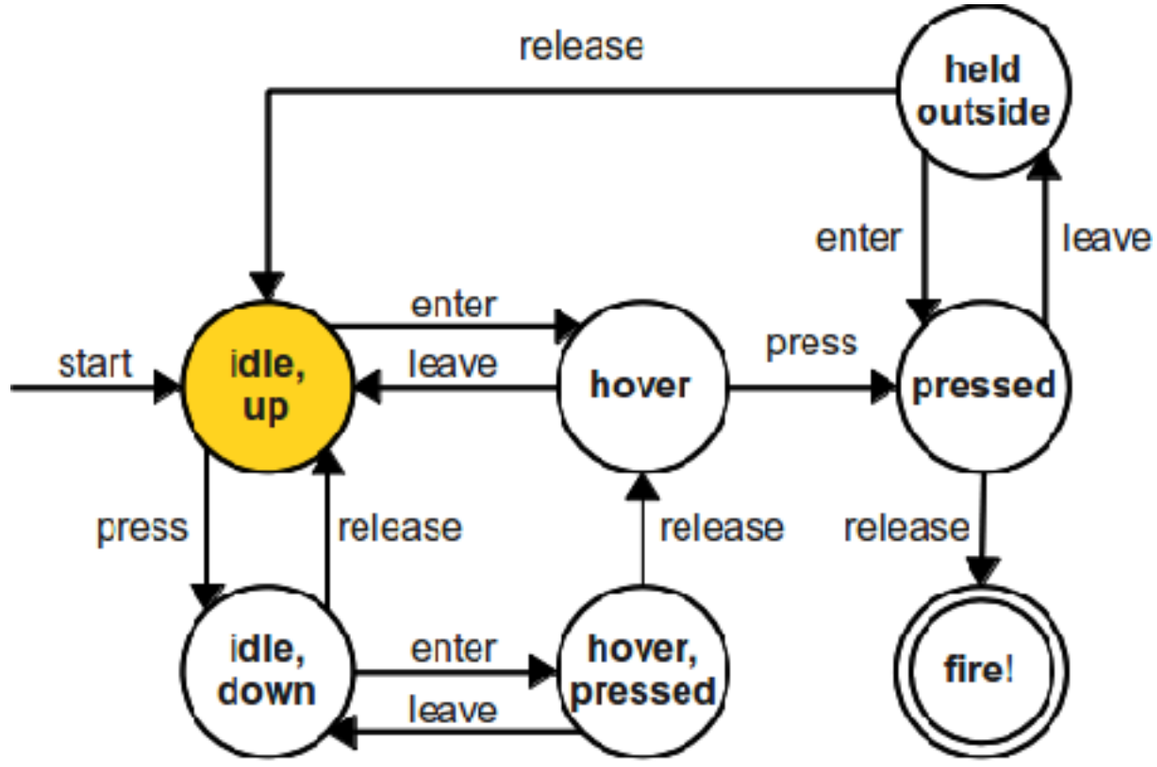


- Translated into a VC ψ such that: **ψ is satisfiable iff ϕ has counterexample of max. depth *k***
- BMC has been applied successfully to verify (embedded) software since early 2000's.

Explicit-State Model Checking

- **Basic Idea**: represent state transition graph explicitly
  - also represents the system as a finite-state machine
  - states are enumerated on-the-fly
  - forward analysis

- Some characteristics
  - memory intensive
  - good for finding concurrency erros
  - this approach can handle dynamic creation of objects/ threads

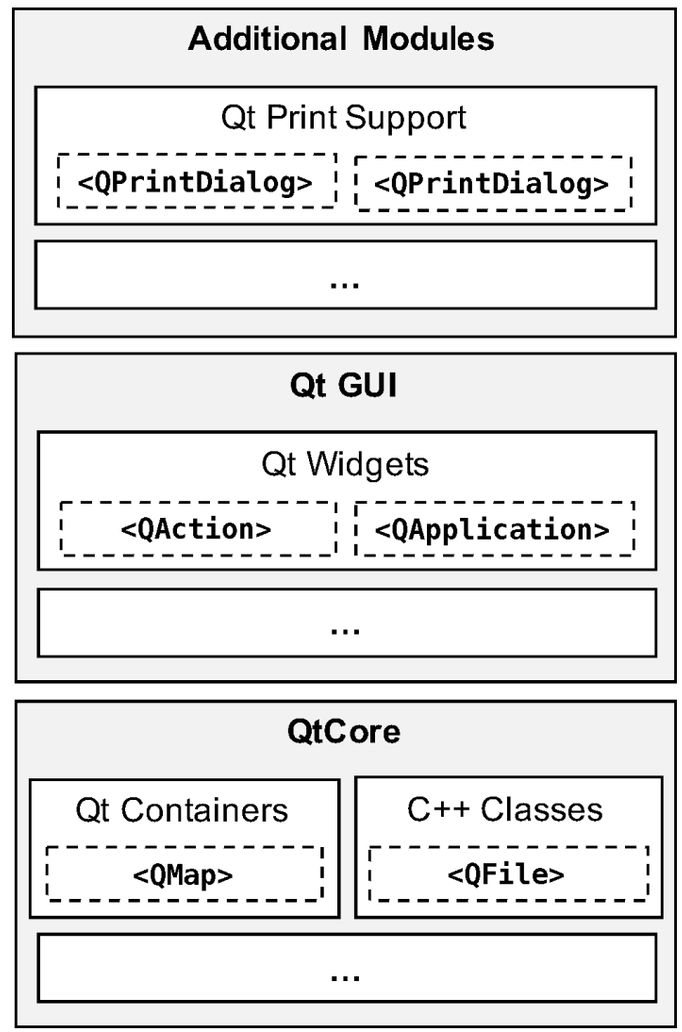


Qt Cross-Platform Framework

- Qt framework provides programs that run on different hardware/software platforms, with as few changes as possible, while maintaining the same power and speed

8 of Top 10 Fortune 500 use

- Its libraries are organised into modules that rely on two main cores:
  - **QtCore** – contains all non-graphical core classes
  - **QtGUI** – provides a complete abstraction for the Graphical User Interface



Qt in Action

Veriskin fights skin cancer with a lightning-fast and low-cost diagnostics device built with Qt

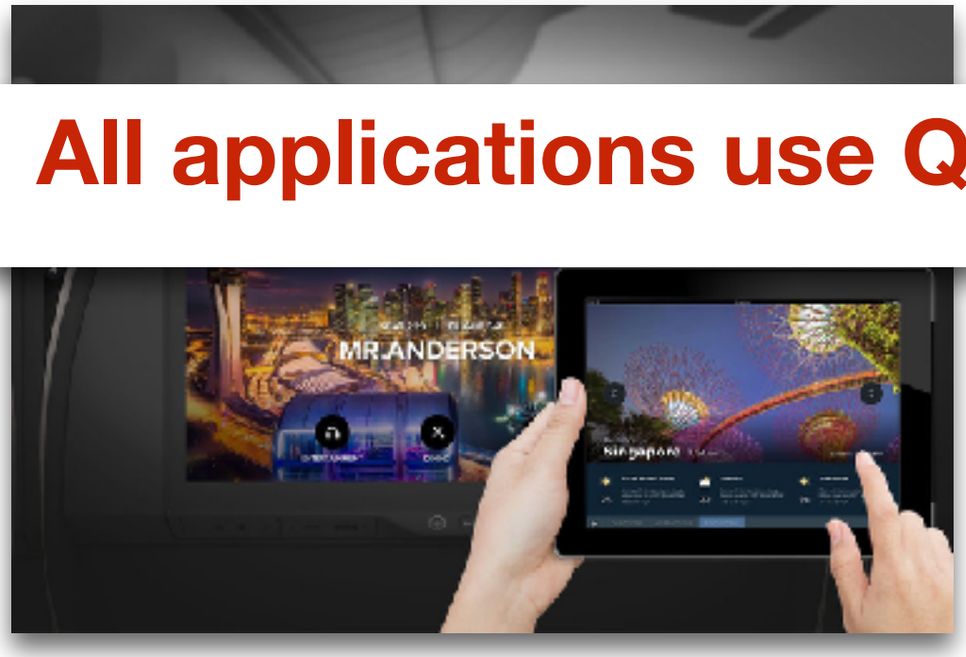
Panasonic Avionics Inflight Entertainment is built with Qt

“To fulfil on its vision as a single OS running on a full range of displays, interfaces and inputs Ubuntu relies on Qt to create the rich visual components which deliver the Ubuntu User Interface that is familiar to the many millions of Ubuntu users.”

Richard Collins, Ubuntu Mobile Product Manager at Canonical

Qt in Action

Veriskin fights skin cancer with a lightning-fast and low-cost diagnostics device built with Qt



All applications use QtCore and QtGUI modules

Panasonic Avionics Inflight Entertainment is built with Qt

“To fulfil on its vision as a single OS running on a full range of displays, interfaces and inputs Ubuntu relies on Qt to create the rich visual components which deliver the Ubuntu User Interface that is familiar to the many millions of Ubuntu users.”

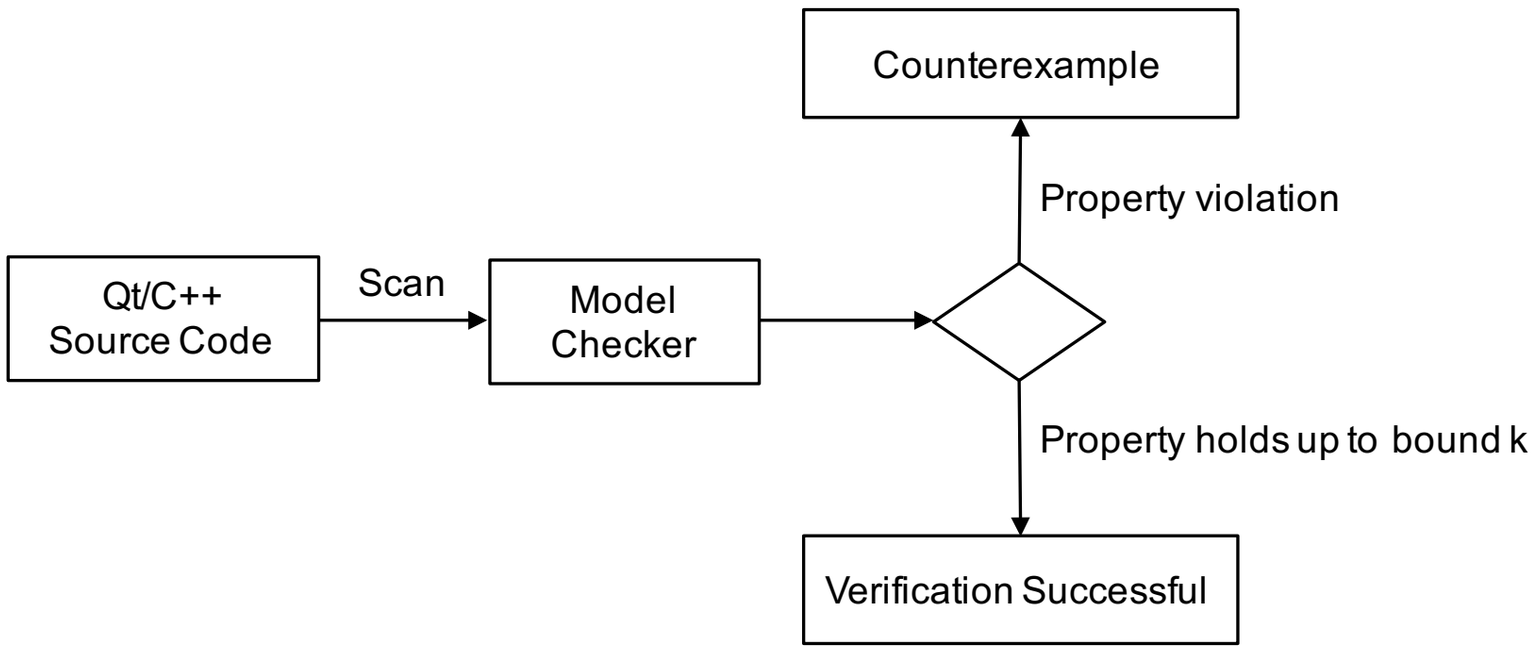
Richard Collins, Ubuntu Mobile Product Manager at Canonical

Approach and Uniqueness

Qt Operational Model

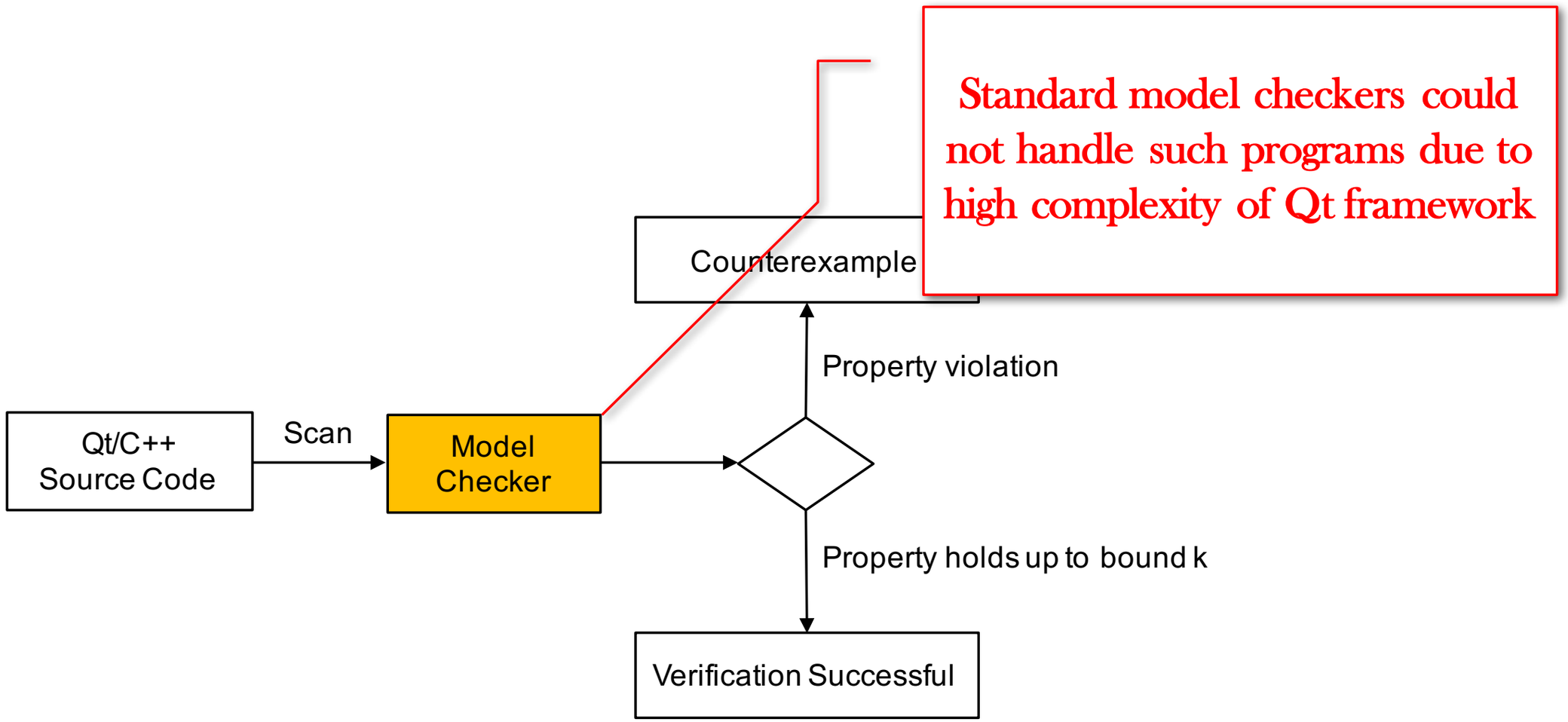
Qt Operational Model

- QtOM is a simplified representation that considers the structure of each Qt library and its associated classes, including attributes, method signatures, and function prototypes.



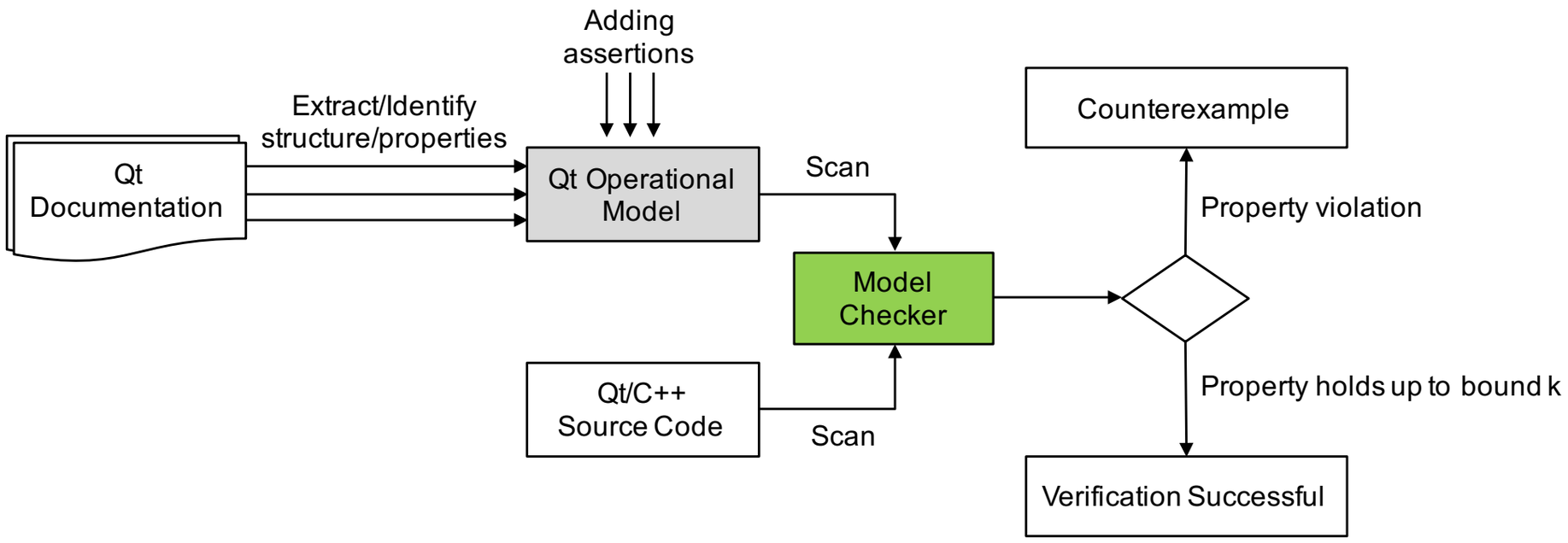
Qt Operational Model

- QtOM is a simplified representation that considers the structure of each Qt library and its associated classes, including attributes, method signatures, and function prototypes.



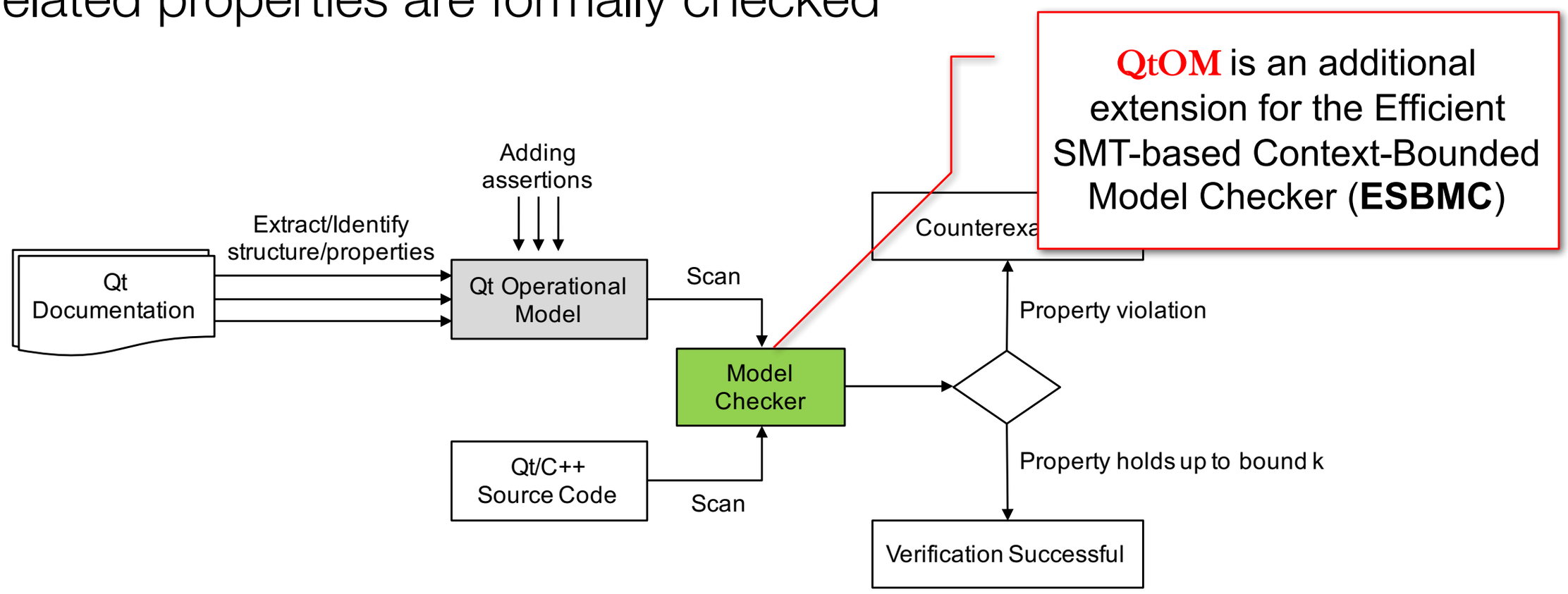
Qt Operational Model

- QtOM is a simplified representation that considers the structure of each Qt library and its associated classes, including attributes, method signatures, and function prototypes
  - QtOM also includes **assertions**, which ensure that specific Qt related properties are formally checked



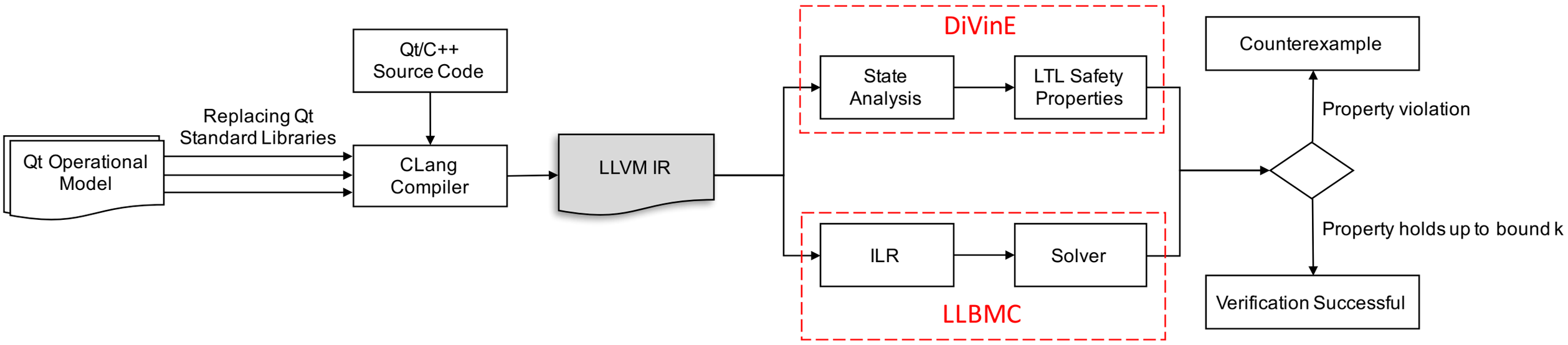
Qt Operational Model

- QtOM is a simplified representation that considers the structure of each Qt library and its associated classes, including attributes, method signatures, and function prototypes
  - QtOM also includes **assertions**, which ensure that specific Qt related properties are formally checked



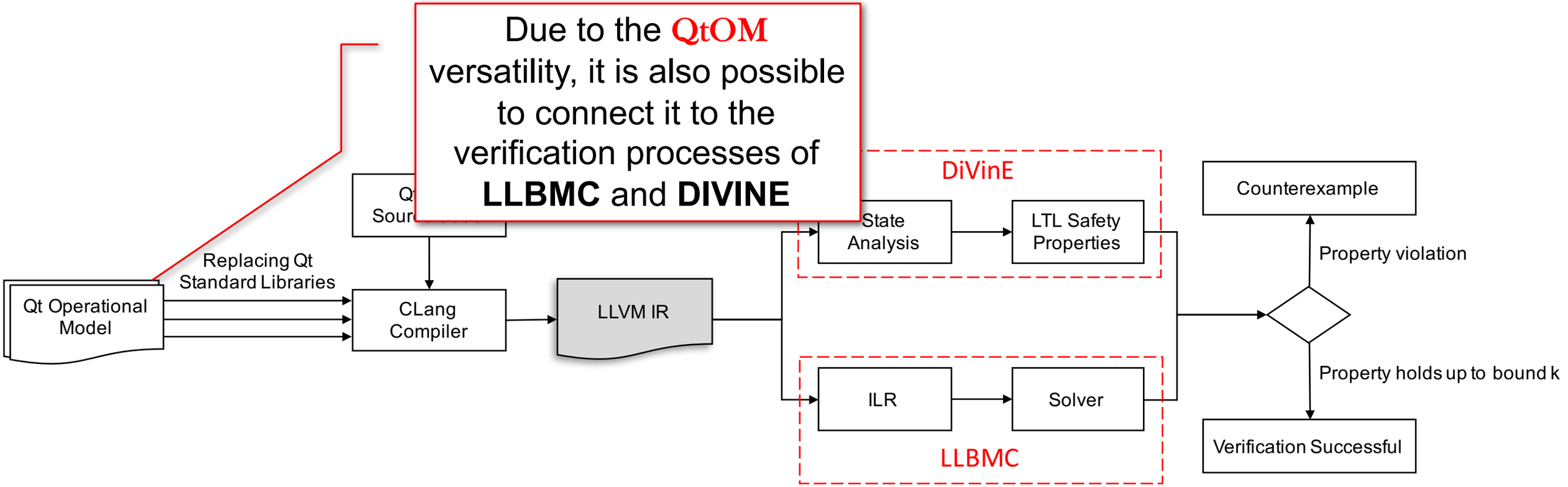
Qt Operational Model

- QtOM is a simplified representation that considers the structure of each Qt library and its associated classes, including attributes, method signatures, and function prototypes
  - QtOM also includes **assertions**, which ensure that specific Qt related properties are formally checked



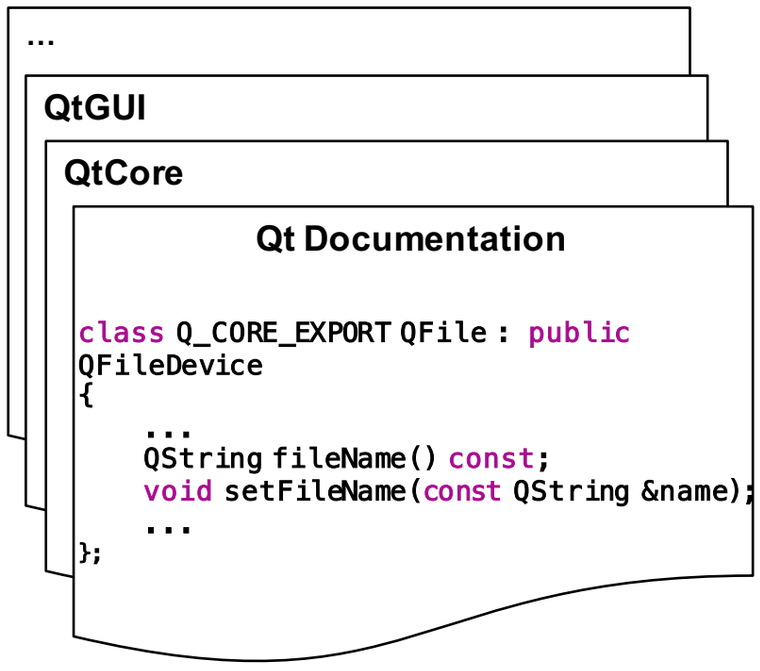
Qt Operational Model

- QtOM is a simplified representation that considers the structure of each Qt library and its associated classes, including attributes, method signatures, and function prototypes
  - QtOM also includes **assertions**, which ensure that specific Qt related properties are formally checked



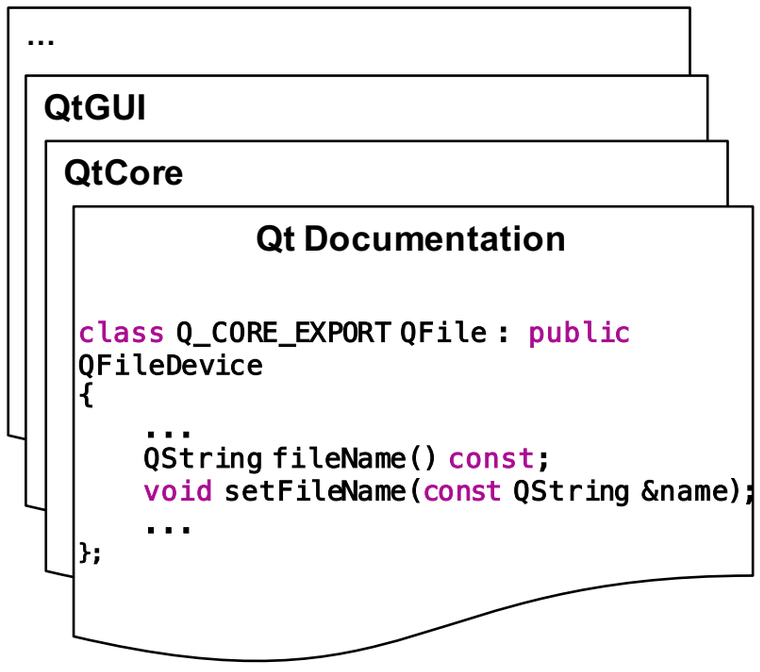
Building Operational Models

- We base the development process of operational models in the **documentation** of the target framework
  - Identify the structure of the framework, focusing on portions (e.g., most used libraries and classes)



Building Operational Models

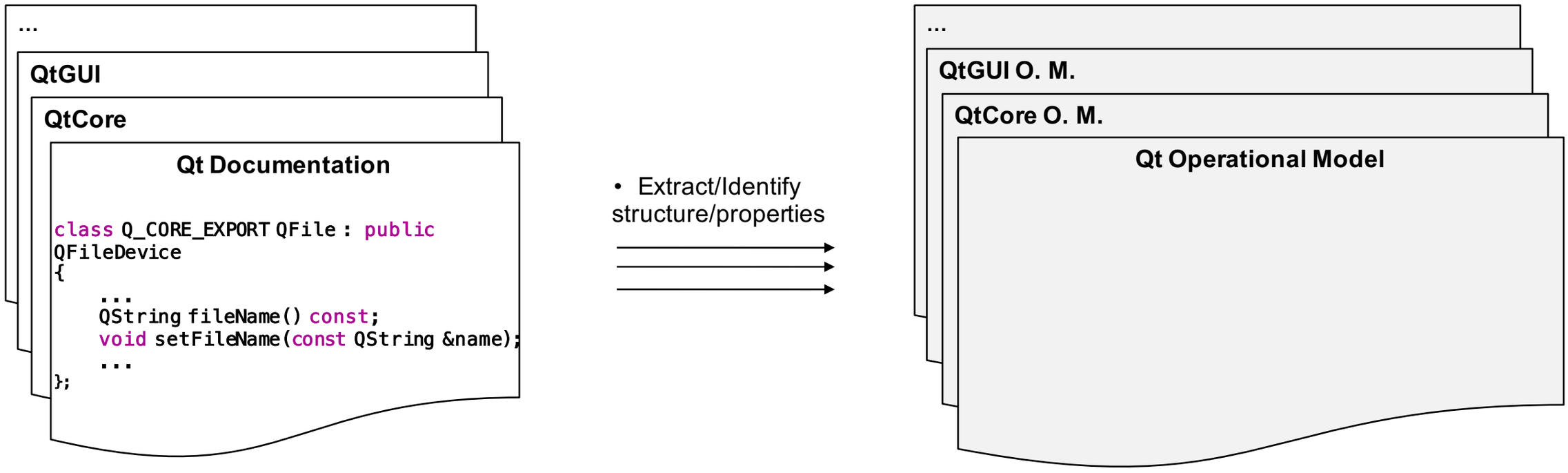
- We base the development process of operational models in the **documentation** of the target framework
  - Identify each property to be verified and transcript them into assertions (i.e., pre- and postconditions)



- Extract/Identify structure/properties

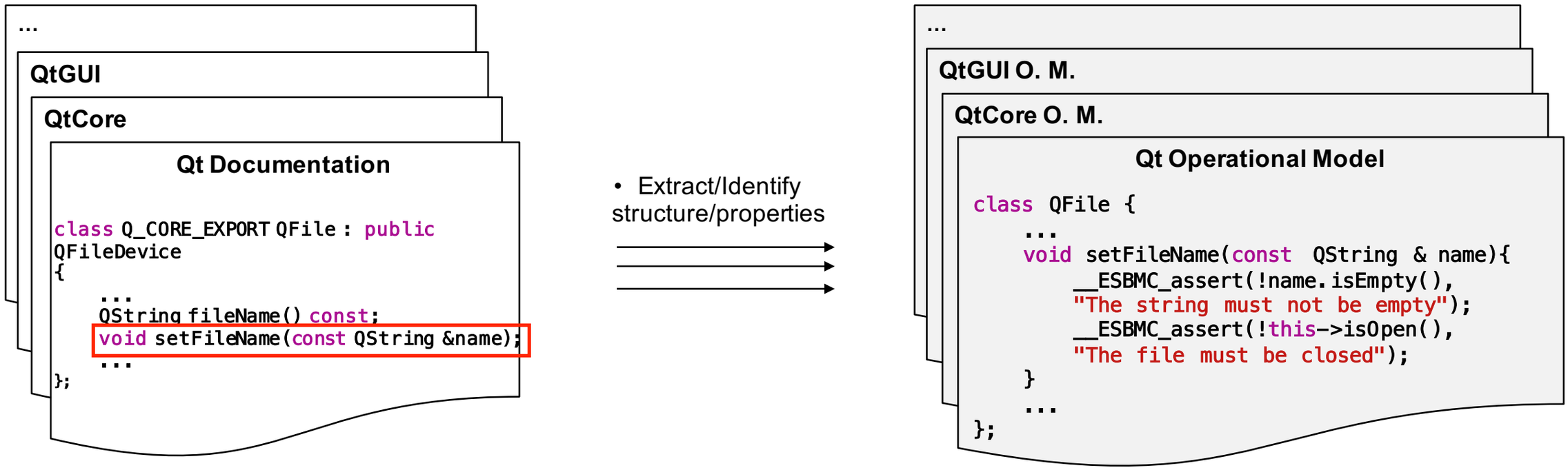
Building Operational Models

- We base the development process of operational models in the **documentation** of the target framework
  - QtOM is an abstract representation, which is used to identify elements and verify specific properties related to such structures



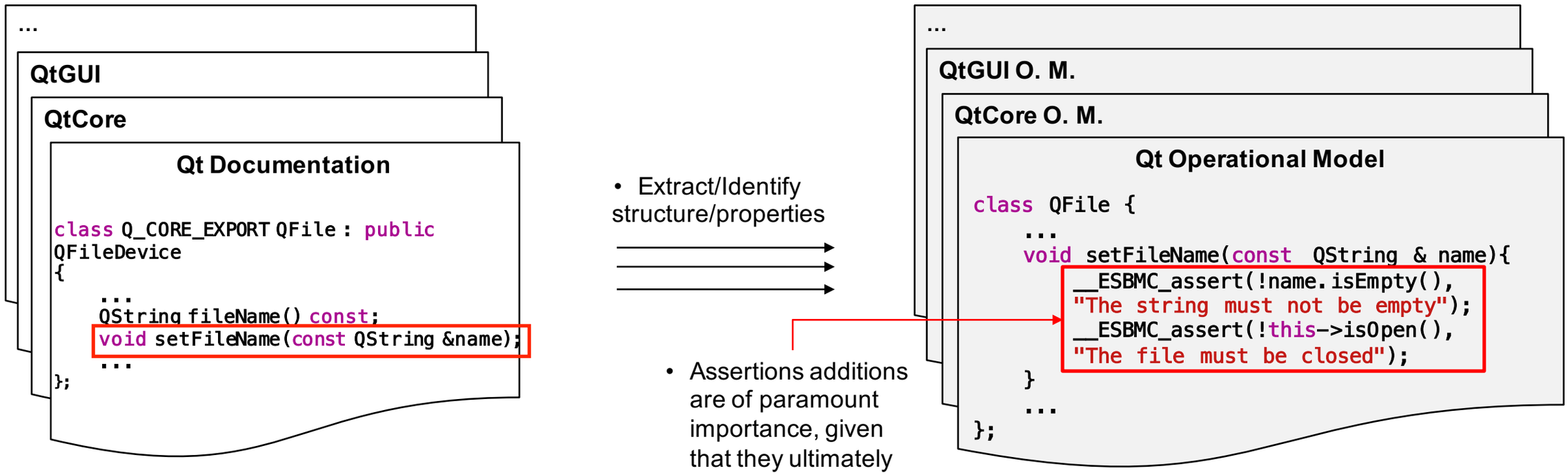
Building Operational Models

- We base the development process of operational models in the **documentation** of the target framework
  - QtOM is an abstract representation, which is used to identify elements and verify specific properties related to such structures



Building Operational Models

- We base the development process of operational models in the **documentation** of the target framework
  - QtOM is an abstract representation, which is used to identify elements and verify specific properties related to such structures

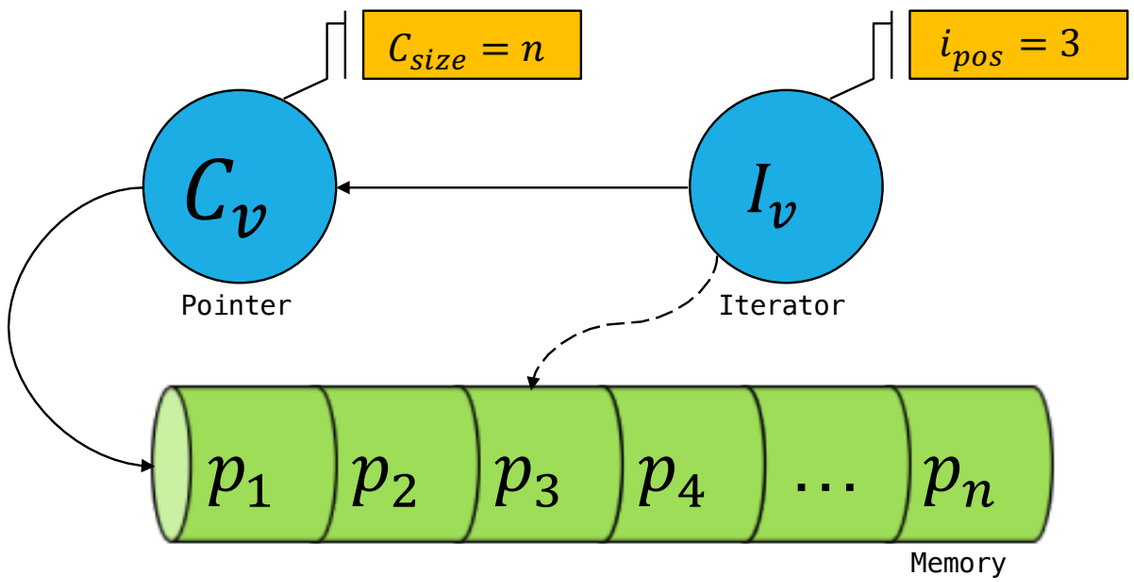


Operational Models for Containers

"The Qt library provides a set of general purpose template-based container classes. These classes can be used to store items of a specified type. For example, if you need a resizable array of QStrings, use QVector<QString>."

The Qt Company Ltd., 2018.

- Qt **sequential containers** are built into a structure to store elements, in a certain sequential order.
- Note that all methods, from those libraries, can be expressed as simplified variations of 3 main operations:
  - insertion **C.insert (I, V, N)**
  - deletion **C.erase (I)**
  - search **C.search (V)**

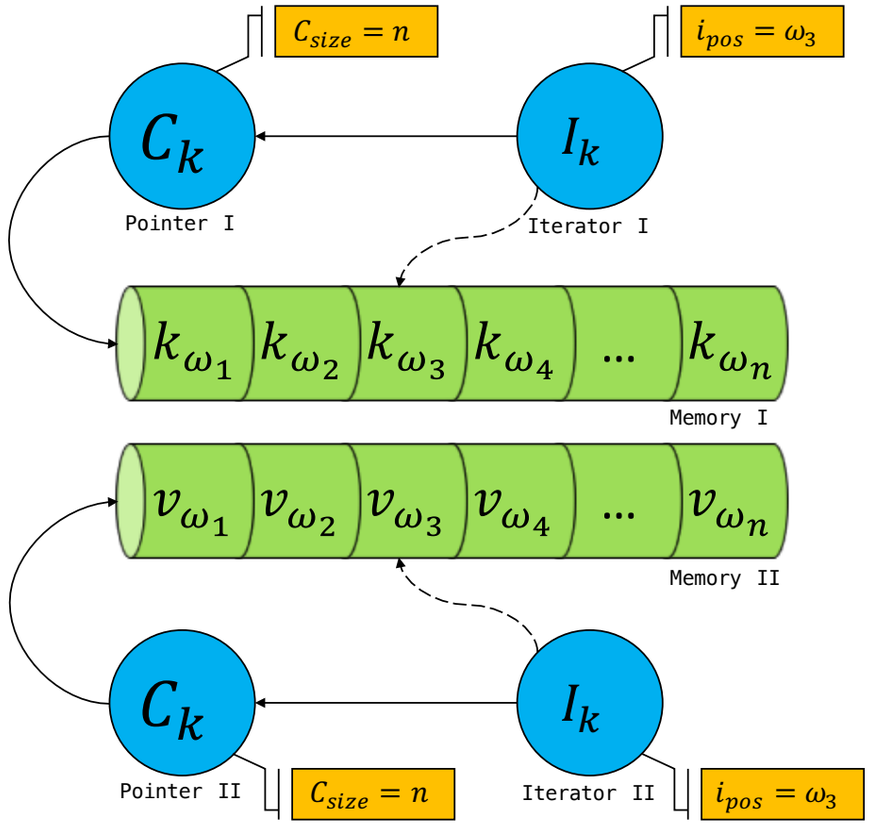


Operational Models for Containers

"*The Qt library provides a set of general purpose template-based container classes. These classes can be used to store items of a specified type. For example, if you need a resizable array of QStrings, use QVector<QString>.*"

The Qt Company Ltd., 2018.

- Qt **associative containers** connects each key, of a certain type **K**, to a value, of a certain type **V**, where associated keys are stored in order.
- Note that all methods, from those libraries, can be expressed as simplified variations of three main operations:
  - insertion **C.insert (I, V, N)**
  - deletion **C.erase (I)**
  - search **C.search (K)**



Experimental Evaluation

Evaluate the soundness of DSVerifier

Setup

- Our set of benchmarks contains **711 Qt/C++ programs** (12,903 LOC).
  - 353 out of the 711 benchmarks contain bugs (i.e., 49.65%) and
  - 358 are flawless (i.e., 50.35%).

- The mentioned benchmarks are split into ten main suites: *QHash*, *QLinkedList*, *QList*, *QMap*, *QMultiHash*, *QMultiMap*, *QQueue*, *QSet*, *QStack*, and *QVector*.

- **ESBMC 1.25.4**
  - Z3 v4.0, Boolector v2.0.1, and Yices 2 v4.1

- **LLBMC v2013.1**

- **DiVinE v3.3.2**

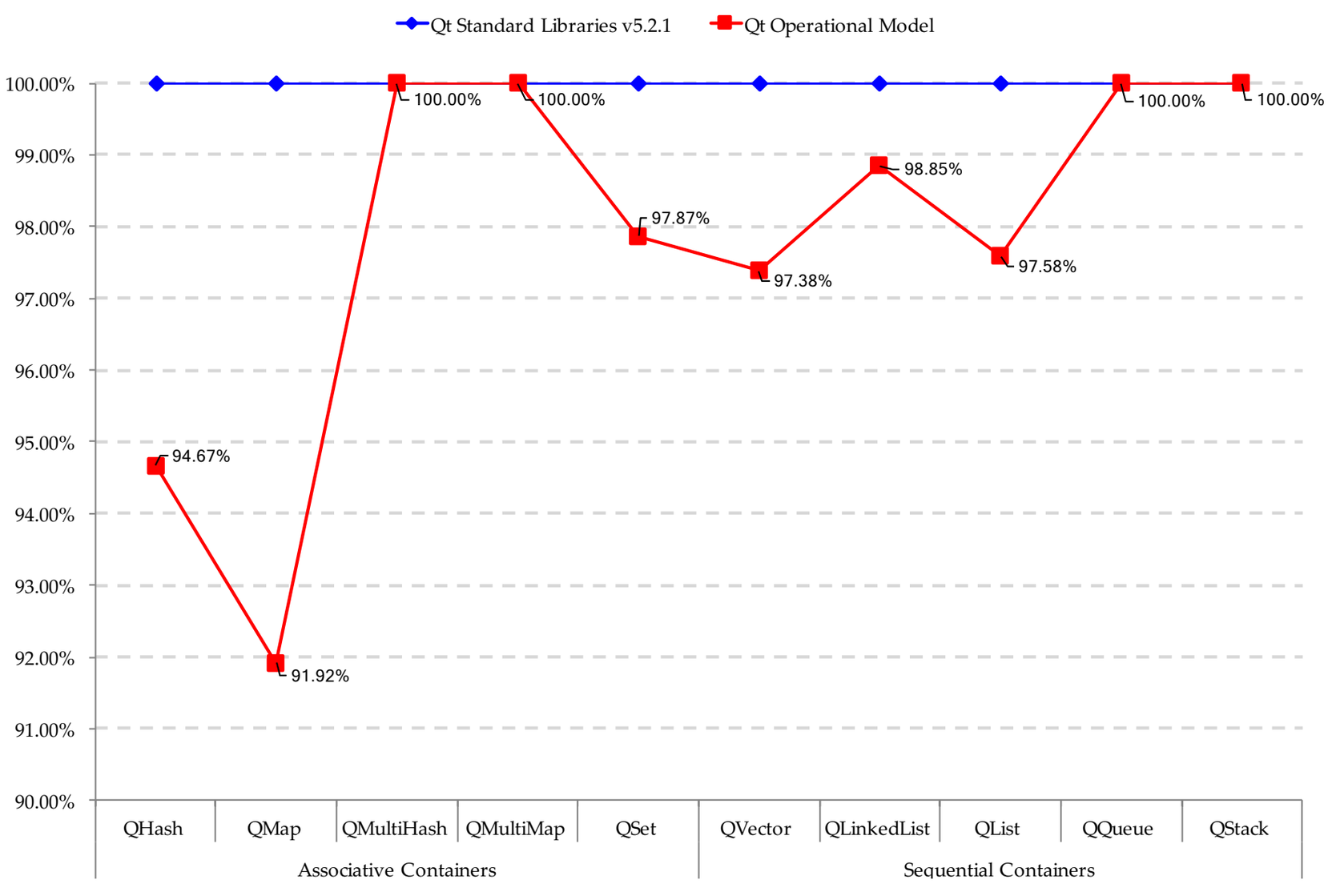
- All experiments were conducted on an otherwise idle Intel Core i7-4790, with 3.60 GHz clock and 24 GB (22 GB of RAM and 2 GB of swap space), running Fedora OS (64 bits)

- The time and memory limits, for each benchmark, were set to 600 seconds and 22 GB, respectively.

Conformance Testing

- the basic idea is to compare the behaviour of standard Qt libraries to QtOM, with the goal of measuring their similarity

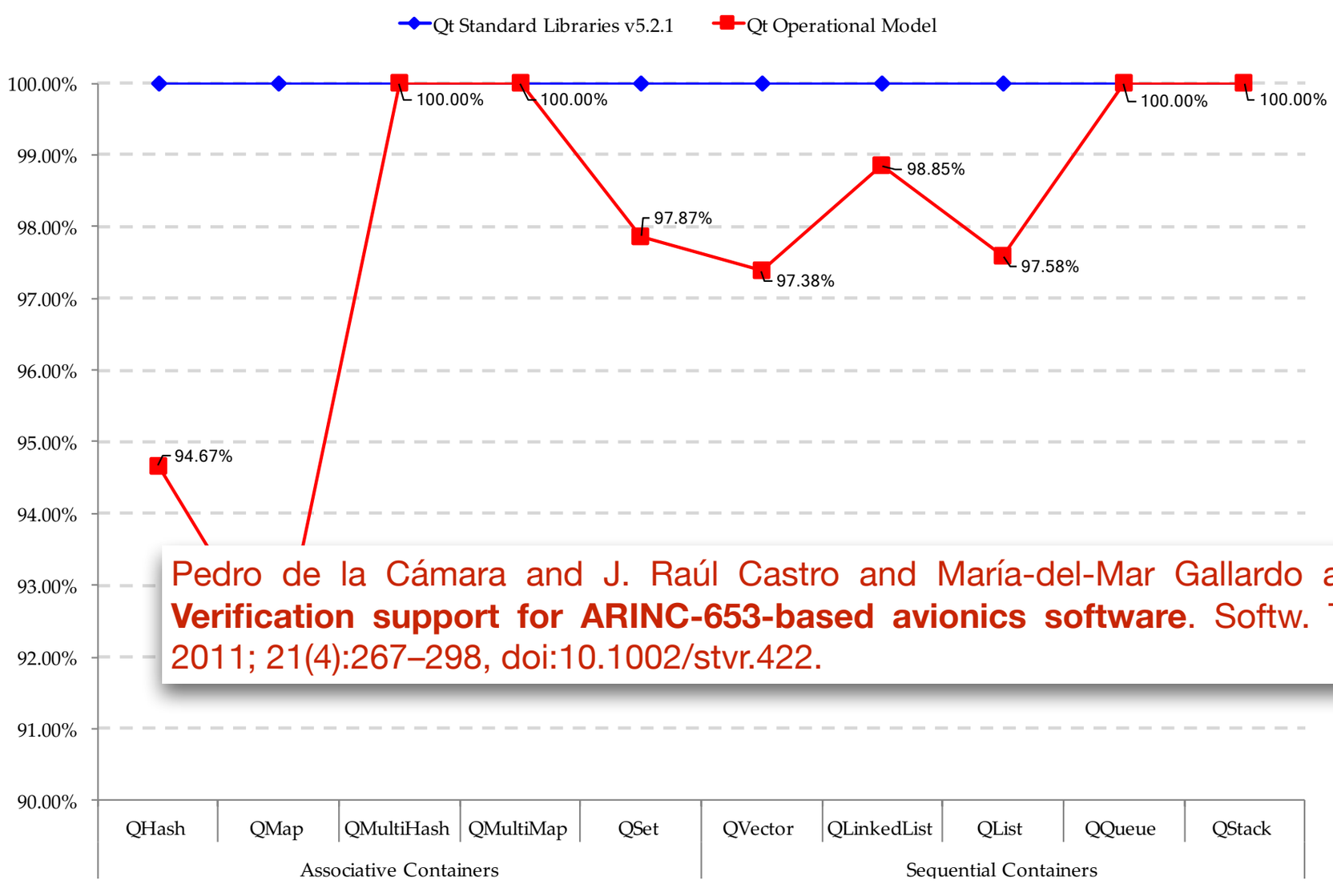
- they all contain **pre- and postconditions**, with the goal to ensure that a (given) predicate holds **before** and **after** the execution of a (given) function



Conformance Testing

- the basic idea is to compare the behaviour of standard Qt libraries to QtOM, with the goal of measuring their similarity

- they all contain **pre- and postconditions**, with the goal to ensure that a (given) predicate holds **before** and **after** the execution of a (given) function

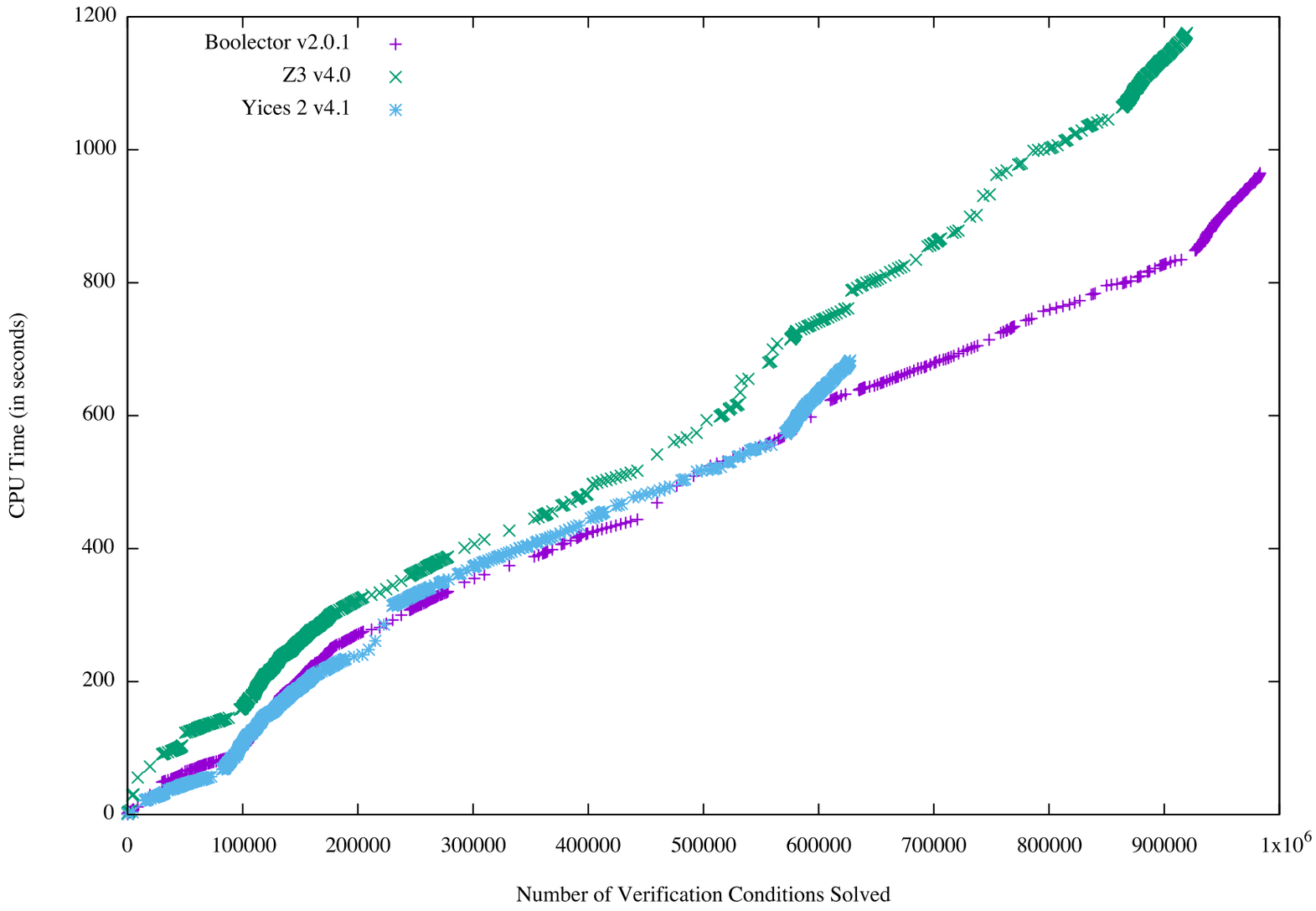


Pedro de la Cámara and J. Raúl Castro and María-del-Mar Gallardo and Pedro Merino. **Verification support for ARINC-653-based avionics software**. Softw. Test., Verif. Reliab. 2011; 21(4):267–298, doi:10.1002/stvr.422.

SMT Solver Evaluation

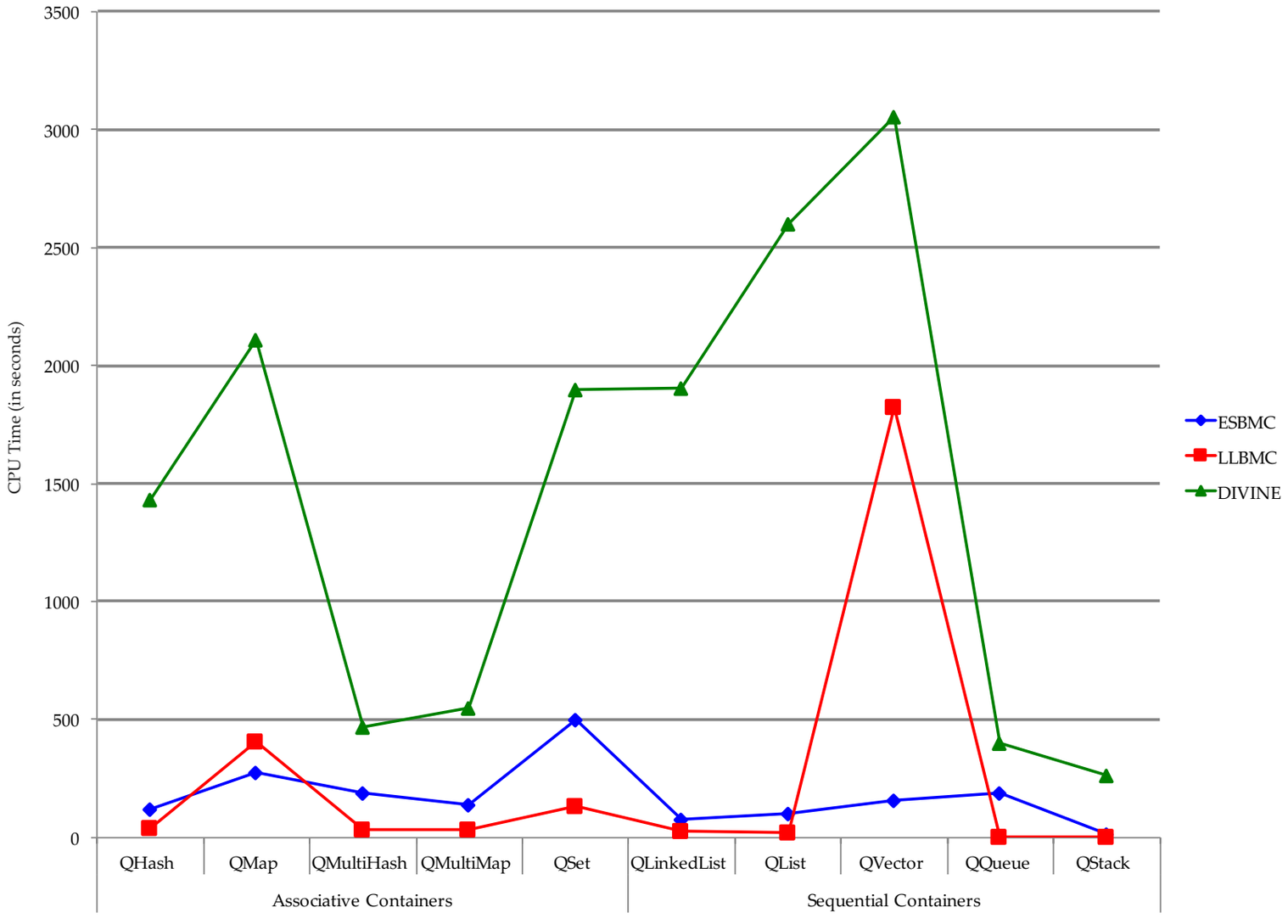
- It is widely known that different SMT solvers can heavily affect the verification process, therefore, verification using the three mentioned SMT solvers were performed: **Z3**, **Boolector**, and **Yices**

- The results are evaluated by means of **coverage** (*i.e.*, percentage of correct verifications) and **verification time**



Comparison Against Model Checkers

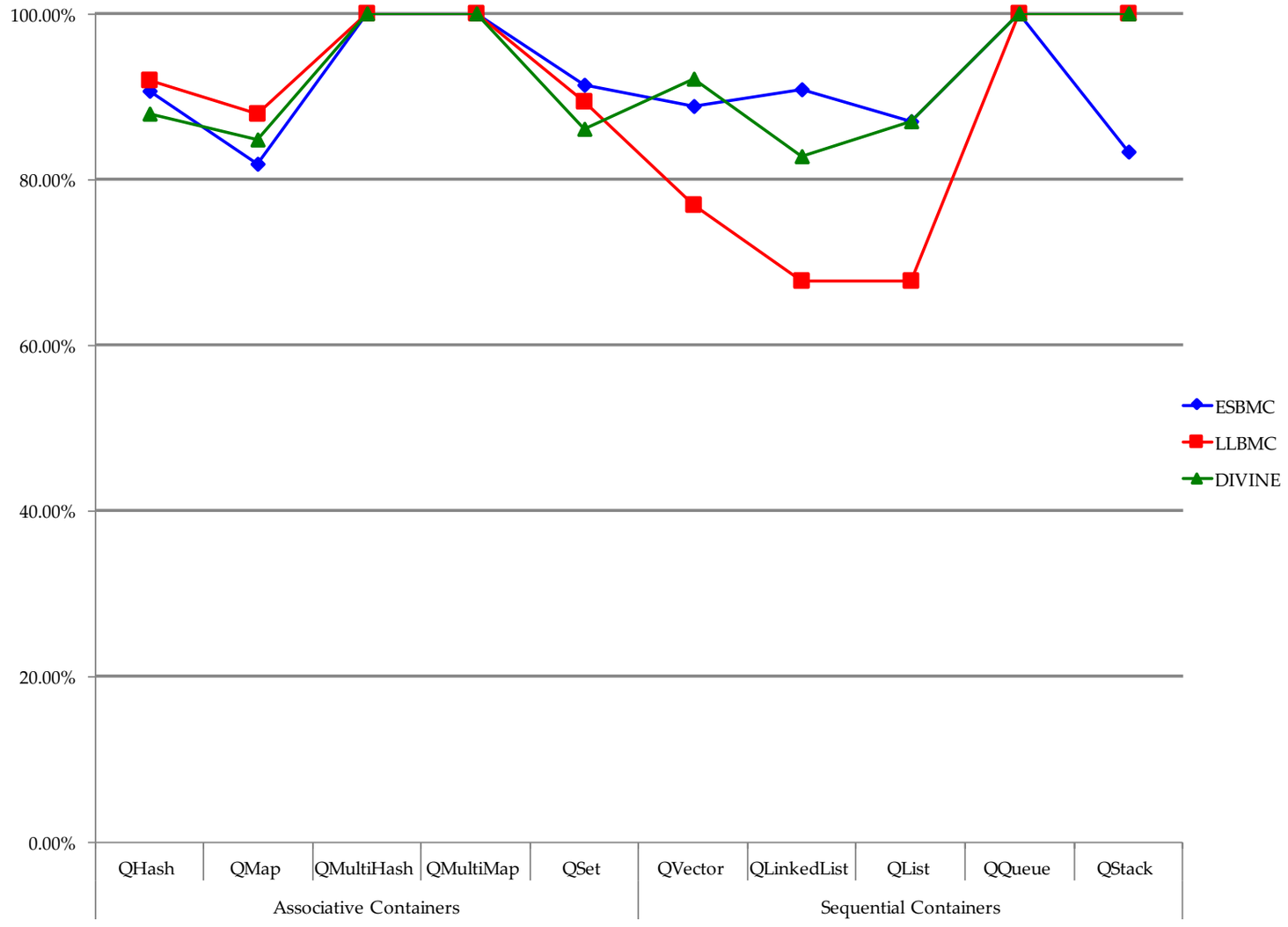
- A **comparison** regarding the **performance** of **LLBMC** and **ESBMC**, which are SMT-based BMC model checkers, and **DIVINE**, which employs explicit-state model checking, was carried out
- LLBMC detected 95% (in **2513.5 seconds**) and DiVinE found 92% (in **14722.4 seconds**) of the existing bugs, overcoming ESBMC that detects 89.2% (in **1760 seconds**)



Comparison Against Model Checkers

- A **comparison** regarding the **performance** of **LLBMC** and **ESBMC**, which are SMT-based BMC model checkers, and **DIVINE**, which employs explicit-state model checking, was carried out

- LLBMC detected **95%** (in 2513.5 seconds) and DiVinE found **92%** (in 14722.4 seconds) of the existing bugs, overcoming ESBMC that detects **89.2%** (in 1760 seconds)



Contributions

QtOM was expanded, in order to include new features from the main Qt modules: Qt GUI and QtCore.

Contributions

- the support for sequential and associative template-based containers

- the integration of QtOM into the verification process of the state-of-the-art C++ verifies DIVINE and LLBMC

- the verification of two Qt-based applications known as *Locomaps* and *GeoMessage*

- As **future work**, the developed QtOM will be extended, in order to fully support the verification of
  i. multi-threaded Qt software
  ii. Qt Event System

“**The main challenge is scalability**: real-world software systems not only include complex control and data structure, but depend on much "context" such as libraries and interfaces to other code, including lower-level systems code. As a result, proving a software system correct requires much more effort, knowledge, training, and ingenuity than writing the software in trial-and-error style.”

–E. M. Clarke et al., Handbook of Model Checking, 201