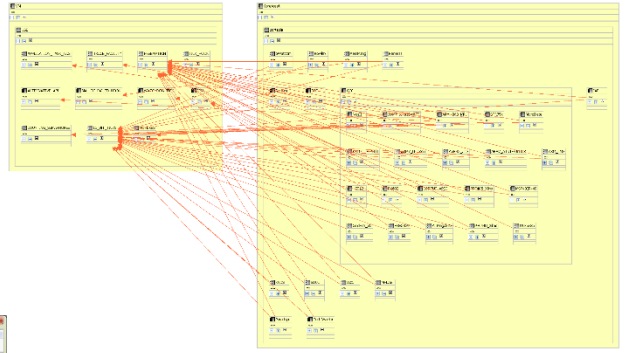
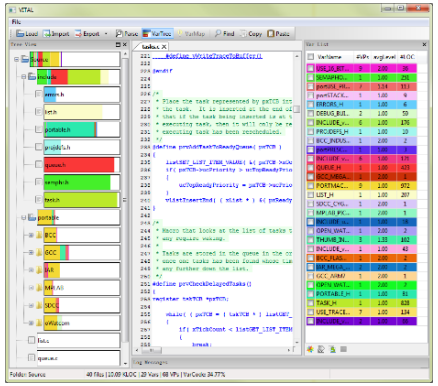
A COMPARATIVE STUDY ON VARIABILITY CODE ANALYSIS TECHNOLOGY

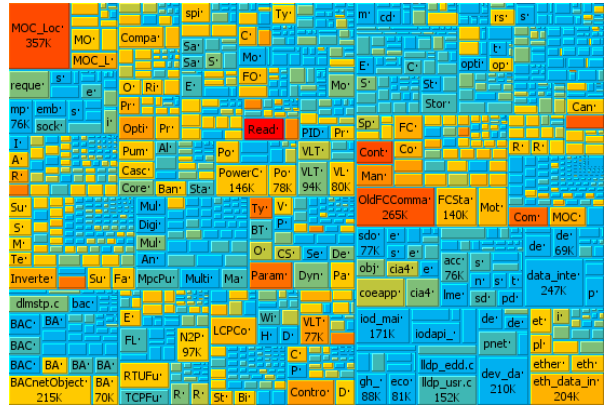
Suparna S Nair, **Martin Becker**, **Vasil Tenev,** 19.10.2020

Context

- The VITAL (**V**ariability **I**mprovemen**T** **A**na**L**ysis) approach and toolset has been developed to support variability management improvement in industrial PLE settings
- Successfully applied in different industrial contexts
- Shortcomings and improvement opportunities identified in the toolset



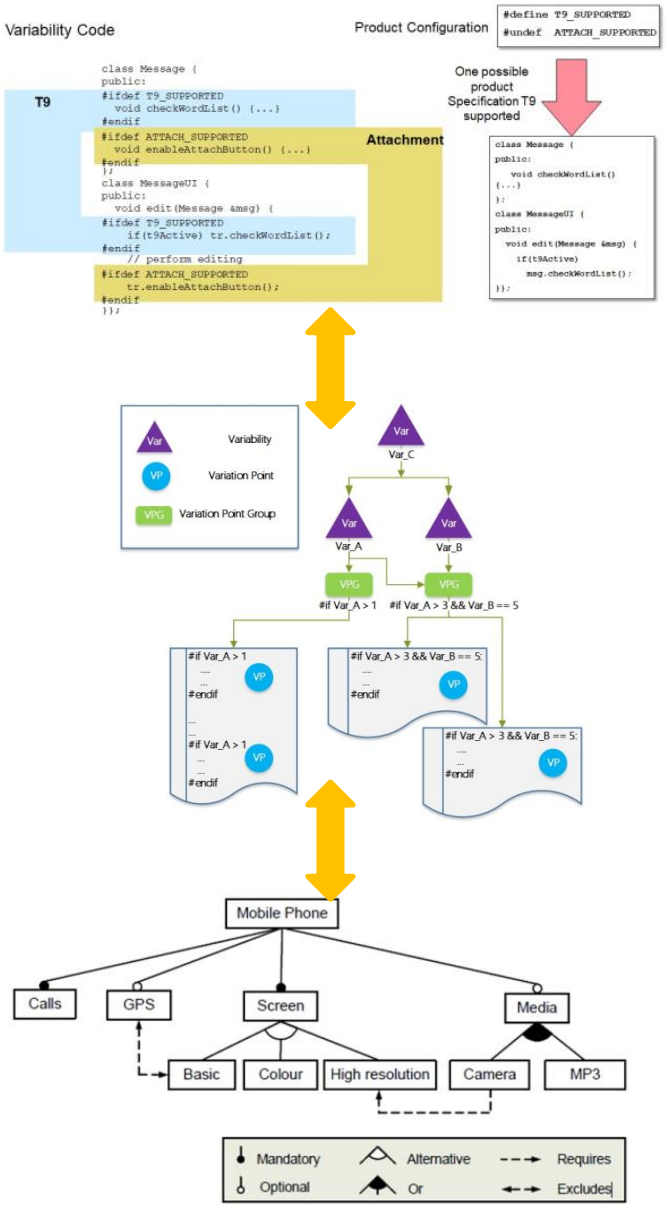




Fraunhofer IESE

Goals

- With a mid-term goal of developing a modular PLE analysis building kit by enhancing the VITAL approach, this study:
  - Explores different frameworks and libraries in academia and industry for variability extraction from software artifacts
  - Reports experiences acquired during the study
  - Compares the features in different tools with respect to improving the VITAL approach
  - Provides recommendations for variability code analysis for different software artifacts
- Focus on variability management on embedded software product lines, with target language as C/C++



Fraunhofer IESE

Investigations

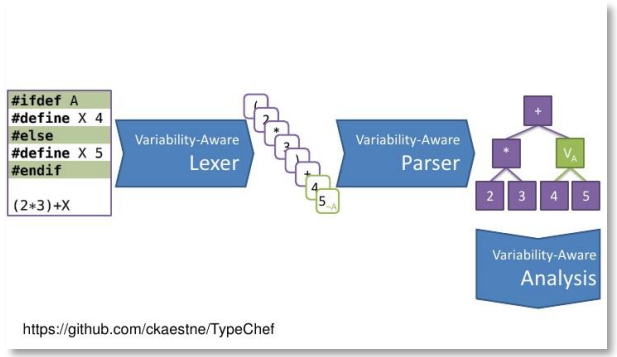
- Approach based on 5 key features of interest with respect to VITAL enhancement:

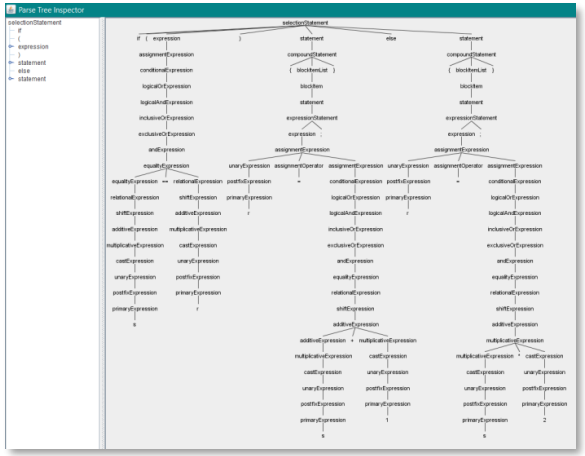
  [Core]: extract basic variability realization information like variabilities, variation points, and presence conditions,
  [Deps]: extract variability inter-dependencies and parent-child relationships, with an aim to reconstruct the feature model,
  [Adv]: extract conditional compilation states and include graphs,
  [Cust]: degree of control the analyst/ developer has in customizing the data objects provided by the respective tool/framework to meet the specific variability analysis requirements,
  [Ext]: possibility to extend the methodology for recovering variability code from generic software artifacts

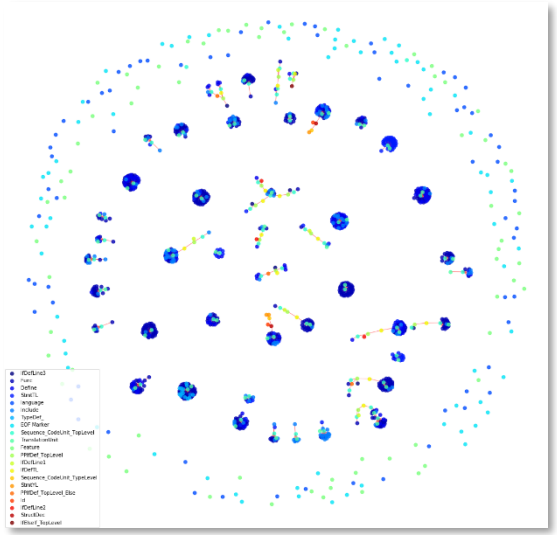
- Frameworks, Parser Generators, Python libraries and C++ Compilers were selected for study

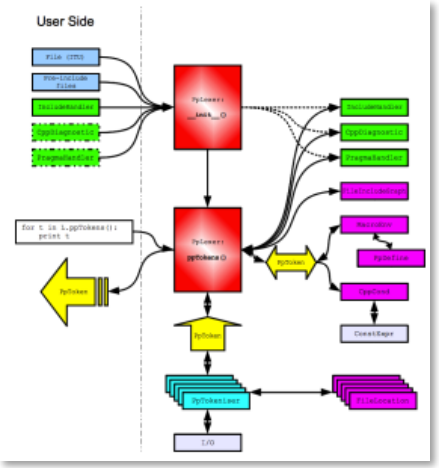
Investigations

- Feasibility studies on:
  - TypeChef
  - ANTLR
  - FeatureHouse
  - REVaMP2 / BUT4Reuse
  - PCPP
  - Clang and LLVM
  - CPIP
- Evaluated the tools against FreeRTOS v10.2.1



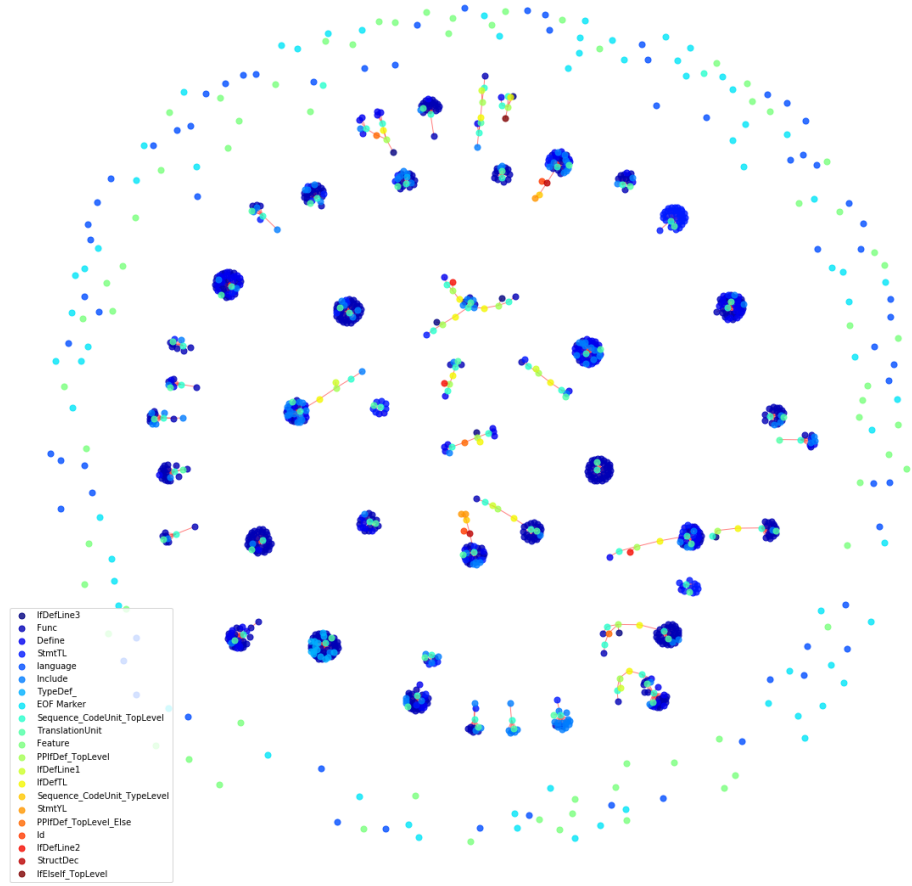






Inferences

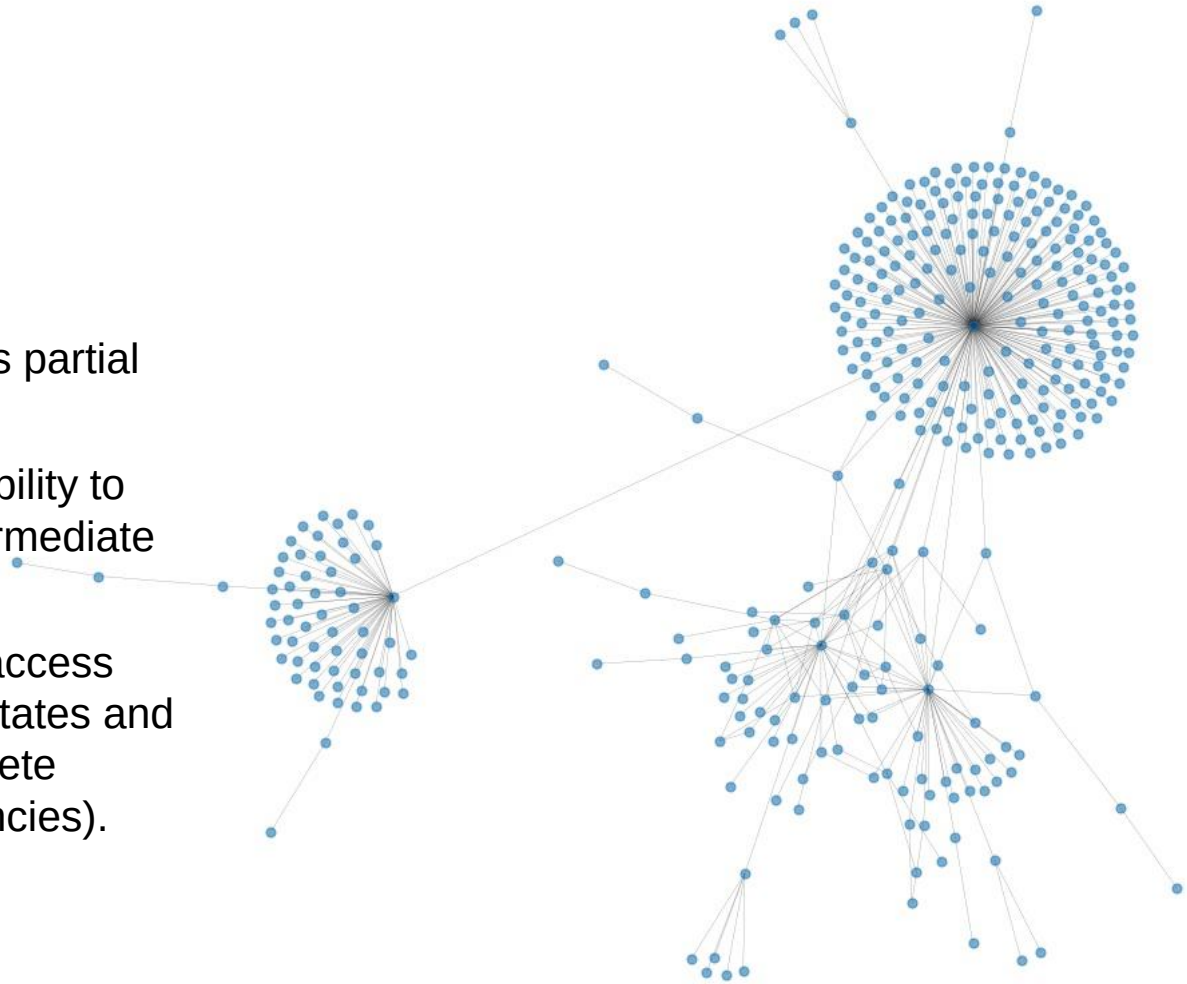
- **TypeChef** – Uses a partial preprocessor developed using ANTLR parser generator. Challenges in preprocessor intricacies
- **ANTLR –** Parser generator which requires specific variability aware grammar file to be developed to extract variability elements and inter-dependencies. Cannot be used *as is*. Requires tailored grammar file.
- **FeatureHouse –** Software composition based on language independent model. Generates Feature Structure Trees based on specific grammar files. Does not support variability code extraction *as is*, requires tailored grammar files. Potential candidate for variabilities realized using conditional execution.



FST generated using FeatureHouse library

Inferences

- **PCPP –** C99 preprocessor written in Python. Ability to access partial preprocessing information. Flexibility is limited
- **Clang & LLVM** – C compiler with Python bindings. Limited ability to parse preprocessor information. *pp-trace* utility provided intermediate preprocessing results.
- **CPIP** – C99 preprocessor implemented in Python. Ability to access preprocessing information including conditional compilation states and include graphs. Additional implementation required for complete variability extraction support (including hierarchical dependencies).



Hierarchical dependencies extracted using modified CPIP library

Comparison

- Following comparison is against the desirable properties derived for the frameworks with respect to VITAL enhancements
  - ++ : fully supported
  - + : supported with extensions
  - - : partially supported with extensions
  - -- : not supported at all

Tool	[Core]	[Deps]	[Adv]	[Cust]	[Extn]
TypeChef	++	+	- -	-	- -
ANTLR	-	-	- -	++	++
FeatureHouse	+	+	- -	-	++
BUT4Reuse	+	+	- -	-	++
PCPP	+	+	- -	+	- -
Clang/LLVM	+	- -	- -	- -	- -
pp-trace	+	+	- -	+	- -
CPIP	+	+	++	++	- -

Conclusion

- Following are the recommendations based on our inferences:
  - For parsing variability code realization using conditional compilation in C/C++ - standard C preprocessor written in Python which provides ability to control the data to the desired form (CPIP)
  - For parsing variability code realization using conditional execution (e.g., C, Java etc.) – a library that provides an AST, which can identify conditional statements and hierarchical dependencies (FeatureHouse, ANTLR)
  - For parsing any other software artifact like requirements, UML diagrams, *.ini – library that allows users to write the grammar for that specific artifact (FeatureHouse, ANTLR)

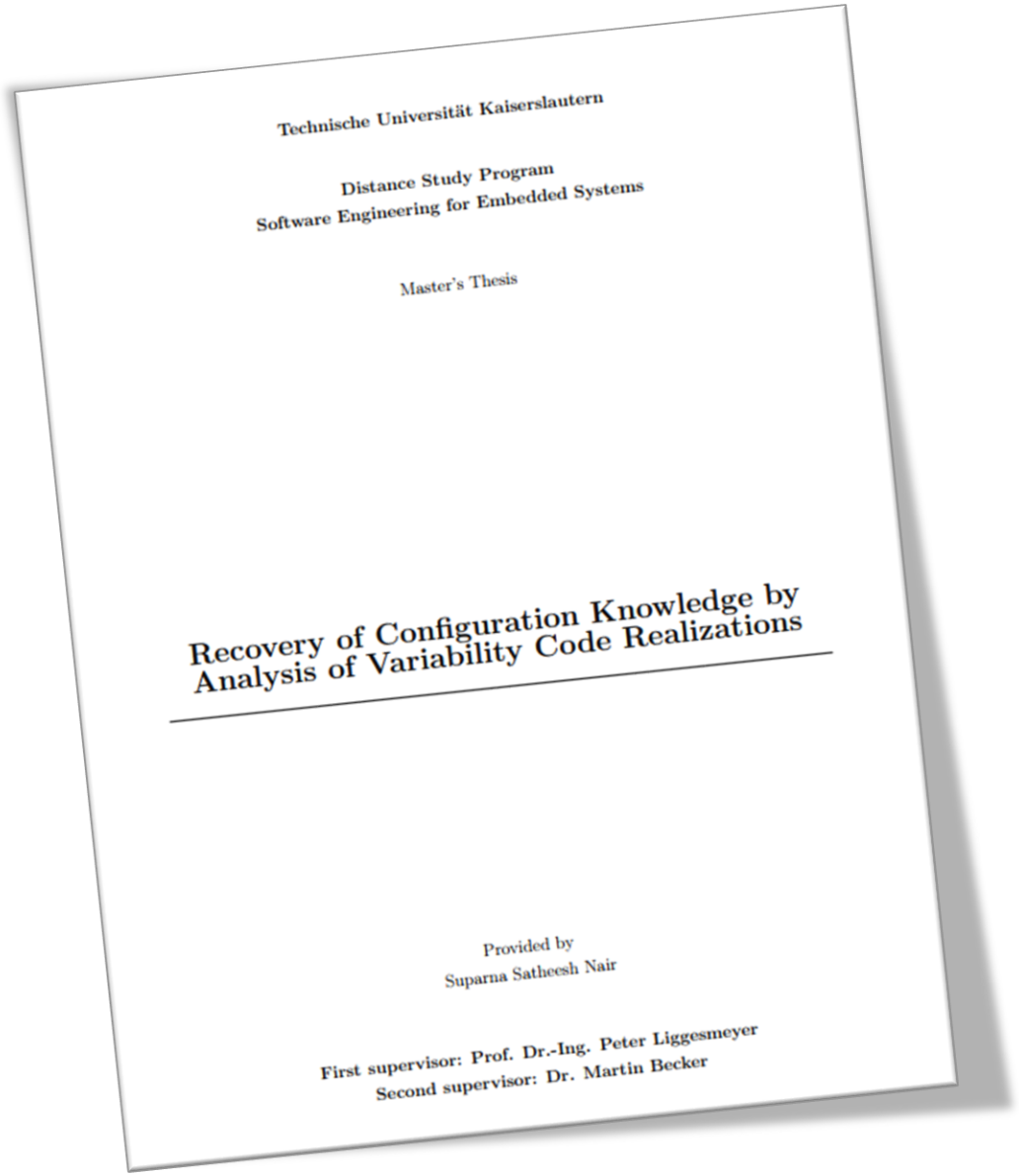
Further Reading

- “Recovery of Configuration Knowledge by Analysis of Variability Code Realization”

  (Master’s Thesis)

  Suparna S Nair

  Email: suparnasnair@gmail.com



Technische Universität Kaiserslautern

Distance Study Program
Software Engineering for Embedded Systems

Master’s Thesis

Recovery of Configuration Knowledge by Analysis of Variability Code Realizations

Provided by
Suparna Satheesh Nair

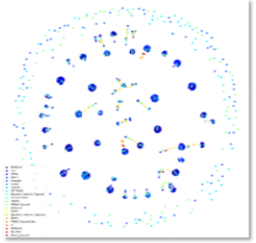
First supervisor: Prof. Dr.-Ing. Peter Liggesmeyer
Second supervisor: Dr. Martin Becker

Thank You

Any Questions?

Investigations

- Feasibility studies on:
  - TypeChef
  - ANTLR
  - FeatureHouse
  - REVaMP2 / BUT4Reuse
  - PCPP
  - Clang and LLVM
  - CPIP
- Evaluated the tools against FreeRTOS v10.2.1



ANTLR

REVaMP²

Feature House

clang

Fraunhofer IESE

Comparison

- Following comparison is against the desirable properties derived for the frameworks with respect to VITAL enhancements
  - ++ : fully supported
  - + : supported with extensions
  - - : partially supported with extensions
  - -- : not supported at all

Tool	[Core]	[Deps]	[Adv]	[Cust]	[Extn]
TypeChef	++	+	- -	-	- -
ANTLR	-	-	- -	++	++
FeatureHouse	+	+	- -	-	++
BUT4Reuse	+	+	- -	-	++
PCPP	+	+	- -	+	- -
Clang/LLVM	+	- -	- -	- -	- -
pp-trace	+	+	- -	+	- -
CPIP	+	+	++	++	- -

Fraunhofer IESE