GÖTEBORGS UNIVERSITET

SOFTWARE COMPLEXITY METRICS IN GENERAL AND IN THE CONTEXT OF ISO 26262 SOFTWARE VERIFICATION REQUIREMENTS

MIROSLAW STARON, ASSOCIATE PROFESSOR, SOFTWARE ENGINEERING
CHALMERS | UNIVERSITY OF GOTHENBURG

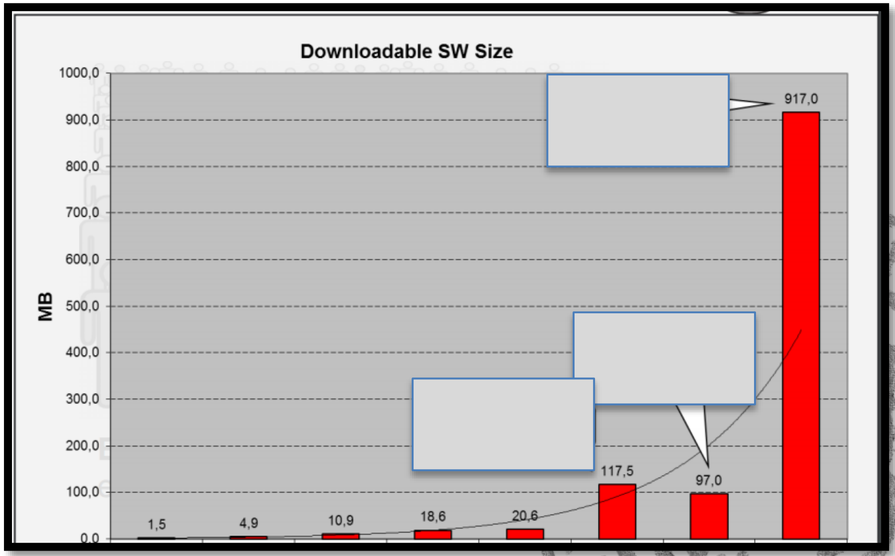
VINNOVA

VOLVO

Software Center

Motivation for our research – safe cars

- The number of functions that are software steered grows as well
  - Autonomous driving >> 50 pure software functions
- Exponential growth of vehicle's software size
  - The number of ECUs grows exponentially (2 ECUs in 1970 to over 130 in 2016)
  - The amount of software grows exponentially
- We face new challenges
  - How to verify and validate all the software?
  - How to increase sw dev. speed if the sw. complexity grows?



Source: www.software-center.se,

Outline of the talk

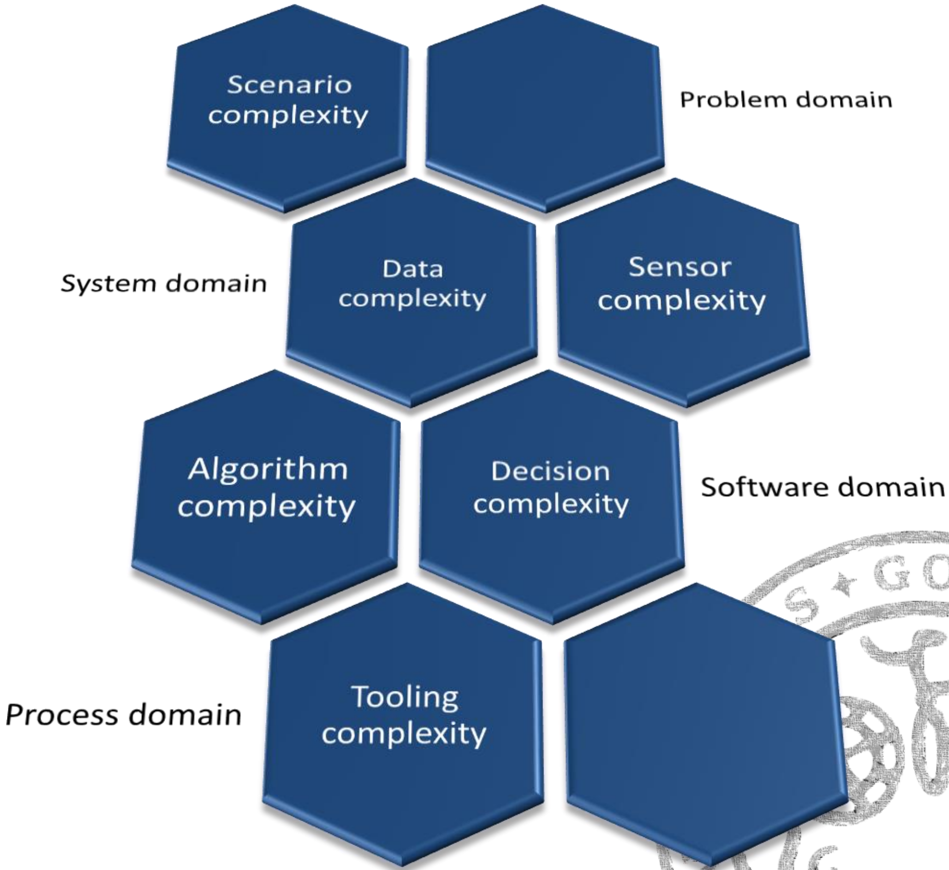
- Software complexity
  - Basic concepts
  - New scenarios for software use
  - New data sets available
- Overview of ISO 26262
  - Basic concepts
  - Software in ISO 26262
  - Software verification requirements
- Challenges for verifying and validating
  - ISO 26262 verification requirements linked to software verification techniques
  - Combining techniques to increase the level of verification and validation

SOFTWARE COMPLEXITY

Complexity in the software of modern cars

- Software complexity
  - The degree of connectivity between entities in a program

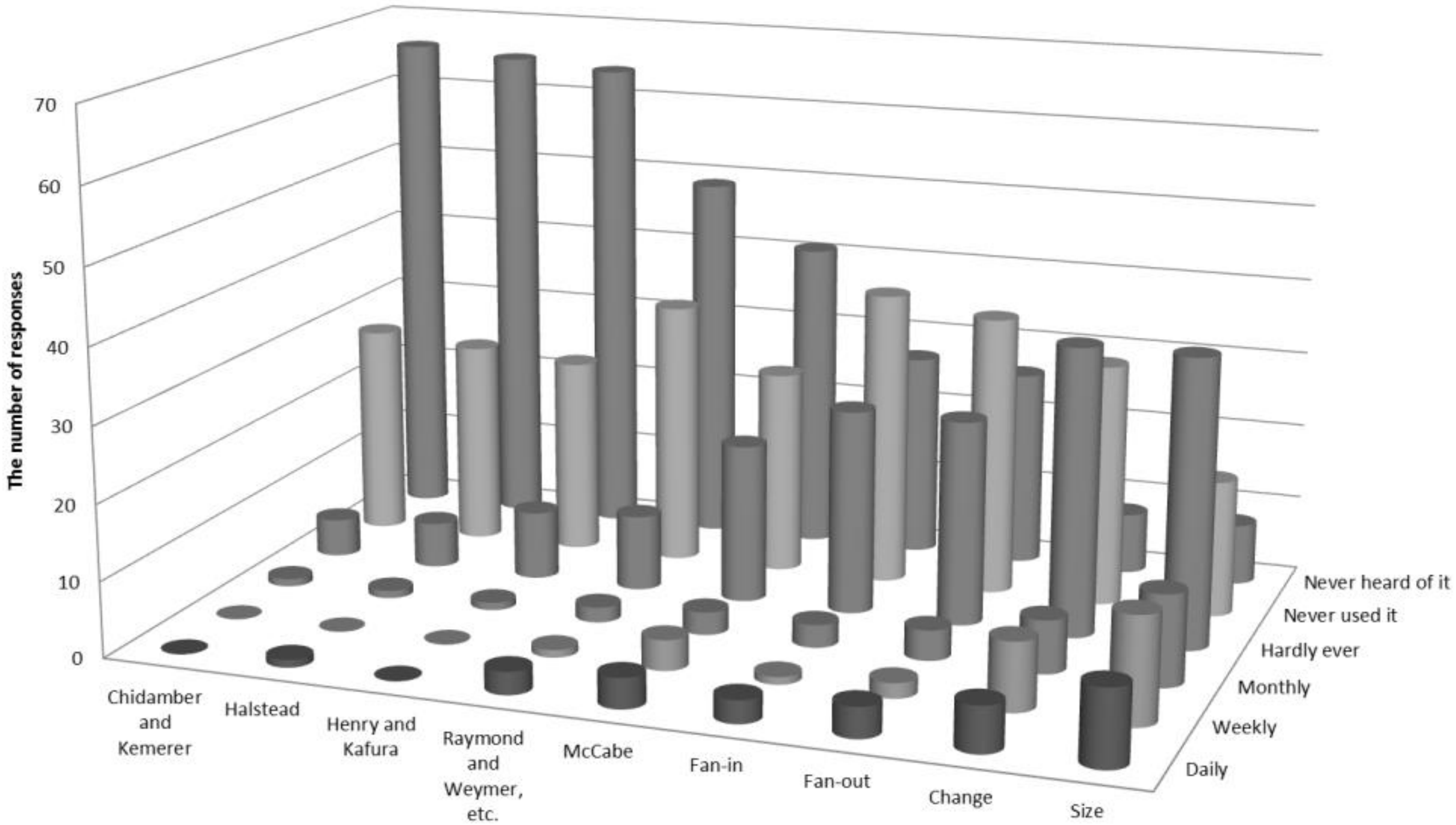
- Metrics (examples)
  - Cyclomatic complexity metric (McCabe)
  - Software science metrics (Halstead)
  - Software Structure Metrics (Henry and Kafura)
  - Metrics Suite for Object Oriented Design (Chidamber and Kamerer)
  - Branching complexity (Sneed)
  - Data access complexity (Card)
  - Data complexity (Chapin)
  - Data flow complexity (Elshof)
  - Decisional complexity (McClure)



Some of the most common complexity metrics, cont.

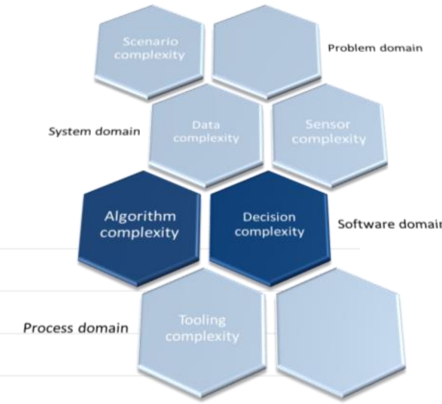
Name of the Measure	Description
McCabe's cyclomatic complexity (1976**)**	The number of linearly independent paths in the control flow graph of code. This can be calculated by counting the number of control statements in the code
Halstead measures (1977)	7 measures completely based on number of operators and operands
Fan-out (Henry and Kafura 1981)	Number of unique invocations found in a given function
Fan-in (Henry and Kafura 1981)	Number of calls of a given function elsewhere in the code
Coupling measures of Henry and Kafura (1981)	Based on size, fan-in, and fan-out
Chidamber and Kemerer OO measures (1994)	Inheritance level and several size measures for class
Size measures	Lines of code, number of statements, etc.
Readability measures, e. g. Tenny (1988), Buse and Weimer (2010)	Line length, indentations, length of identifiers, etc.

How often are they used in industry?

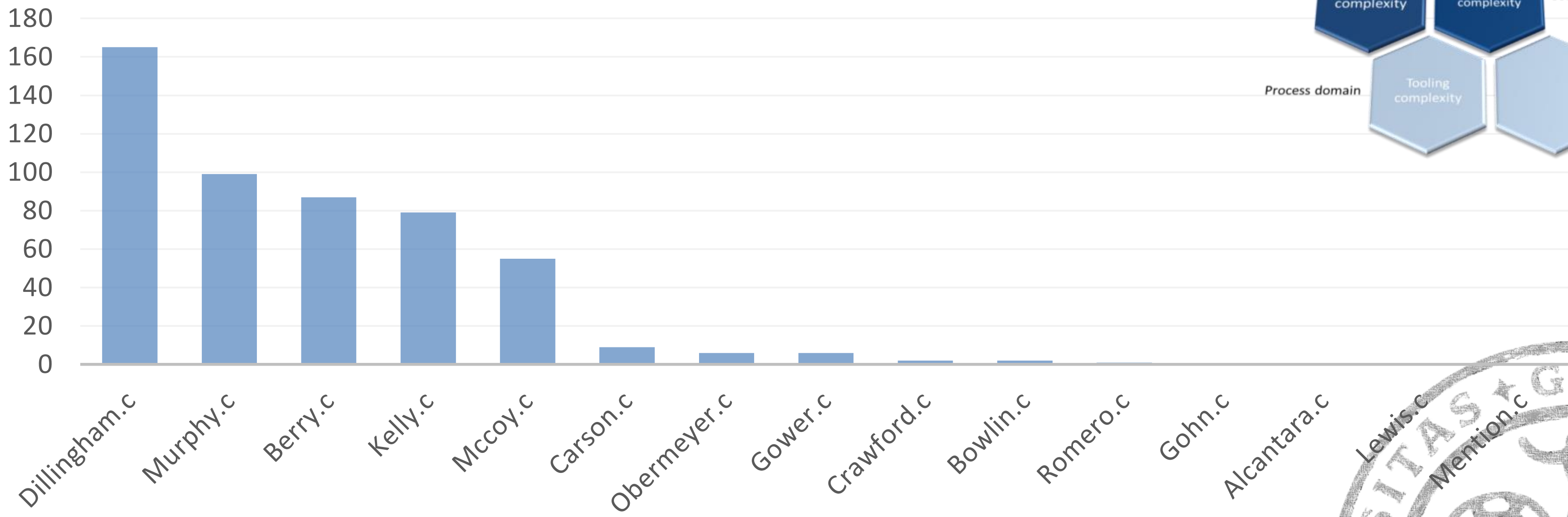


Survey done by V. Antinyan, M. Staron, A. Sandberg, J. Hansson, in submission

Complexity of decision algorithms in practice (automotive)



# of independent data paths



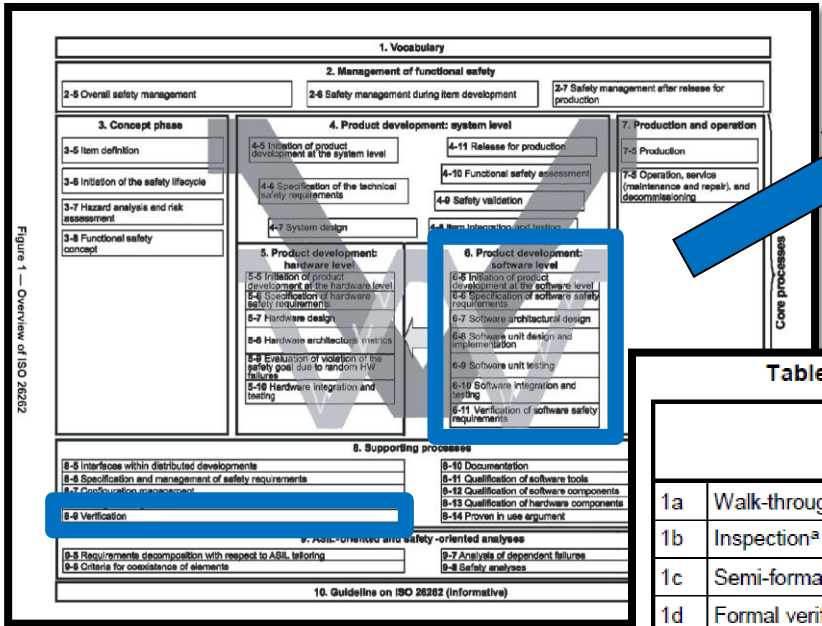
Altinger, H., Siegl, S., Dajsuren, Y., & Wotawa, F. (2015, May). A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), pp. 494-497, IEEE Computer Society Press.

Implications

- One control path => at least one test case
  - 511 for each path
  - test all combinations (theoretical) – anything between 511 and $1.5 * 10^{22}$
  - In practice >> 1 trillion 10^{12} test cases is required due to co-dependency of test cases
- One control path => at least one fault injection
  - 511 injections
- One test case => one mutation
  - 511 – $1.5 * 10^{22}$ mutations

OVERVIEW OF ISO 26262
ROAD VEHICLES — FUNCTIONAL SAFETY

ISO 26262 – Functional Safety – Road vehicles



ISO 26262
– Chapter 6: Product development: software level
- Chapter 8, clause 9: Verification

Table 9 — Methods for the verification of software unit design and implementation

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Source of figure: ISO, C. (2011). "26262, Road vehicles–Functional safety." International Standard ISO/FDIS 26262.

Software complexity in ISO 26262

Chapter 6

- Data complexity
  - Data structures, classes, packets

Table 4 — Mechanisms for error detection at the software architectural level

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check[a]	+	+	+	++
1c	Detection of data errors[b]	+	+	+	+
1d	External monitoring facility[c]	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

[a] Plausibility checks can include using a reference model of the desired behaviour, assertion checks, or comparing signals from different sources.

[b] Types of methods that may be used to detect data errors include error detecting codes and multiple data storage.

[c] An external monitoring facility can be, for example, an ASIC or another software element performing a watchdog function.

- Code/control flow complexity
  - Algorithms, state machines, block diagrams

Table 6 — Methods for the verification of the software architectural design

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Walk-through of the design[a]	++	+	o	o
1b	Inspection of the design[a]	+	++	++	++
1c	Simulation of dynamic parts of the design[b]	+	+	+	++
1d	Prototype generation	o	o	+	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis[c]	+	+	++	++
1g	Data flow analysis[c]	+	+	++	++

[a] In the case of model-based development these methods can be applied to the model.

[b] Method 1c requires the usage of executable models for the dynamic parts of the software architecture.

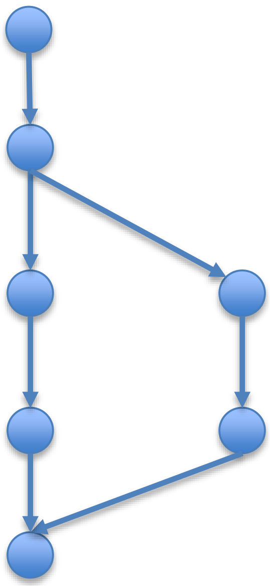
[c] Control and data flow analysis may be limited to safety-related components and their interfaces.

GÖTEBORGS UNIVERSITET

Overview of V&V requirements from ISO 26262

Software design and implementation

- Walkthrough
- Inspection
- Semi-formal verification
- **Control-flow analysis**
  - **McCabe cyclomatic complexity**
- Data-flow analysis
- Static code analysis
- Semantic code analysis



```
void main(int a) {

   if (a == 0) {


   } else {


   }

}
```

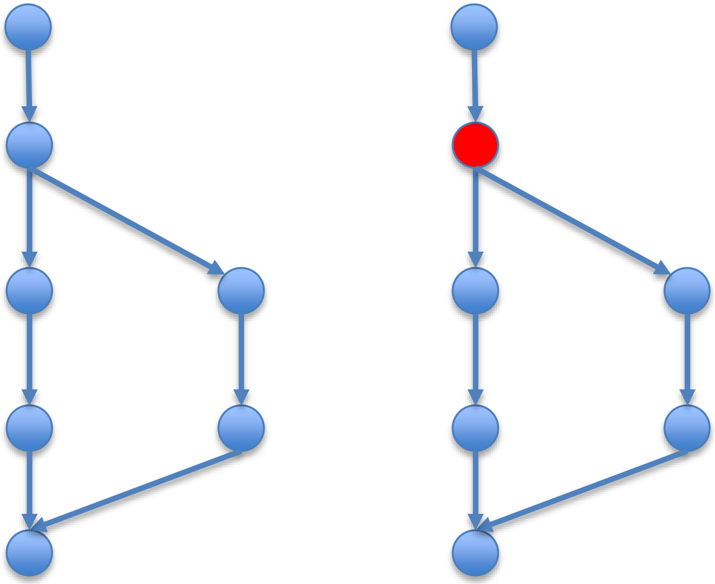
COMBINING FAULT INJECTION AND MUTATION TESTING

Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M., & Törner, F. (2013, July). Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model based Development. In ICSOFT (pp. 251-257).

Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M., & Törner, F. (2014). Early Verification and Validation According to ISO 26262 by Combining Fault Injection and Mutation Testing. In Software Technologies (pp. 164-179). Springer Berlin Heidelberg.

Fault injection
Principles of mutation testing

- Exchange a piece of code into a different onw
- Observe whether the change results in test cases failures



Mutation Testing

Principles of mutation testing

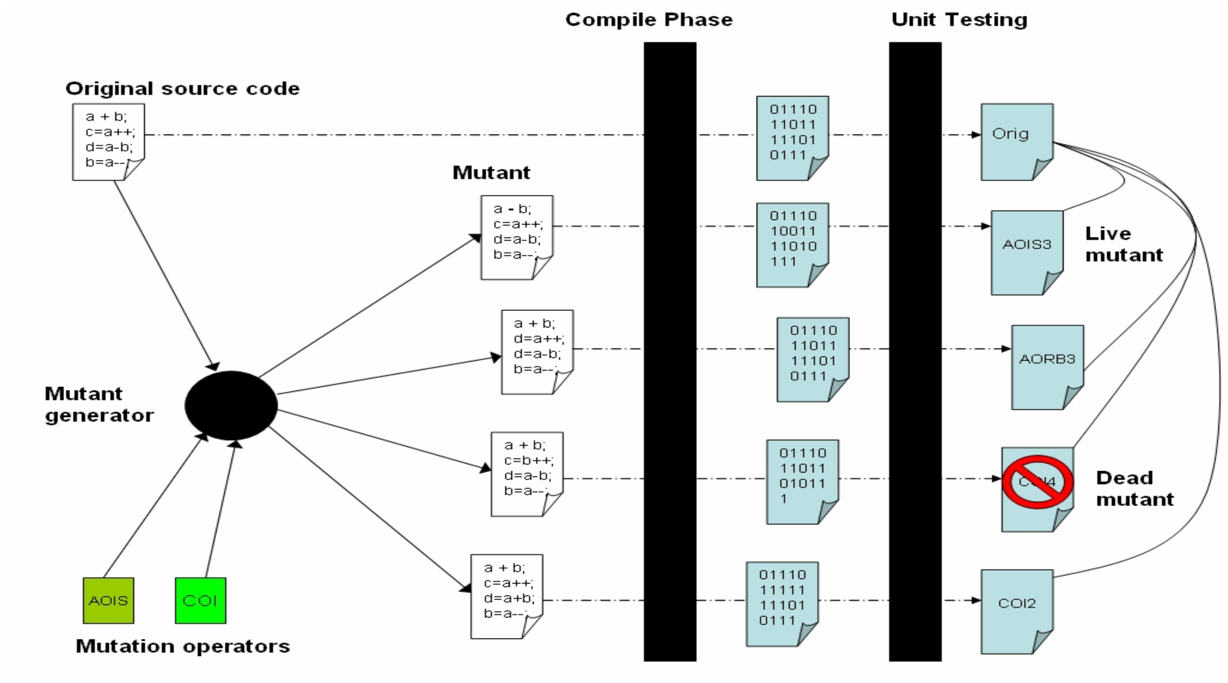
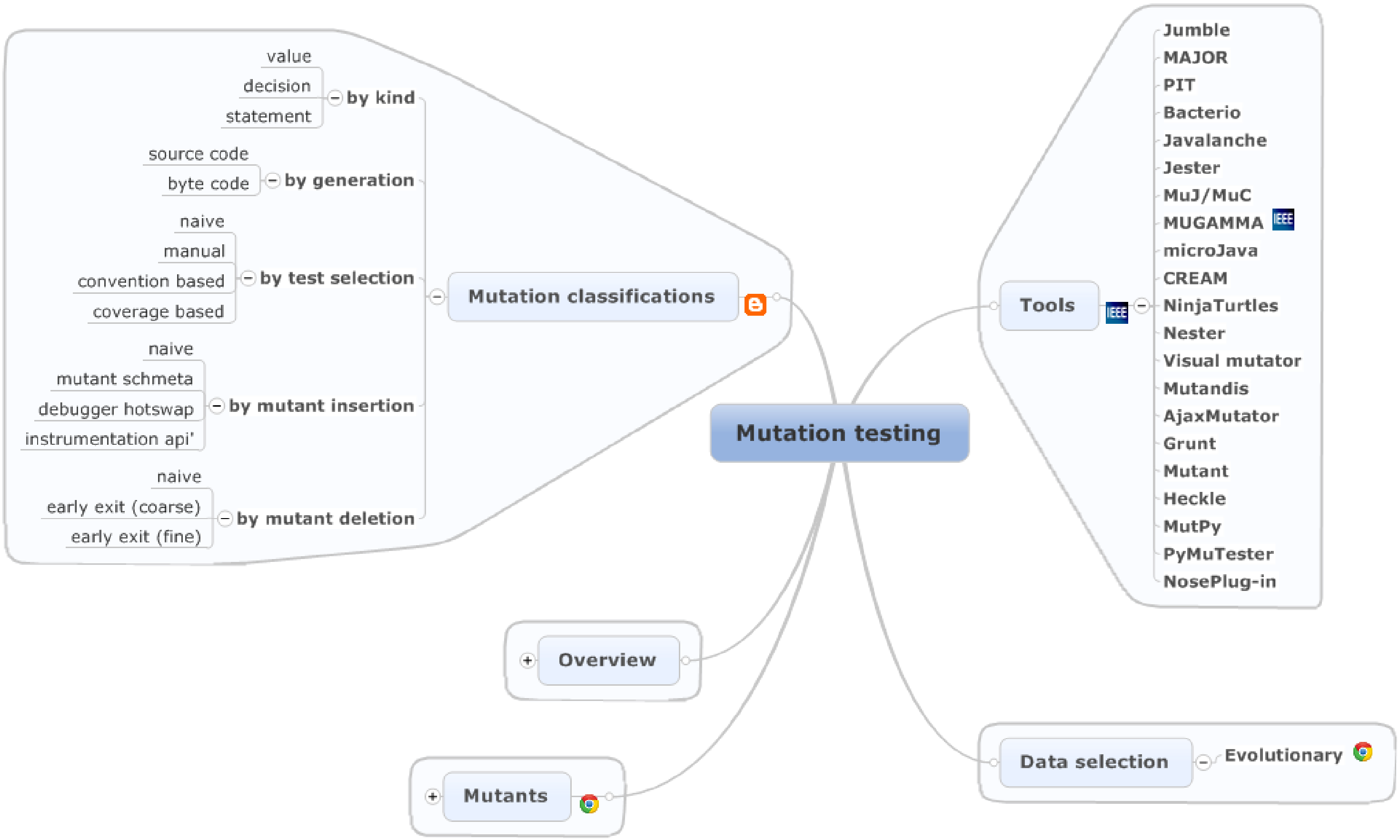


Figure: http://muclipse.sourceforge.net

Mutation testing
Overview of major techniques/tools



Summary

- Two take-aways
  - As the number of software functions (usage scenarios) increase in cars => complexity of the software increases
  - Testing for all possible execution paths becomes almost impossible => we need to test for subsets and understand how good our testing is

- Further directions
  - Software reliability growth modelling and latent defect inflow prediction
  - Combining formal verification with software testing
  - Using machine learning/search-based software testing to find the best testing combination for a given software functionality

UNIVERSITAS HOBURGENSIS

Overview of V&V requirements from ISO 26262

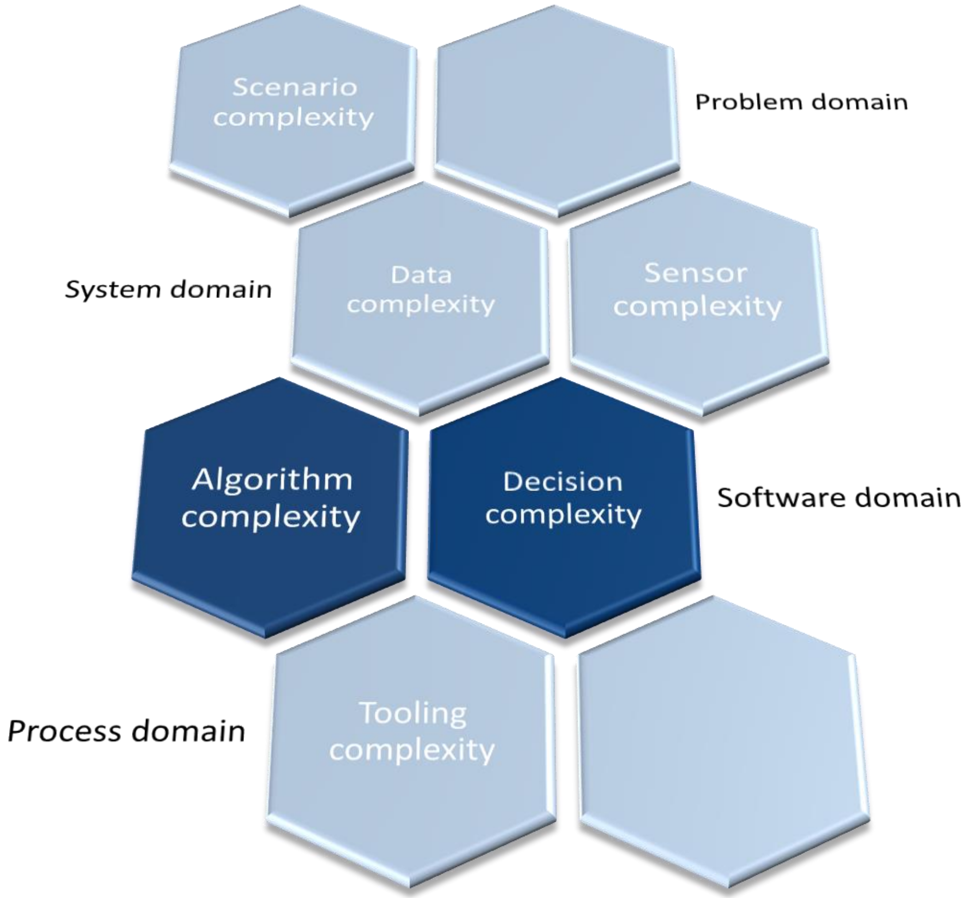
Software design and implementation

- Walkthrough
- Inspection
- Semi-formal verification
- Control-flow analysis
- Data-flow analysis
- **Static code analysis**
- Semantic code analysis

file	class	function	McCabe 01	McCabe 02	Delta
			-	51	51
			-	47	47
			57	90	33
			-	30	30
			-	28	28
			-	27	27
			-	27	27
			-	26	26
			-	26	26
			-	25	25
			-	25	25
			-	23	23
			-	23	23
			142	160	18
			31	47	16
			9	2	-7
			16	8	-8
			12	4	-8
			28	13	-15

Benefits of combining

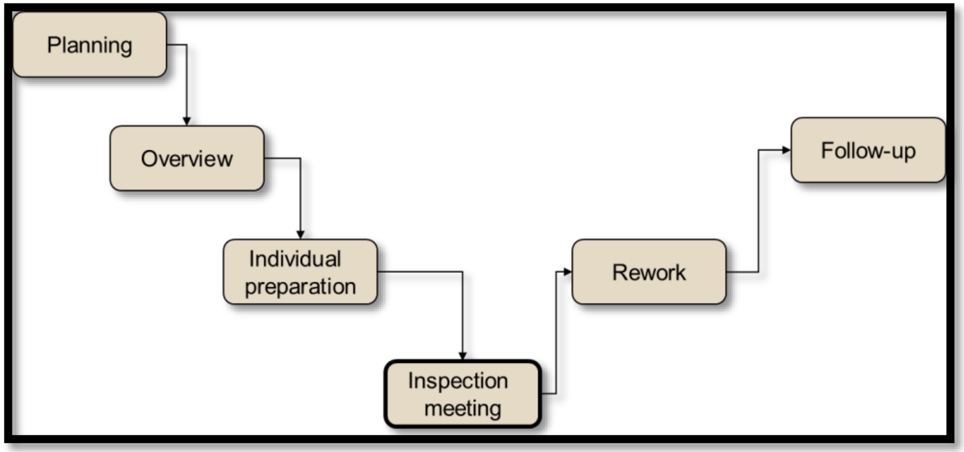
- Assessment of the quailty of software
  - We know if the software can handle problems with failures during the operation
- Assessment of the quality of the ”process” – or testing
  - We know if the test cases test the faulty programs
  - We know if we can trust the testing
- Where do we go from here
  - Software reliability assessment


Scenario complexity
Problem domain
System domain
Data complexity
Sensor complexity
Algorithm complexity
Decision complexity
Software domain
Process domain
Tooling complexity

GÖTEBORGS UNIVERSITET

Overview of V&V requirements from ISO 26262
Software design and implementation

- Walkthrough
- **Inspection**
- Semi-formal verification
- Control-flow analysis
- Data-flow analysis
- Static code analysis
- Semantic code analysis



- Efficiency
  - 125 source statement/hour during individual preparation
  - 90-125 statements/hour can be inspected during inspection meeting
- Inspection is therefore an expensive process
  - Inspecting 500 lines costs about 40 man/hours effort – about €2000