

Results of a Comparative Study of Code Coverage Tools in Computer Vision

Iulia Nica, **Franz Wotawa**, Gerhard Jakob, and Kathrin Juhart

TU Graz, Institute for Software Technology *and* Joanneum Research

Index

- Motivation
- Tools Selection
- Case Study
- Comparison of Coverage Results
- Conclusions

Motivation I

- The high **quality of computer vision** (CV) software has a great impact on the usability of the overall CV systems
- **Standardized quality assurance** methods, metrics and tools can quickly improve the overall process
- **Initial goal:** identify a **coverage-based testing tool**, capable to quickly find deficiencies in the available test suites.

Motivation II

- Highly varying results reported by different coverage tools for the example application
 - ➡ Which might be the reasons for this variation?
 - ➡ Which of the computed values better reflect the real quality of the code?

Tools Selection I

- Target programming language: C/C++
- 9 candidate tools:





















		Coverage			
Tool	Access	Line	Function	Branch	More
COVTOOL	free	Y			
gcov	free	Y			
Testwell CTC++	charge	Y	Y	Y	Y
CoverageMeter v1.4	charge	Y	?	?	?
BullseyeCoverage	charge		Y	Y	Y
C++ Coverage Validator	charge	Y	Y	Y	Y
Squish Coco	charge	Y	Y	Y	Y
C++ Test Coverage Tool	charge	?		Y	?
OpenCPPCoverage	free	Y			

Tools Selection II
















C++ Coverage Validator

color	59,41%
colorexception.hpp	0,00%
colormap.hpp	0,00%
colormap.hpp	73,30%
colormapreader.cpp	55,38%
colormapreader.hpp	0,00%
colormapwriter.cpp	0,00%
colormapwriter.hpp	89,61%
colornormalization.hpp	88,00%
rgbcollection.hpp	60,00%
rgbcollection.hpp	100,00%
rgbtohsv.hpp	0,00%

BullseyeCoverage

Name	Function coverage	Uncovered functions
DevelopIulia/Source/dibgen/Libraries/Color/	81% 	16 
ColorMap.hpp	33% 	2 
ColorMapWriter.cpp	50% 	2 
ColorException.hpp	50% 	1 
RGBtoHSV.hpp	62% 	3 
ColorMapReader.cpp	75% 	2 
ColorNormalization.hpp	78% 	3 
ColorMap.hpp	88% 	3 
ColorMapReader.hpp	100% 	0
ColorMapWriter.hpp	100% 	0
RGBColor.hpp	100% 	0
RGBColor.hpp	100% 	0

Testwell CTC++

50 % -	(1/2)		ColorException.hpp
100 % -	(0/0)		ColorMap.cpp
0 % -	(0/3)		ColorMap.hpp
88 % -	(22/25)		ColorMap.hpp
2 % -	(6/339)		ColorMapReader.cpp
25 % -	(1/4)		ColorMapReader.hpp
1 % -	(2/335)		ColorMapWriter.cpp
0 % -	(0/3)		ColorMapWriter.hpp
0 % -	(0/14)		ColorNormalization.hpp
100 % -	(0/0)		RGBColor.cpp
75 % -	(3/4)		RGBColor.hpp
100 % -	(12/12)		RGBColor.hpp
100 % -	(0/0)		RGBtoHSV.cpp
0 % -	(0/8)		RGBtoHSV.hpp
6 % -	(47/749)		DIRECTORY OVERALL

Squish Coco

Source	Coverage
color	81.176% (69/85)
colorexception.hpp	50.000% (1/2)
colormap.hpp	33.333% (1/3)
colormap.hpp	87.500% (21/24)
colormapreader.cpp	75.000% (6/8)
colormapreader.hpp	100.000% (4/4)
colormapwriter.cpp	50.000% (2/4)
colormapwriter.hpp	100.000% (3/3)
colornormalization.hpp	78.571% (11/14)
rgbcollection.hpp	100.000% (4/4)
rgbcollection.hpp	100.000% (12/12)
rgbtohsv.hpp	57.143% (4/7)

Case Study I

- *Dibgen* - a collection of basic C++ libraries implemented by JOANNEUM RESEARCH (JR).
- The libraries cover basic, mostly matrix based mathematical operations, color handling and evaluation, generic parameter storage, progress information handling, different types of basic file IO methods often used in CV, and value-to-string conversion (and back-conversion).
- The libraries are implemented using template-heavy C++ code

Case Study II

- For the experiments we used the same unit test suites and the same configuration.
- We have fully automated the tests running and coverage measurement process.
- Execution time for the defined unit tests:

Program	Execution Time
<i>Non-instrumented program</i>	<i>68,44 sec</i>
C++ Coverage Validator instrumented program	475,68 sec
CTC++ instrumented program	74,16 sec
Bullseye instrumented program	68,97 sec
Squish Coco instrumented program	70,81 sec

Comparison of Coverage Results I

- Overall DIBGEN Coverage Results

	C++ Coverage Validator	Testwell CTC++	BullseyeCoverage	Squish Coco
BC%	31,27%	9%	40%	45,30%
FC%	39,86%	8%	52%	51,95%

➡ How does each tool compute the coverage?

- I. Analyze the exact definitions for function and branch coverage
- II. Compare the instrumented files

Comparison of Coverage Results II

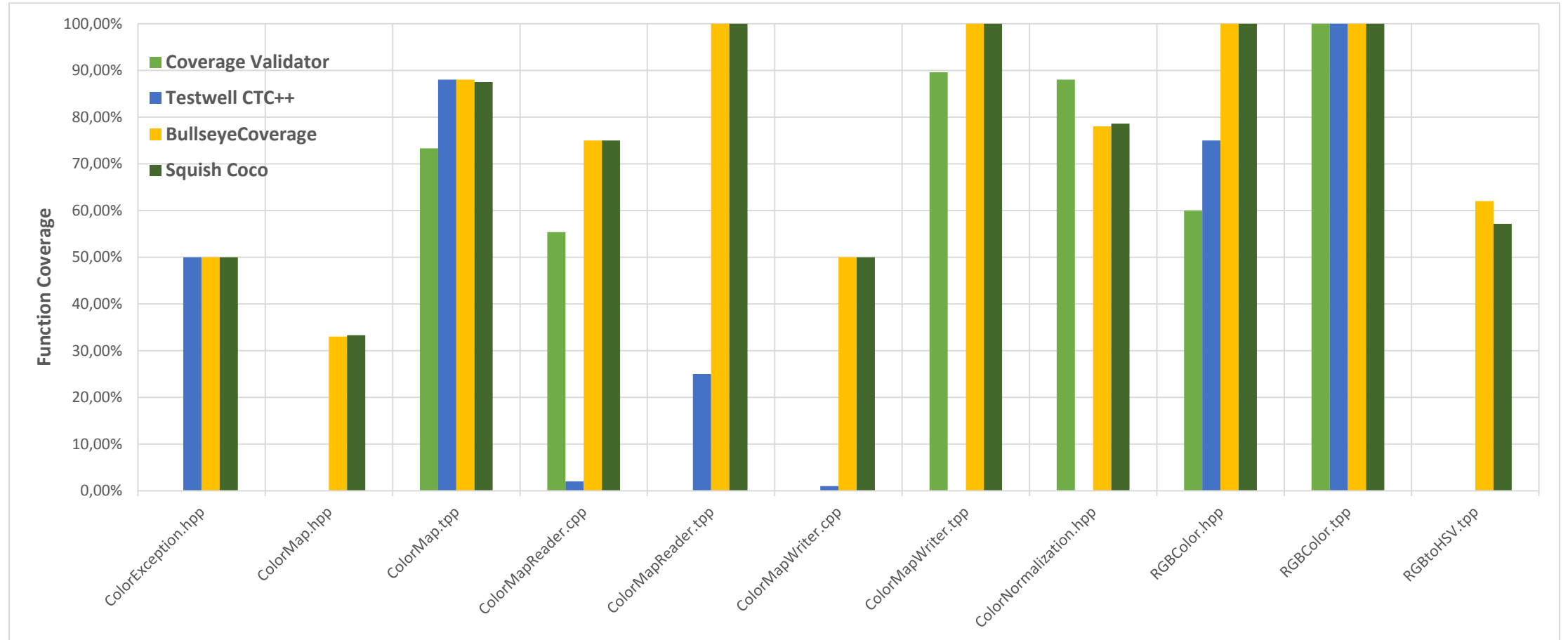
- *Function coverage* (generally accepted definition): *a function is covered if the function is entered.* (Testwell CTC++, BullseyeCoverage, Squish Coco)
- *Function coverage* (C++ Coverage Validator): *"focuses on line coverage at the function level".*
- *Branch coverage: reports whether Boolean expressions tested in control structures evaluate to both true and false.* (all the tools)
 - + *coverage of switch statement cases and unconditional control.* (Testwell CTC++)
 - + *coverage of switch statement cases, exception handlers, and all points of entry and exit.* (BullseyeCoverage)

Comparison of Coverage Results III

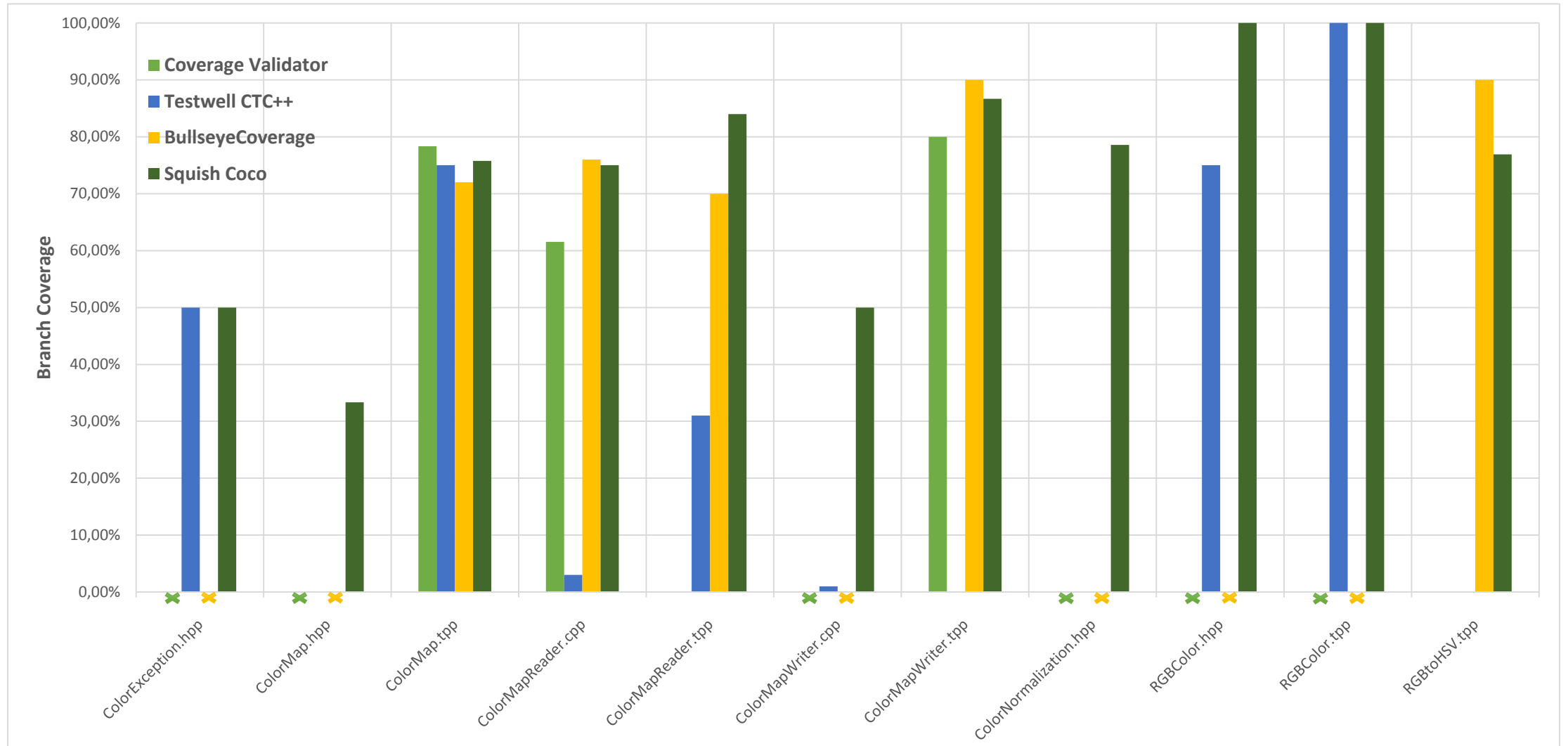
- Coverage results per Library (Function Coverage) computed with all the four tools

	C++ Coverage Validator	Testwell CTC++	BullseyeCoverage	Squish Coco
Color	59,41%	6%	81%	81,17%
Exception	32,50	36%	45%	45,45%
Fileio	16,81%	26%	60%	56,52%
Internationalisation	77,67%	95%	94%	94,44%
Math	74,22%	3%	47%	47,08%
ModuleInterface	23,49%	4%	37%	35,68%
ParameterPool	36,13%	6%	72%	69,17%
ParameterPoolDocumentation	N.A.	0%	0%	N.A.
ProgramOptions	0%	0%	50%	0%
Progress	14,30%	9%	63%	68,46%
ResultDataPool	8,60%	2%	37%	34,69%
Serialization	11,22%	6%	86%	86,20%
Strings	49,46%	37%	65%	64,66%
Types	51,87%	1%	22%	22,88%
UserDataBase	10,45%	80%	86%	86,20%
Utilities	0%	7%	26%	30,76%

Comparison of Coverage Results IV



Comparison of Coverage Results V



Comparison of Coverage Results VI

File	C++ Coverage Validator						Testwell CTC++						BullseyeCoverage						Squish Coco					
	#F	#FC	FC%	#B	#BC	BC%	#F	#FC	FC%	#B	#BC	BC%	#F	#FC	FC%	#B	#BC	BC%	#F	#FC	FC%	#B	#BC	BC%
ColorException.hpp	(1)	(0)	0,00				2	1	50,00	4	2	50,00	(2)	(1)	50,00				2	1	50,00	2	1	50,00
ColorMap.cpp							0	0	100,00	0	0	100,00												
ColorMap.hpp	(1)	(0)	0,00				3	0	0,00	6	0	0,00	(3)	(1)	33,33				3	1	33,33	3	1	33,33
ColorMap.hpp	(50)	(42)	73,30	120	94	78,33	25	22	88,00	144	108	75,00	(25)	(22)	88,00	(72)	(52)	72,00	24	21	87,50	66	50	75,75
ColorMapReader.cpp	(8)	(5)	55,38	13	8	61,54	339	6	2,00	988	32	3,00	(8)	(6)	75,00	(21)	(16)	76,00	8	6	75,00	20	15	75,00
ColorMapReader.hpp	(8)	(0)	0,00	56	0	0,00	4	1	25,00	54	17	31,00	(4)	(4)	100,00	(40)	(28)	70,00	4	4	100,00	25	21	84,00
ColorMapWriter.cpp	(4)	(0)	0,00				335	2	1,00	956	4	1,00	(4)	(2)	50,00				4	2	50,00	4	2	50,00
ColorMapWriter.hpp	(6)	(6)	89,61	20	16	80,00	3	0	0,00	28	0	0,00	(3)	(3)	100,00	(20)	(18)	90,00	3	3	100,00	15	13	86,66
ColorNormalization.hpp	(32)	(28)	88,00				14	0	0,00	28	0	0,00	(14)	(11)	78,00				14	11	78,57	14	11	78,57
RGBColor.cpp							0	0	100,00	0	0	100,00												
RGBColor.hpp	(4)	(1)	60,00				4	3	75,00	8	6	75,00	(4)	(4)	100,00				4	4	100,00	4	4	100,00
RGBColor.hpp	(24)	(24)	100,00				12	12	100,00	24	24	100,00	(12)	(12)	100,00				12	12	100,00	12	12	100,00
RGBtoHSV.cpp							0	0	100,00	0	0	100,00												
RGBtoHSV.hpp	(7)	(0)	0,00	14	0	0,00	8	0	0,00	26	0	0,00	(8)	(5)	62,00	(10)	(9)	90,00	7	4	57,14	13	10	76,92

- Testwell CTC++ instruments 3 additional cpp files, which contain only preprocessor directives and namespace declarations.
- Another major discrepancy appears in case of two files *ColorMapReader.cpp* and *ColorMapWriter.cpp*.

Conclusions

- There are three main reasons for the varying results:
 - some of the tools report header files either inside the code files they were included in or as own file entities, while others report them separately,
 - the definition of the used coverage metric also differs,
 - some of the tools seem not to consider all source files provided.
- Due to the fast learning curve, intuitive user interface, and easy automation, we decided to further use *C++ Coverage Validator*.

Collaboration

- *How did you get incontact?*
 - Joanneum Research contacted us
- *How did you collaborate?*
 - Joint meetings for obtaining the needs and requirements
 - Providing a solution for the most challenging need
 - Discussing the solution with Joanneum Research
- *How long have you collaborated?*
 - A little bit more than one year
- What challenges/success factors did you experience?
 - Knowing the needs and requirements of the partner
 - Experience should fit needs and requirements
 - Open discussion culture
 - There is sometimes a gap between what academic partners can provide and industry is asking for.