International Journal of Current Innovations in Interdisciplinary Scientific Studies

Volume 6, Issue 1, (2022) 69-77 www.ij-ciss.eu

Static memory analyzer for automotive industry.

Berkay Saydam [1,], TTTech Auto Turkey.

Suggested Citation:
Saydam, B. (2022). Static memory analyzer for automotive industry. *International Journal of Current Innovations in Interdisciplinary Scientific Studies. 6*(1), 69-77. Available from: www.ij-ciss.eu

Received from March 29, 2022; revised from April 25, 2022; accepted from June 15, 2022.
Selection and peer-review under the responsibility of Vasfi Tuğun Ph.D. Kyrenia University, Girne, North Cyprus.

Abstract

The number of electronic systems and functions used in modern vehicles increased greatly. Developments in the Internet of Things also brought with it the need to keep more information on non-volatile memory. Previous static tools which have been used for this purpose have been relatively slow. With this study, we aimed to decrease the time to reach knowledge and decrease the effort to follow up on the condition of a memory system, by designing a tool. The tool is then tested on 6 participants, using different parameters, and the measurements are taken and analyzed. Waste of time was measured according to mid-level and senior developers. Our tool takes the input to prepare the report which includes the necessary information. According to the test results, which are given in the evaluation part, our tool performs as expected. The effort of release management was decreased with this tool. Memory information was made available daily by adding Jenkins. Our tool gives the opportunity to be worked independently and the system also can be followed up easily.

Keywords: AUTOSAR; memory-stack; memory visualization; NVM analyzer; static tool.

***** ADDRESS FOR CORRESPONDENCE: Berkay Saydam Evaluation of Scheduling Architectures in Automotive Industry. Berkay Saydam1,2, Tolga Ayav2. 1Ar-Ge Merkezi. TTTech Auto Turkey*.*
Email address: berkay.saydam@tttech-auto.com

1. Introduction

Many new sensors are activated with the development of the Internet of Things (IoT). Nowadays, these sensors are applied to the automotive industry (Uden & He, 2017). While the increase of the sensor causes the increase of the data, this rising causes the need to use the memory more effectively. This also causes the development time to increase. Automotive Open System Architecture (AUTOSAR, 2020) is used in the software development of the automotive industry, commonly. This architecture provides modularity by having various modules which are developed by a different developer. At the development step, developers need to see the various conditions of their component such as where it is located, and how much size it is needed. Besides, system architects have to know the usage of memory per module to verify the system.

We designed a tool that provides us with locations and sizes of blocks. It is added to the nightly build in Jenkins (2020). Jenkins is an open-source automation system that provides the building and testing of software. Our tool was adapted to Jenkins and detection of changes on the memory map is done with this nightly build. Overlap and oversize conditions are detected with automated test routines as well. When a developer changes code, others can access the information of blocks, easily. The developer can verify their code. System architects also can follow up on the development process and can decide on future works. Our purpose also was to decrease the waste of time. It was tested with some different parameters and measurements. The results were given in Table 1.

In this article, related works in literature were researched and findings were found in the second section. The differences between tools and also innovation aspect of our tool were told in the related works section. General information about non-volatile memory (NVM) was given in the second section to create a background. The third section includes a description of memory structure and the position of the AUTOSAR. The importance of analyzing tools in software development was briefly described to emphasize our study in section 5. Our implementation and its capabilities were shown in section 6. Our outputs and the result of the examination were presented under the evaluation heading. Implementation is given in section 5. The last part was the conclusion part which includes information that leads the future studies.

1.1. Related research

Other types of tools check from a hardware perspective. One of them is MEMPOWER (Rawson & Austin, 2004). MEMPOWER is a toolset that has tools to determine utilization and calculate the power and energy consumption of the memory hardware. This also does in the automotive industry with similar tools. Our study has a different perspective according to these studies. AUTOSAR has a modular structure and it aims to provide modularity for working on different parts of architecture independently. However, it is not provided completely. Each unit has a domain expert. Domain experts need to check some information every day because of other unit developer changes. This also causes a waste of time. Usage of memory unit by unit and the values of them are generated with our tool. Thus, developers can access it easily each day. Besides, system architects also can use this document to decide on future units. There is no tool to do unit-based memory analysis and visualize it.

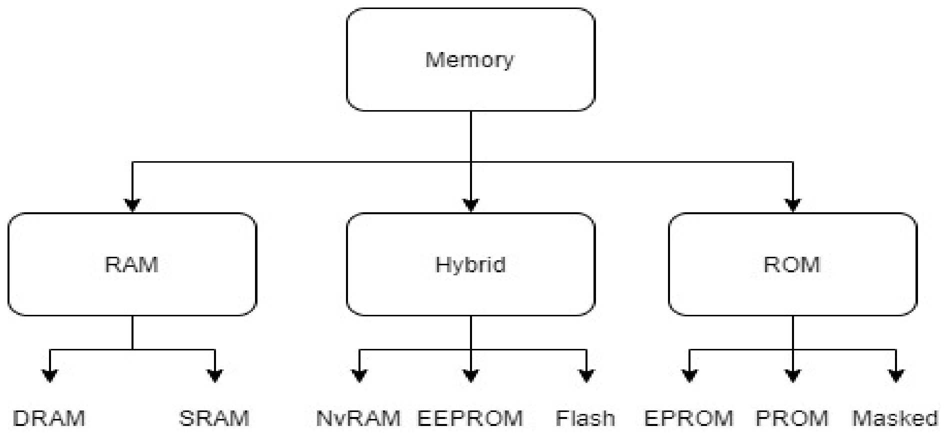
1.1.1. Classification Of Memory

Memory can be classified as volatile and non-volatile memory (Barr, 2001). Volatile memory loses all data when the system is shut down; it requires constant power to provide data. Non-volatile memory does not lose its data when the system or device is shut down. This is the main difference between them.

Memory devices are separated into three groups from a software developer's perspective. This classification is represented in Figure 1.

Figure 1

Classification of Memory



The first one is Random Access Memory (RAM). Static RAM (SRAM) and Dynamic RAM (DRAM) are important parts of devices that are found in this group. The main difference between them is DRAM loses its data in a very short time even if it has power. DRAM can behave like SRAM by refreshing the data before it expires.

Another group is Read-Only Memory (ROM). Devices of this group can retain data even during a power failure. The most basic ROM is masked ROM which is a hardwired device. Programmable ROM (PROM), which is one level up according to ROM, can be programmed by a device programmer, and its contents can never be changed. Therefore, they are called as one -time programmable (OTP). Erasable-and-programmable ROM (EPROM) is similar to PROM, EPROM can be erased and reprogrammed in addition.

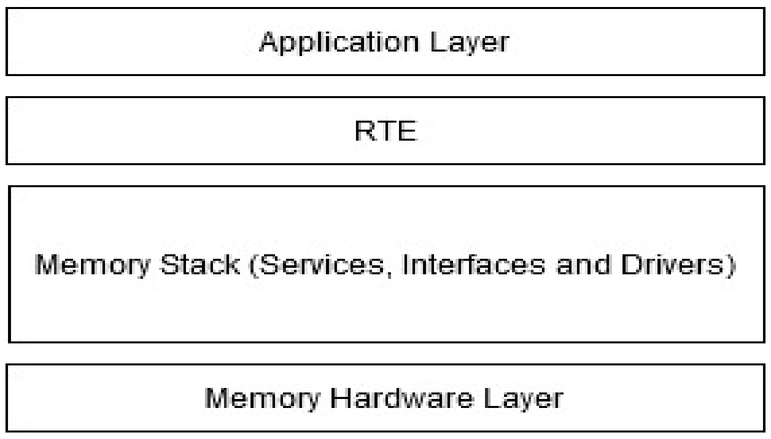
The last group is Hybrid which is a combination of RAM and ROM. These memories are used to write and read like RAM. They also don't lose their data even if they haven't power like ROM. EEPROM and flash are used to store code. NVRAM, which is a modified version of SRAM, is used to hold persistent data.

1.1.2. Autosar architecture and memory stack

AUTOSAR is an architecture that is commonly used in the automotive industry. This architecture is divided into some layers such as drivers, stacks, Runtime Environment (RTE), and also applications. Memory Stack (MemStack) provides memory management services from the lower layer to the upper layer (AUTOSAR 4.3.1., 2020). The layers are found in Figure 2.

Figure 2

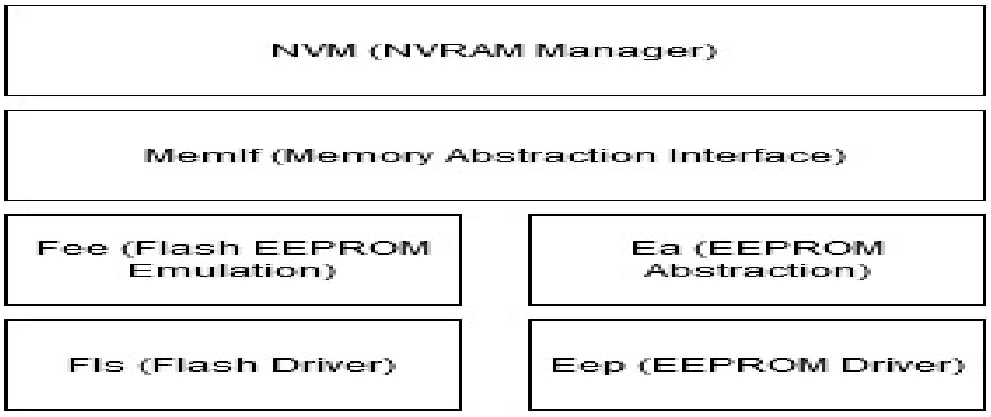
Memory Section of Autosar Architecture



An application, which is found in the application layer, can access Non-volatile memory only via NVRAM Manager (NvM). Data can be stored as RAM/ROM/NV blocks in a different memory location such as RAM, ROM, and Nv Memory. NvM module provides management and maintenance service for data of EEPROM and Flash devices. Memory Abstraction Interface (MemIf) is an interface that provides to replace access between Flash EEPROM Emulation (FEE) and EEPROM Abstraction (EA). Flash and EEPROM Driver can be accessed via these two modules. The Diagram of MemStack is shown in Figure 3.

Figure 3

The structure of MemStack



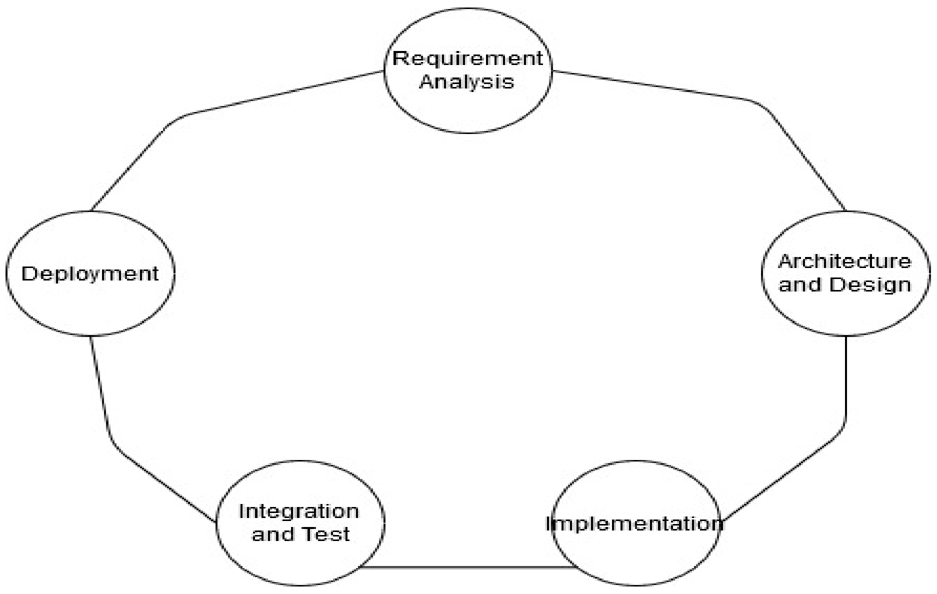
Each module that is found in MemStack can be implemented and maintained by different developers because they include wide scope. This condition occurs dependency between developers.

1.1.3. The Importance of Analysing Tools in Software Development

The key point in software development is developing software according to Software Development Life Cycle (SDLC) (Pressman, 2005; Sommerville, 2008). There are many stages in this cycle. Although these stages seem to extend the software development process, they are not. They produce software with the highest quality and lowest cost in the shortest time possible. An example of SDLC is shown in Figure 4.

Figure 4

An instance SDLC



Static analysis tools provide critical support before runtime. It ensures continuous code quality both in the development and maintenance stages. Another benefit of them is reducing costs and risks of security. Security is a hot topic in the automotive industry, so these tools are common use in this sector (Graham, 2017).

There have been many studies about efficient usage of memory in long term. Some of them do this by finding some mistakes in software. There are predefined rules and a tool applies them to find developers' mistakes. One of these tools is Gleipnir (Janjusic, Kavi & Potter, 2011; Tao et al., 2021). The automotive industry has also a similar way. Motor Industry Software Reliability Association (MISRA) established a set of software development guidelines for the use of C programming language in safety-critical systems. These predefined rules are checked out by the MISRA checking tool.

1.2. Purpose of study

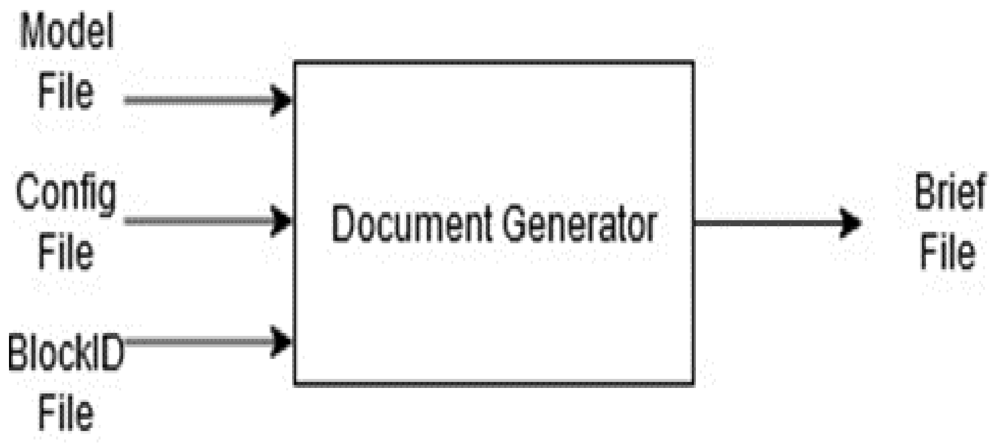
There were many modules in AUTOSAR architecture. Each module had its data. Whole data locations and their sizes, which are determined by their developers, must be seen together at the system level. There caused too much waste of time to obtain these data. Development and verifying efforts are reduced with our tool. Besides, developers should work independently from each other. Available organizations forced developers to align with each other. With this study, we aimed to decrease the time to reach knowledge and decrease the effort to follow up on the condition of a memory system, by designing a tool.

2. Materials and Methods

Our implementation is a static analyzer tool that provides a useful document to be used by developers and architects. It evaluates the project and takes Microcontroller Unit (MCU) specific files as inputs. According to inputs, it prepares output such as the location of an entity on non-volatile memory, size of the entity, and component that entity belongs. The system is represented in Figure 5.

Figure 5

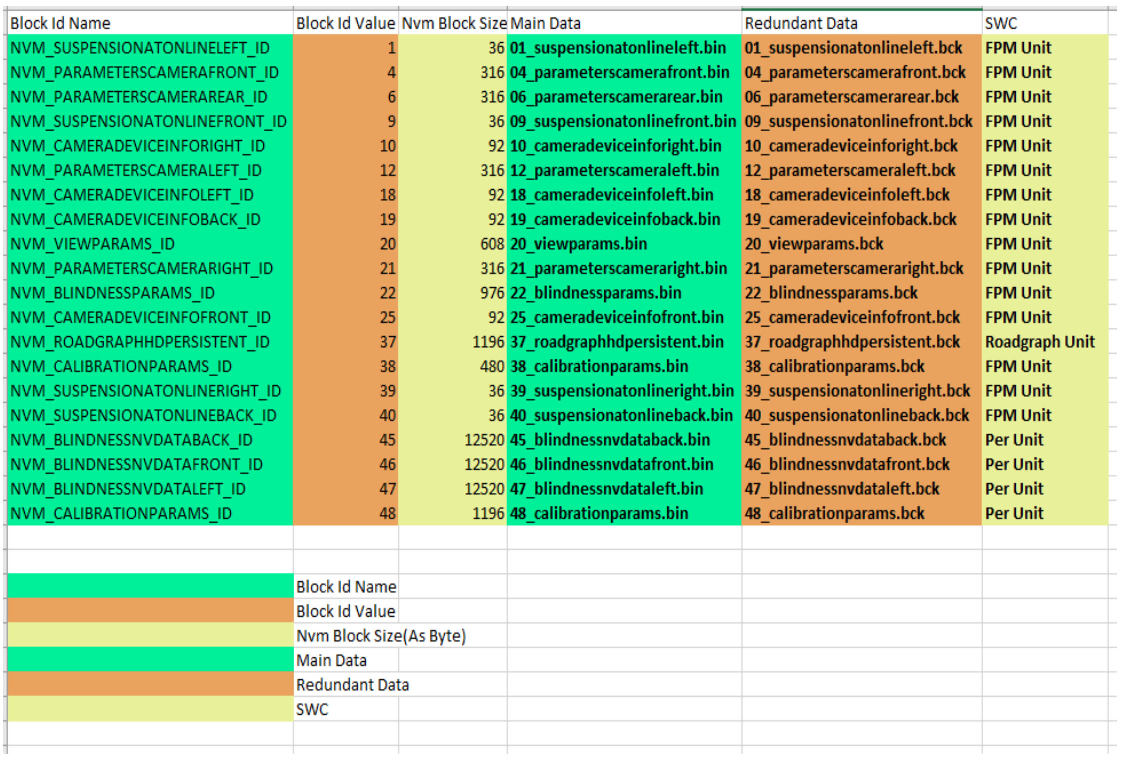
System I/O



A batch file is designed. This file sets the paths of files that are necessary as input first. It calls a python file to parse inputs for obtaining desired information about memory. The obtained information is filled into a brief file using some libraries. This output includes the name, id, nvm block size, paths, and SWC of related blocks as columns. It is found in Figure 6 as an example.

Figure 6

System Brief File



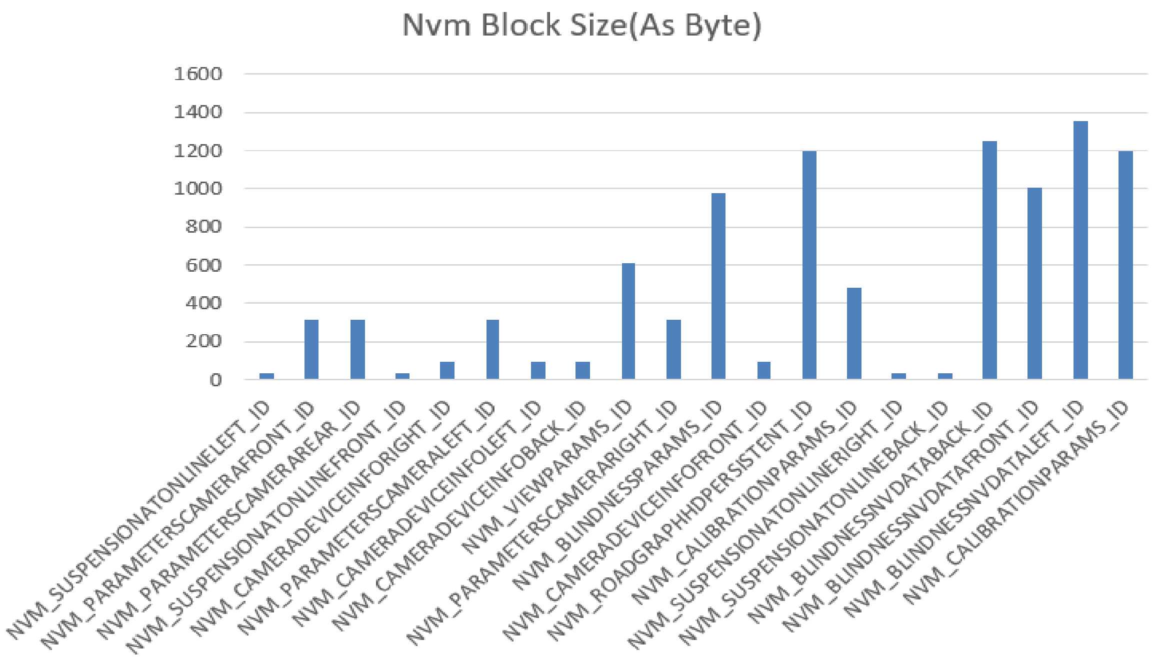
Block Id Name	Block Id Value	Nvm Block Size	Main Data	Redundant Data	SWC
NVM_SUSPENSIONATONLINELEFT_ID	1	36	01_suspensionatonlineleft.bin	01_suspensionatonlineleft.bck	FPM Unit
NVM_PARAMETERSCAMERAFRONT_ID	4	316	04_parameterscamerafront.bin	04_parameterscamerafront.bck	FPM Unit
NVM_PARAMETERSCAMERAREAR_ID	6	316	06_parameterscamerarear.bin	06_parameterscamerarear.bck	FPM Unit
NVM_SUSPENSIONATONLINEFRONT_ID	9	36	09_suspensionatonlinefront.bin	09_suspensionatonlinefront.bck	FPM Unit
NVM_CAMERADEVICEINFORIGHT_ID	10	92	10_cameradeviceinforight.bin	10_cameradeviceinforight.bck	FPM Unit
NVM_PARAMETERSCAMERALEFT_ID	12	316	12_parameterscameraleft.bin	12_parameterscameraleft.bck	FPM Unit
NVM_CAMERADEVICEINFOLEFT_ID	18	92	18_cameradeviceinfoleft.bin	18_cameradeviceinfoleft.bck	FPM Unit
NVM_CAMERADEVICEINFOBACK_ID	19	92	19_cameradeviceinfoback.bin	19_cameradeviceinfoback.bck	FPM Unit
NVM_VIEWPARAMS_ID	20	608	20_viewparams.bin	20_viewparams.bck	FPM Unit
NVM_PARAMETERSCAMERARIGHT_ID	21	316	21_parameterscameraright.bin	21_parameterscameraright.bck	FPM Unit
NVM_BLINDNESSPARAMS_ID	22	976	22_blindnessparams.bin	22_blindnessparams.bck	FPM Unit
NVM_CAMERADEVICEINFOFRONT_ID	25	92	25_cameradeviceinfofront.bin	25_cameradeviceinfofront.bck	FPM Unit
NVM_ROADGRAPHHDPERSISTENT_ID	37	1196	37_roadgraphhdpersistent.bin	37_roadgraphhdpersistent.bck	Roadgraph Unit
NVM_CALIBRATIONPARAMS_ID	38	480	38_calibrationparams.bin	38_calibrationparams.bck	FPM Unit
NVM_SUSPENSIONATONLINERIGHT_ID	39	36	39_suspensionatonlineright.bin	39_suspensionatonlineright.bck	FPM Unit
NVM_SUSPENSIONATONLINEBACK_ID	40	36	40_suspensionatonlineback.bin	40_suspensionatonlineback.bck	FPM Unit
NVM_BLINDNESSNVDATABACK_ID	45	12520	45_blindnessnvdataback.bin	45_blindnessnvdataback.bck	Per Unit
NVM_BLINDNESSNVDATAFRONT_ID	46	12520	46_blindnessnvdatafront.bin	46_blindnessnvdatafront.bck	Per Unit
NVM_BLINDNESSNVDATALEFT_ID	47	12520	47_blindnessnvdataleft.bin	47_blindnessnvdataleft.bck	Per Unit
NVM_CALIBRATIONPARAMS_ID	48	1196	48_calibrationparams.bin	48_calibrationparams.bck	Per Unit

🟩	Block Id Name
🟧	Block Id Value
🟨	Nvm Block Size(As Byte)
🟩	Main Data
🟧	Redundant Data
🟨	SWC

Visualization is a very important feature for analyzer tools. Visualization of values makes it easier to decrease developer effort. The second sheet of the tool output has a chart that shows the usage of memory module by module. It is shown in Figure 7.

Figure 7

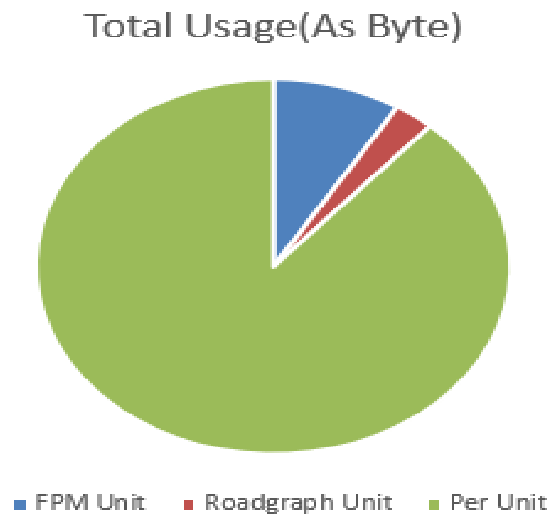
Usage of memory module by module



Generally, system architects want to know the usage of memory module by module. Our tool can calculate and visualize it as a pie chart due to byte. The output of this visualization is represented in Figure 8.

Figure 8

The visualization of memory usage



3. Results

Our implementation aimed to decrease the waste of time. Therefore, it was tested with some different parameters, and the measurements were recorded. These values are given in Table 1.

Table 1

The results of the tests

Person who attends the test	Experienced (years)	Title	Time spent without Document Generator (hours/week)	Average time saved (hours/week)
Person 1	3-5	Mid-Level	20	17
Person 2			16.5	
Person 3			14.5	
Person 4	5-10	Senior (Domain Expert)	6.5	6
Person 5			3	
Person 6			8.5	

In the first step, we want to make mid-level employees find related information. They should know the correct locations and interpret them. They also calculate the size of related units. It was tested for three different people and the average of their time was 17 hours. A software engineer in a general topic needs this information once every two weeks. It equals 4.25 working days a month. There is a waste of 51 business days for a year. The average domain experts' time was 6 hours.

4.Discussion

This test also is applied to domain experts. They have experienced between 5 and 10 years. They are pretty much accustomed to doing this process. Nevertheless, they spent an average of 6 hours on this work. A domain expert in memory topics needs this information once a week. It equals 3 working days a month and it is a waste of 36 business days for a year. Our tool finds this information each day after nightly build.

With our tool, this information and calculations are done automatically with a nightly build. Thus, this information is available daily. Nowadays, encountering customer needs is very important. Companies focus on providing products to market as fast as possible. They use some methodology like Agile Software Development (Knaster, 2022), as well.

These are not enough to give a significant response to market changes. Therefore, some impediments, which are seen in SDLC, should be solved to avoid waste of time. The motivation of this paper is a decrement in dependency between engineers and decreasing the effort of engineers to have product knowledge. There is no study with this approach in literature as mentioned before in the related works section. This tool can be added to bug tracking and agile project management system as a future study. Thus, the developers can share their analysis of the output of our tool, and they can discuss and follow some open points there.

5. Conclusion

With this study, we aimed to decrease the time to reach knowledge and decrease the effort to follow up on the condition of the memory system. According to test results, which are given in the evaluation part, our tool provides them as expected. The effort of release management was decreased with this tool. Memory information was made available daily by adding Jenkins. Related software can be built and tested with an automated building and testing routine. The important situations in memory like overlap and oversize are detected with these routines. Memory locations can be checked and maintained easily. Now developers can work independently. There is no blocking between them.

NVRAM changes have been automated. This benefit provided easy tracking for system architects. Now, a system architect can track the system and he/she can decide on future works, easily. For future studies, many memory maps can be defined, and then they can be used to avoid aging with changing maps in runtime. Another idea is this tool can be designed for RAM usage to increase memory performance.

References

AUTOSAR 4.3.1. (2020). Document ID: 053. (Accessed 2 April 2020). Retrieved from https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

AUTOSAR Official Website. (2020). (Accessed 29 March 2020). Retrieved from https://www.autosar.org/

Barr, M. (2001). Memory Types Embedded Systems Programming. pp. 103-104. https://barrgroup.com/embedded-systems/how-to/memory-types-ram-rom-flash

Graham B. (2017)., "The Role of Static Analysis in a Secure Software Development Life Cycle", GrammaTech Blog. https://blogs.grammatech.com/the-role-of-static-analysis-in-a-secure-software-development-lifecycle

Janjusic, T., Kavi, K., & Potter, B. (2011). International conference on computational science, iccs 2011 gleipnir: A memory analysis tool. *Procedia Computer Science*, *4*, 2058-2067. https://doi.org/10.1016/j.procs.2011.04.225.

Jenkins Official Website (Accessed 23 September 2020). Retrieved from https://www.jenkins.io/doc/

Knaster, R. (2022, June 2). *Safe 5.0 framework*. Scaled Agile Framework. Retrieved June 10, 2022, from https://www.scaledagileframework.com/

Pressman, R. S. (2005). Software Engineering: A Practitioner's Approach. 6th. http://rku.ac.in/syllabus/syllabus/190721085652Information%20Technology%20Sem-6%20Syllabus.pdf

Rawson, F., & Austin, I. (2004). Mempower: A simple memory power analysis toolset. IBM Austin Research Laboratory, 3. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.4597&rep=rep1&type=pdf

Sommerville, I. (2008, March). Construction by configuration: Challenges for software engineering research and practice. In *19th Australian Conference on Software Engineering (ASWEC 2008)* (pp. 3-12). IEEE. https://ieeexplore.ieee.org/abstract/document/4483184/

Tao, R., Shi, Y., Yao, J., Hui, J., Chong, F. T., & Gu, R. (2021, June). Gleipnir: toward practical error analysis for Quantum programs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (pp. 48-64). https://dl.acm.org/doi/abs/10.1145/3453483.3454029

Uden, L., & He, W. (2017). How the Internet of Things can help knowledge management: a case study from the automotive domain. *Journal of Knowledge Management*. https://www.emerald.com/insight/content/doi/10.1108/JKM-07-2015-0291/full/html