International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 12 | Dec 2020 www.irjet.net p-ISSN: 2395-0072

Improving Software Quality by Developing Redundant Components

Onuora, Augustine Chidiebere[1], Ana, Prince[2], Nwanhele, U. N.[3], Idemudia, Osahon Joseph[4]

[1, 4]Department of Computer Science, Akanu Ibiam Federal Polytechnic Unwana
[2]Department of Computer Science, Cross River University of Technology Calabar
[3]Department of Office Technology and Management, Akanu Ibiam Federal Polytechnic Unwana

---

Abstract - *Software systems have become pervasive in everyday life and are the core component of many crucial activities. A software system is redundant when it performs the same functionality through the execution of different elements. An inadequate level of reliability may determine the commercial failure of a software product. Nevertheless, despite the commitment and the rigorous verification processes employed by developers, software is deployed with faults. To increase the reliability of software systems, Programmers and software developers need to embrace some of the redundancy techniques highlighted in this study. This study x-rayed previous works with the aim of getting best practices that will help in improving the quality of software. It further reviewed literatures on the subject and highlighted various fault tolerance taxonomy that can help a software developer or programmer in developing redundant components thereby increasing the reliability of a software system with improved overall quality.*

***Key Words*: Software quality, Reliability, Fault-tolerance, Redundancy, Diversity**

1. INTRODUCTION

Recently the world was agog with the recent Boeing 737 Max jets en route to Nairobi, crashed shortly after take-off from Addis Ababa. It has been confirmed that 157 passengers on board all lost their lives. This tragedy was as a result of an error in the Boeing aircraft's flight-control software (AJC, 2019).

Numerous softwares all over the world today have one type of error or the other. The consequences of this errors ranges from financial loss, communication loss to even the loss of human life as the case of the Boeing 737 Max aircraft. It is the duty of software developers and programmers to design softwares that are fail-safe. Software generally should be developed with the best software engineering practice. Error should be eliminated from not only critical software but softwares at all levels, be it the operating system on mobile phones, Televisions, Pcs or embedded software on electronic gadgets.

Software errors lead to software failures. A software failure is not healthy for the computing world. Softwares should be developed with all correctness and made fail-safe. Mission critical software doesn't need to fail because whatever that can cause error in the software can bring about a disastrous output. If we consider a rocket launch software that was developed with a fault, the outcome of the rocket launch could be disastrous. Similarly, assuming road traffic software has fault and was implemented on a road, the number of cars and people that will be accident casualties might be high.

Software errors are always directly caused by either the programmers or program developer that left those errors in the code. As humans they have a large probability of doing something wrong.

There is usually an industry standard or framework to stipulate how softwares especially mission critical software can be developed. The challenge is that there are few trained software developers and programmers that are aware of this industry standard. One of the industry standards to solving the issue of failing softwares is the development of redundancy components.

2. REVIEW OF RELATED LITERATURES

This section reviewed related work done on software redundancy, redundant component and diversity.

Antonio Carzaniga, Andrea Mattavelli, and Mauro Pezzè. (2015) stated in their work that Redundancy simply is the occurrence of different elements with the same functionality. In software, redundancy is useful (and used) in many ways, for example for fault tolerance and reliability engineering, and in self-adaptive and self-checking programs. Airplane softwares should be fault tolerant. They should also be self-adaptive and self-checking. This is an area of utmost concern because this could be a reason the Boeing 737 Max of the Ethiopian airline crashed. Software developers and programmers should indeed find a way to determine that critical softwares are fail-safe through redundancy.

Antonio Carzaniga, Andrea Mattavelli, and Mauro Pezzè. (2015) further opined that, we still do not know how to measure software redundancy to support a proper and effective design. If, for instance, the goal is to improve reliability (software quality), one might want to measure the redundancy of a solution to then estimate the reliability gained with that solution. Or one might compare alternative solutions to choose the one that expresses more redundancy and therefore, presumably, more reliability. This can be actualized through formalizing a notion of redundancy whereby two code fragments are considered redundant when they achieve the same functionality with different executions. On the basis of this notion, Programmers and software developers working with Boeing are counselled to adapt to the software engineering principles of redundancy where various versions of code fragments are written to solve a task.

Mark vandenBrand and Jan Friso Groote (2013) stressed that Software engineers are humans and so they make lots of mistakes. Typically 1 out of 10 to 100 tasks go wrong. The only way to avoid these mistakes is to introduce redundancy in the software engineering process. However, one cannot say that mission critical softwares will not be developed because of the imminent error that is expected instead, programmers and software developers should consciously follow laydown industrial software engineering practices. Software redundant components are part of software engineering best practices to improve software quality and make a software fail-safe.

They went ahead to propose that depending on the required level of correctness, expressed in a residual error probability (typically 10); each programming task must be carried out redundantly 4 to 8 times. This number is hardly influenced by the size of a programming endeavour. Trained software engineers require a double amount of redundant tasks to deliver software of a desired quality. More compact programming, for instance by using domain specific languages, only reduces the number of redundant tasks by a small constant. (Mark vandenBrand and Jan Friso Groote, 2013).

According to National Research Council (2015), Redundancy exists when one or more of the parts of a system can fail and the system can still function with the parts that remain operational. Two common types of redundancy are active and standby.

In active redundancy, all of a system's parts are energized during the operation of a system. In active redundancy, the parts will consume life at the same rate as the individual components. An active redundant system is a standard "parallel" system, which only fails when all components have failed.

In standby redundancy, some parts are not energized during the operation of the system; they get switched on only when there are failures in the active parts. In a system with standby redundancy, ideally the parts will last longer than the parts in a system with active redundancy. A standby system consists of an active unit or subsystem and one or more inactive units, which become active in the event of a failure of the functioning unit. The failures of active units are signalled by a sensing subsystem, and the standby unit is brought to action by a switching subsystem.

There are three conceptual types of standby redundancy: cold, warm, and hot. In cold standby, the secondary part(s) is completely shut down until needed. This type of redundancy lowers the number of hours that the part is active and does not consume any useful life, but the transient stresses on the part(s) during switching may be high. This transient stress can cause faster consumption of life during switching. In warm standby, the secondary part(s) is usually active but is idling or unloaded. In hot standby, the secondary part(s) forms an active parallel system. The life of the hot standby part(s) is consumed at the same rate as active parts. Redundancy can often be addressed at various levels of the system architecture.

For a software to be fault tolerant, there are various techniques that can be employed. Omar Anwer Abdul, HameedIsraa Abdulameer Resen and Saif A Abd (2019) advocated that there are two types of software fault tolerance techniques namely single version and multi version.

- Single version techniques aim to improve the fault tolerance of a software component by adding to it mechanisms for fault detection, containment, and recovery.
- Multi-version techniques use redundant software components which are developed following design diversity rules. As in the hardware case, various choices have to be examined to determine at which level the redundancy has to be provided and which modules are to be made redundant.

One has to be aware that the increase in complexity caused by redundancy can be quite severe and may diminish the dependability improvement, unless redundant resources are allocated in a proper way. This could be a major setback to producing quality software for mission critical situations like that of an aircraft or rocket launcher. When redundancy is adopted, software engineers should be very mindful of the complexity of the system.

Mark vandenBrand and Jan Friso Groote (2013) remarked that Redundancy is obtained through the independent development of components with the same functional behaviour. In its most extreme form two independent groups develop components that can be executed in parallel. These components need not be programmed in the same language. A variant is the development of a (executable) model, which can be used for prototyping, testing or code generation. If the model is machine-processable, it can be used for simulation and/or model checking.

Thus, for a software developer or programmer to employ redundancy in the software design process, he should be programmer with the skill of developing in more than one programming language. Often times, it is better to allow a different programmer or software developing firm handle the redundant component in a different language aimed at achieving the same result with the goal of removing as many of the flaws that will be inherent in each description.

They further opined that that several forms of redundancy are already present in actual programming, such as type checking and testing. However, these forms of redundancy came about as good practices, not conscious ways to introduce redundancy with a view to attaining a certain level of software quality. Active redundancy can be brought into the software design process through the introduction of high level models of the software, for instance, in the form of domain specific languages, property languages such as modal logics to independently state properties, independently (and perhaps multiple) constructed implementations, and a priori described test cases. The comparison of these different views can be done by model checking (software or models against properties), model based testing (model against implementation), and systematic testing (tests against model or software). Code inspection and acceptance tests are also fruitful, but lack the rigour of comparison that the more

mathematical methods have. (Mark vandenBrand and Jan Friso Groote, 2013).

3. REDUNDANCY AND DIVERSITY

Redundancy and diversity are fundamental strategies for enhancing the dependability of any type of system. Redundancy means that spare capacity is included in a system that can be used if part of that system fails. Diversity means that redundant components of the system are of different types, thus increasing the chances that they will not fail in exactly the same way. (Sommerville, 2011).

We use redundancy and diversity to enhance dependability in our everyday lives. As an example of redundancy, most people keep spare light bulbs in their homes so that they can quickly recover from the failure of a light bulb that is in use. Commonly, to secure our homes we use more than one lock (redundancy) and, usually, the locks used are of different types (diversity). This means that if an intruder finds a way to defeat one of the locks, they have to find a different way of defeating the other lock before they can gain entry. As a matter of routine, we should all back up our computers and so maintain redundant copies of our data. To avoid problems with disk failure, backups should be kept on a separate, diverse, external device.

Software systems that are designed for dependability may include redundant components that provide the same functionality as other system components. These are switched into the system if the primary component fails. If these redundant components are diverse (i.e., not the same as other components), a common fault in replicated components will not result in a system failure. Redundancy may also be provided by including additional checking code, which is not strictly necessary for the system to function. This code can detect some kinds of faults before they cause failures. It can invoke recovery mechanisms to ensure that the system continues to operate.

In systems for which availability is a critical requirement, redundant servers are normally used. These automatically come into operation if a designated server fails.

Sometimes, to ensure that attacks on the system cannot exploit a common vulnerability; these servers may be of different types and may run different operating systems. Using different operating systems is one example of software diversity and redundancy, where comparable functionality is provided in different ways.

Diversity and redundancy may also be also used to achieve dependable processes by ensuring that process activities, such as software validation, do not rely on a single process or method. This improves software dependability because it reduces the chances of process failure, where human errors made during the software development process lead to software errors. For example, validation activities may include program testing, manual program inspections and static analysis as fault-finding techniques. These are complementary techniques in that any one technique might find faults that are missed by the other methods. Furthermore, different team members may be responsible for the same process activity e.g., a program inspection. (Sommerville, 2011).

4. METHODOLOGY / FINDINGS

4.1 Methodology

Several works that was done on the subject of software redundancy was x-rayed with a view to fish out ways to improve software quality through redundant components. This previous works was thoroughly examined and the researchers came up with the findings below.

4.2 Findings

The table below highlights the various techniques for improving software quality through redundancy.

Reviewed Papers			
S/No	Year	Authors / (Title of Journal)	Proposed Solutions
1	2013	Markvan den Brand and Jan FrisoGroote (Software engineering: Redundancy is key)	Reduce the number of tasks when programming. Train the software engineers. Introduction of redundancy when developing software. Programming languages that facilitate statistical analysis of the developed programs Reviewing & Testing Formalisation of requirements Re-use Specification of interfaces Adherence to architectural rules
2	2015	The National Academies Press (Reliability Growth: Enhancing Defense System Reliability)	Iintegrity tests virtual qualification Reliability testing
3	2015	Antonio Carzaniga, Alessandra Gorla and Mauro Pezz`e (Handling Software Faults with Redundancy)	Intention: (deliberate Types: (Code, data and Environment) Triggers and adjudicators: (preventive and reactive) Faults addressed by redundancy: (interaction and Development)

Table 1: Table showing solutions from reviewed papers.

Antonio Carzaniga, Alessandra Gorla, and Mauro Pezze (2015) further classified the various fault tolerance techniques used to improve software quality through redundancy in the table below.

Taxonomy	Intention	Type	Adjudicator	Faults
N-version programming	deliberate	Code	Reactive implicit	development
Recovery blocks	deliberate	Code	Reactive explicit	development
Self-checking programming	deliberate	Code	Reactive Explicit / implicit.	development
Self-optimizing code	deliberate	Code	Reactive explicit	development
Exception handling, rule engines	deliberate	Code	reactive explicit	development
Wrappers	deliberate	Code	preventive	Bohrbugs malicious
Robust data structures, audits	deliberate	Data	reactive implicit	development
Data diversity	deliberate	Data	Reactive Explicit / implicit.	development
Data diversity for security	deliberate	Data	reactive implicit	malicious
Rejuvenation	deliberate	environment	preventive	Heisenbugs
Environment perturbation	deliberate	environment	reactive explicit	development
Process replicas	deliberate	environment	reactive implicit	malicious
Dynamic service substitution	opportunistic	Code	reactive explicit	development
Fault fixing, genetic programming	opportunistic	Code	reactive explicit	Bohrbugs
Automatic workarounds	opportunistic	Code	reactive explicit	development
Checkpoint-recovery	opportunistic	environment	reactive explicit	Heisenbugs
Reboot and micro-reboot	opportunistic	environment	reactive explicit	Heisenbugs

Table 2: Table showing taxonomy of redundancy

5. CONCLUSIONS

The place of redundant components in developing software cannot be overemphasized. This is not for only mission critical softwares like that of the Boeing 737 Max, Rocket launcher software but for day to day utility software. Redundancy is the presence of different elements with the same functionality. In software development, redundancy is applied in fault tolerance, reliability engineering, self-adaptive and self-checking software programs.

Software Developers and Programmer should be trained on the importance of using redundant software component in improving software quality and making the software product a fail-safe product. Whenever an error occur on a software component, (which is probable to occur), the outcome should not be disastrous as in the case of mission critical system or stop production as in day to day utility software. Software should be built to switch to redundant components which these components are designed with high fault tolerance techniques as highlighted from reviewed techniques above. Softwares should be fault tolerant and self-healing to reduce the runtime effects of faults during software execution, to guarantee software reliability also in the presence of faults..

ACKNOWLEDGEMENT

I will like to acknowledge Dr. Edim A. E. who took time to pilot us in the “Software Engineering”. He is an Associate Professor at the Department of Computer Science, University of Calabar.

REFERENCES

[1] Antonio C., Andrea M., and Mauro P. (2015). Measuring software redundancy. In Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15), Vol. 1. IEEE Press, Piscataway, NJ, USA, 156-166.

[2] AJC. (2019, March 15). Ethiopian Airlines crash: Captain reported issues shortly after takeoff. Retrieved from https://www.ajc.com/news/national/ethiopian-airlines-flight-nairobi-crashes-with-157-people-board/PJsqjNZbGr7DBtVDrxlV2N/

[3] Kunz, P. (2014, September 24). Redundancy in the Software Design Process is Essential for Designing Correct Software. Retrieved from https://ercim-news.ercim.eu/en99/special/ redundancy-in-the-software-design-process-is-essential-for-designing-correct-softwareK. Elissa, “Title of paper if known,” unpublished.

[4] National Research Council. (2015). 5 System Design for Reliability | Reliability Growth: Enhancing Defense System Reliability. Washington, DC: The National Academies Press. https://doi.org/10.17226/18987.

[5] Omar, A. A., Israa, A. R., & Saif, A. A. (2019). Software Fault Tolerance: A Theoretical Overview. International Journal of Simulation: Systems, Science & Technology, 20(3). doi:10.5013/IJSSST.a.20.03.07

[6] Sommerville, I. (2011). Dependability engineering. In Software Engineering (9th ed., p. 767). New York, NY: Pearson Higher Ed.

[7] The National Academies Press. (2015). 5 System Design for Reliability | Reliability Growth: Enhancing Defense System Reliability. Retrieved from https://www.nap.edu/read/ 18987/chapter/7#79

[8] Van den Brand, M. G., & Groote, J. F. (2014). Redundancy on the software design process is essential for designing correct software. Retrieved from https://pure.tue.nl/ws/files/3952807/354957597377 50.pdf

[9] Van den Brand, M., & Groote, J. F. (2015). Software engineering: Redundancy is key. Science of Computer Programming, 97, 75-81. doi:10.1016/j.scico. 2013.11.020

BIOGRAPHIES

Onuora, Augustine Chidiebere is currently a lecturer with the Department of Computer Science, AIFPU.

Ana Prince is a Lecturer with the Department of Computer Science, Cross-River State University of Technology (CRUTECH).

Noble Nwanhele is currently a technologist with the Department of Office Technology and Management, AIFPU.

Joseph Osahon Idemudia is currently a lecturer with the department of Computer science, AIFPU.