Continuous Improvement (CI) on PCB Layout using DRC (Design Rule Check) Settings Conceptual Analysis

R. Kishen Kumar Naidu[1], Mohammad Harith Bin Amlus[2], Muhammad Shahar Jusoh[3], Hafizah Abdul Rahim[4], Putri Aliah Mohd Hidzir[5], Syafiqah Md Nayan[6]

Faculty of Business & Communication, Malaysia, Sport Engineering Research Centre (SERC) Universiti Malaysia Perlis (UniMAP)

DOI: https://dx.doi.org/10.47772/IJRISS.2025.91100068

Received: 14 November 2025; Accepted: 24 November 2025; Published: 29 November 2025

ABSTRACT

Continuous Improvement (CI) has become a critical strategy in enhancing the quality, reliability, and manufacturability of printed circuit board (PCB) designs. In PCB engineering, the application of Design Rule Check (DRC) settings serves as a systematic mechanism to identify layout violations and ensure compliance with electrical, mechanical, and fabrication standards. This conceptual paper examines how CI principles can be embedded into PCB layout workflows through optimized DRC configurations. The analysis highlights the role of structured rule-setting, iterative verification, and feedback loops in reducing design errors, minimizing rework, and improving overall design efficiency.The study further conceptualizes how disciplined DRC management—encompassing trace width control, spacing validation, thermal relief parameters, and manufacturability rules—supports CI initiatives by standardizing best practices and enabling consistent quality enhancements. The integration of CI frameworks with PCB design automation tools is also discussed as a strategic approach to strengthening decision-making, reducing cycle time, and facilitating cross-functional collaboration between design and manufacturing teams. This conceptual analysis provides a foundation for researchers and practitioners to better understand the relationship between CI methodologies and DRC-driven PCB optimization, while offering insights for future empirical investigation and process improvement model

Keywords: Continuous Improvement, Design Rule Check, Technical Design and Mass production

INTRODUCTION

An embedded software system is sometimes defined as a computing system that interacts with the physical world. This definition is incomplete, because every software system, once it is up and running, interacts with the physical world. More precisely, what is meant is that an embedded software system has non-functional requirements, which concern the system's interaction with the physical world.

There are two interfaces of a software system with the physical world: the environment and the platform. The environment includes the human users of the system, possibly a physical plant that is controlled by the system, and other application software processes that interact with the system. The platform consists of software and hardware components that implement a virtual machine on which the system is executed; it includes the operating system and network, with specific scheduling and communication mechanisms. Correspondingly, the non-functional requirements of an embedded software system can be classified as follows (Henzinger & Sifakis 2007):

1: reaction requirements, which concern the interaction of the system with the environment

2: execution requirements, which concern the interaction of the system with the platform.

The source of reaction requirements is user expectation, where the user may be a human, a physical plant or some other software. A common reaction requirement is response time, which bounds the worst- or average-case delay between an external stimulus of the system and its response. The source of execution requirements is resource constraints, which may be hardware imposed, such as limits on available machine cycles, memory

space, battery capacity and channel bandwidth, or software imposed, such as the use of a specific scheduling algorithm or communication protocol. Like reaction requirements, execution requirements, e.g. a bound on the power consumption of the system, may be hard (worst case) or soft (average case). While reaction requirements are independent of the platform, execution requirements change from platform to platform.

By contrast, a system that is not embedded has only functional requirements. For example, the functional requirement on a sorting program is that it outputs a sorted permutation of the input. For sorting programs, usually there is neither a reaction requirement (it is not specified when the sorted output must be provided) nor an execution requirement (it is not specified how much memory space or energy the sorting program may consume). The reaction and execution requirements are absent not because there are no corresponding user expectations or resource constraints—in practice, the patience of a user to wait for the response of a sorting program is limited, and so is the available memory space—but these constraints are neglected because they are secondary to the functional requirement. In other words, to control design complexity, it is useful to abstract reaction and execution constraints whenever this can be done, i.e. when writing non-embedded software. Indeed, the success of high-level programming languages is built to a large degree on their ability to relieve the programmer from worrying about execution details such as memory management. This is why introductory programming is best taught in a language with garbage collection. It is, however, not prudent to abstract the response time of the electronic braking system in automobiles, nor the power consumption of remote sensor nodes that scavenge their energy from the environment. In these examples, the reaction and execution requirements are not secondary to the functional requirements, but they are integral to the correct operation of the system. It is for this reason that conventional high-level languages are not suitable for programming safety-critical real-time and highly resource-constrained software systems. Hence the challenge for real-time system researchers is to develop approaches to design fast systems with easily predicted performance, or to more accurately measure existing complex but fast systems. These issues related to the performance of the system falls back to the R&D team who was in charge of the design and development of the embedded system; most issues are related to the component selection, hardware design, layout design, verification and validation of the product resulting in system’s poor response and latency in data transmission.

REVIEW OF LITERATURE

A continuous improvement process, also often called a continual improvement process (abbreviated as CIP or CI), is an on-going effort to improve products, services, or processes. These efforts can seek "incremental" improvement over time or "breakthrough" improvement all at once. Delivery (customer valued) processes are constantly evaluated and improved in the light of their efficiency, effectiveness and flexibility. Some see CIPs as a meta-process for most management systems (such as business process management, quality management, project management, and program management). W. Edwards Deming, a pioneer of the field, saw it as part of the 'system' whereby feedback from the process and customer were evaluated against organisational goals. The fact that it’s can be called a management process does not mean that it needs to be executed by 'management'; but rather merely that it makes decisions about the implementation of the delivery process and the design of the delivery process itself.

Conceptual Framework

Principles of Continuous Improvement

Whatever approach is used, the following framework helps to drive and support the process:

1. Care recipient
2. Focused
3. Strategic planning and implementation
4. Involvement of key stakeholders
5. Innovation

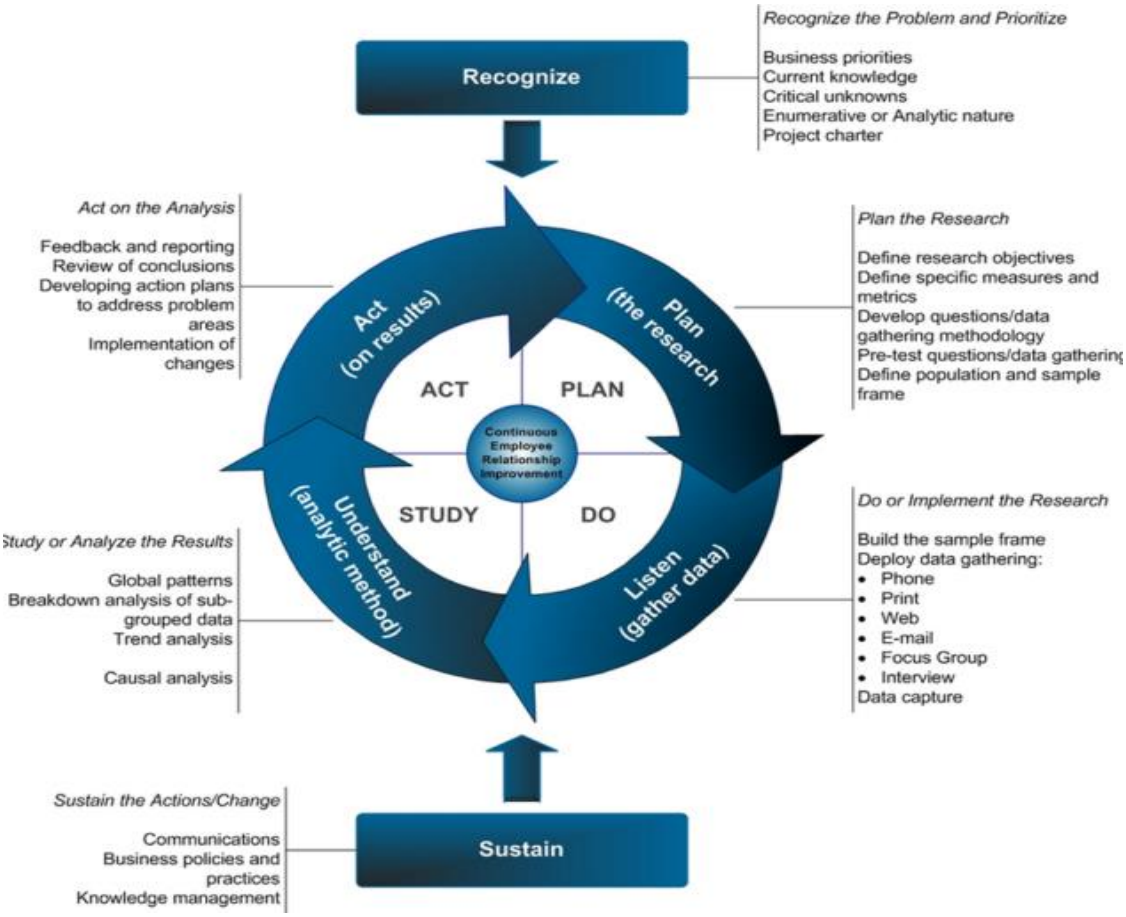
6. Regular monitoring and evaluation

A culture of continuous improvement ensures a service is responsive to change and can continually develop a quality service that is of value to its care recipients.

A sound continuous improvement program can demonstrate:

1. Baseline - the current situation the service is trying to change
2. Planned improvements and the expected benefit to care recipient
3. Monitoring - Systems to monitor a new process or activity during its implementation
4. Evaluation - Systems to monitor a process or activity once it has been implemented, which should help ensure its sustainability and capture the actual improvements.

The Continuous Improvement Model



Plan The Improvement

Continuous improvement means taking a systematic and planned approach to improving the quality of care and services including:

1. Analysing complaints trends and themes.
2. Researching possible solutions at the service level.
3. Planning and prioritising improvement activities.
4. Listening to suggestions from care recipients, representatives and staff, monitoring and evaluating new solutions, processes and improvements.

Improvements that are made in response to problems (for example, malfunctions being corrected, broken furniture being repaired) are not planned continuous improvement.

Implement the Improvement

Services should monitor new processes and activities to make sure the change is not causing problems. This will allow services to make modifications to an activity or process as required and ensure positive results for care recipients.

Evaluate Success of the Improvement Activity

Evaluating the effectiveness of a new activity or process is an important step. Ensure all components of the activity have been closed-off, for instance, updating of any policies and procedures, and seeking care recipient and staff input.

Decide Next Steps

There are at least two possible situations in this step:

1. If the improvement activity has been successful you can close the loop

2. The improvement activity has been unsuccessful or partially successful and staffs need to make amendments and start a new cycle of planning, implementing, evaluating and deciding.

Keeping Track of Important Activities

Keeping track of improvement activities ensures a strategic approach to continuous improvement, including prioritisation of activities. It also allows services to reflect back on what worked well, and what didn't.

Benefits of Implementing Continuous Improvement

The implementation of CI is a highly dynamic and complex process that brings several benefits to firms. These benefits occur in a multidimensional environment and may improve all aspects of the organization. The reinforcement of organizational learning is the first large benefit observed when implementing CI. According to Stuckman and Yammarino, continuous improvement encourages the organization to undergo a learning process to develop and select optimal solutions to problems. LeBrasseur, Whissell & Ojha recognize how continuous improvement forces the organization to revise its assumptions and values, enabling the creation of new problem solving approaches. In terms of work force learning, continuous improvement also supports the development of an integrated training and educational program. This program aims to provide skills in areas such as statistical control process (SPC), problem solving, team building, and leadership. This education is crucial, since its goal is to provide a more fundamental understanding of the elements necessary for continuous improvement to occur in the organization instead of merely creating awareness of CI. Other benefits obtained by implementing continuous improvement are observed in process and consequently in financial performance. Several studies have shown how performance metrics such as cycle time, labour usage and quality exhibit a positive trend when implementing CI. Implementing CI also benefits organizational culture and employee satisfaction. Literature indicates that CI promotes a culture where workers are willing to learn and teach, reinforcing team building concepts. Also, leaders in CI focused firms manage by facts and lead by example which creates a trustworthy work environment.

METHODOLOGY

This conceptual paper consists of a summary literature proceeding, reports and working papers. The time frame of the literature review was from 1991 to 2015. The aim of conducting this archival research method was not solely focus on the literature, but also to further investigate into the elements of achieving zero-errors in early design phase which is bringing in continuous improvement method to help improve the DRC(Design Rule Check)settings in PCB Layout . Besides, this paper also intended to disclose the stages that are involved in

continuous improvement process which can be used as a strategy to achieve an error free PCB Layout design before releasing it mass production.

DISCUSSION AND CONCLUSION

Design rule check (DRC) is a way to verify that your design meets standards or specifications provided by a manufacturer or underlying process. Running the DRC is an iterative process as you go through and fix your errors until your design meets the spec and becomes manufacturable. The DRC tests basic specifications regarding distances, trace clearances, pad clearances, and various others that when you perform a design rule check, it goes through your design and verifies that all of the edits you performed are within those tolerances.

In conclusion, implementing a continuous process improvement system in a R&D department should be done in a systematic and structured manner with a lot of careful planning and brainstorming between cross-functional or common leaders. Continuous process improvement experts such as professional consultants can be brought in to facilitate this transformation. By veering away from unstructured, ad-hoc improvements to gain short-term savings, companies can avoid failed improvement efforts, waste of resources and time, and employee frustration.

REFERENCES

1. Avoiding Common Electronics Design Mistakes. (n.d.). Retrieved August 25, 2015, from http://valydate.com/images/stories/pdf/valydate_WP_Jan2013.pdf
2. Misra, K. B. (n.d.). Handbook of Performability Engineering. Retrieved September 25, 2015, from https://books.google.com.my/books?id=cPgXg3GIMAsC&pg=PA1028&lpg=PA1028&dq=HANDBOOK OF PERFORMABILITY ENGINEERING pdf
3. Barnhart, T. (n.d.). Creating a Lean R&D System. Retrieved August 25, 2015, from https://books.google.com.my/books?id=I7FE_DLSmJQC&pg=PA18&dq=continuous improvement in R&D
4. Mitzner, K. (n.d.). Complete PCB Design Using OrCad Capture and Layout. Retrieved September 25, 2015, from https://books.google.com.my/books?id=z-tRRE9O8xMC&printsec=frontcover&dq=pcb design
5. Savolainen, T. I. (1999). Cycles of continuous improvement: Realizing competitive advantages through quality. International Journal of Operations & Production Management, 19, 1203–1222.
6. Montgomery, D. (2007, May 30). The Use of Statistical Process Control and Design of Experiments in Product and Process Improvement. Retrieved August 25, 2015, from http://www.tandfonline.com/doi/abs/10.1080/07408179208964241
7. Research. (n.d.). Retrieved September 25, 2015, from http://www.ece.ncsu.edu/research/cas/ecs
8. Schweikhart, S., & Dembe, A. (2009, October 1). The Applicability of Lean and Six Sigma Techniques to Clinical and Translational Research. Retrieved September 25, 2015, from http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2835466/
9. Sánchez, J. (2012, August 9). Assessing Sustainability of the Continuous Improvement Through the Identification of Enabling and Inhibiting Factors. Retrieved September 25, 2015, from http://scholar.lib.vt.edu/theses/available/etd-08222012-183238/unrestricted/Madrigal_J_D_2012_f1.pdf
10. Staples, M., Niazi, M., Jeffrey, R., Abrahams, A., Byatt, P., & Murphy, R. (2006). An exploratory study of why organizations do not adopt CMMI. The Journal of Systems and Software, 80, 12.