Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 53 (2016) 231 – 236

www.elsevier.com/locate/procedia

10th International Conference on Axiomatic Design — ICAD 2016

The axiomatic design of Chessmate: a chess-playing robot

Freyja Yeatman Ómarsdóttir[a], Róbert Bjarnar Ólafsson[a], Joseph Timothy Foley[a,*]

[a]*Reykjavik University, Menntavegur 1, Reykjavík 101, Iceland*

[*] Corresponding author. Tel.: +354–599–6569; fax: +354–599–6201. *E-mail address:* foley@ru.is

Abstract

Successfully completing a project on time is often a difficult task especially when the project is not well defined. This paper demonstrates the application of Axiomatic Design principles to shape and direct a multi-disciplinary project from initial conception to the final tested product. This product is Chessmate: a small robot which plays chess on a physical board. This robot is intended as a telepresence mechanism or for players who are physically challenged. Verifiable requirements were developed at the beginning of the project based upon this top level goal. These Functional Requirements ensured that the team focused on the necessary capabilities of the end product even while working on electrical, mechanical, and software elements in parallel. Construction of a design matrix identified sources of coupling that would require additional effort to avoid delays. Coupling was reduced in software by careful Application Programming Interface (API) and abstraction development. Testing parameters were explicitly stated by the requirements enabling regular validation in both software and hardware. The result was a complete chess-playing system from start to finish in 12 weeks.

Keywords: Axiomatic Design; chess; robotics;

1. Introduction

Chessmate is a software controlled mini robotic arm which moves small chess pieces around a chessboard. The target audience was players who wished to play against each other remotely or who have physical disabilities. In both of these cases, interfacing with a standard chess board is not usually an option.

This project developed over 12 weeks in the Reykjavík University Mechatronics 1 course. In this course, Axiomatic Design Theory (ADT) was first introduced as a mechanism for structuring multi-disciplinary complex projects. The ADT principles [1] were applied at the start of the project and provided guidance for the most challenging elements in the design.

Nomenclature

ADT	Axiomatic Design Theory
FR	Functional Requirement
DP	Design Parameter
C	Constraint

1.1. Chess Robots

The concept of a robot that can play a game such as chess has been around for a long time. Sadly, the earliest well-known robot “The Turk” was actually a hoax where an operator inside a cabinet controlled a manikin which moved the pieces [2]. A simple search turns up many similar concepts for automated chess players. The two most common mechanism for moving pieces is either a robotic arm that lifts the pieces or an X-Y controlled magnetic slider underneath the board [3].

Using a similar mechanism, a project called “Wireless Arduino Powered Chess” consists of twin chess boards which physically mimic each other’s moves and could theoretically be separated by thousands of miles [4] Chhangani describes a similarly named Arduino-based telepresence chess robot which uses an X-Y magnetic pickup below the chessboard for the remote player [5]. In both designs, piece movement is identified by magnetic reed switches beneath each board.

A graduation project from the Kuwait University College of Engineering and Petroleum [6] as well as a number of home projects (e.g. [7] posted on `instructables.com` use a 3 or 4 axis robotic arm. The mechanism for piece detection is also often reed switches.

The focus of these existing projects is to provide an AI (or remote player) physical mimicry during games. The human player must move the piece on the physical board to indicate a move in these cases. In comparison, the aim of Chessmate is to allow players with physical barriers to moving the pieces (e.g. remote distance or limited hand-eye coordination) to still enjoy the game.

Chess robots are inherently mechatronic systems with many parts. Developing one requires interacting software, electronics, and mechanical components. Multi-disciplinary products like this are particularly vulnerable to becoming unreasonably complicated and complex [8] without a guiding methodology. Modern designers have a plethora of methodologies to choose from including Axiomatic Design [9], TRIZ [10], Design Thinking [11], and Innovative Design Thinking [12]. Each methodology has its own means and methods with a particular focus. Axiomatic Design Theory (ADT) was chosen to be included in the Mechatronics curriculum due to its successful deployment at Reykjavik University in many disciplines including additive manufacturing [13], rocket parachute deployment [14], educational spectrometers [15], and ultrasonic carburetors [16].

1.2. Axiomatic Design Theory

The Axiomatic Design Theory (ADT) methodology [1] is a structured way to formulate a design due to its focus on how different requirements interact. This is particularly relevant on time-constrained (or otherwise resource-constrained) projects where such coupling factors can result in significant delays and overruns [8,17]. This is due to these interactions creating often unpleasant surprises later in development. Best practice indicates that Functional Requirements (FR) are focused on transformative or action verbs and can be verified [9,18]. By following this rule, the testing process flows directly from these explicit criteria. This generalized description of "what is needed" reminds the entire team of the goals and how they will be validated without constraining how to solve them. In the same way, best practice for Design Parameters (DP) indicates the focus on a noun that can be quantified or instantiated. This restriction allowed the DPs to effectively be a consistent universal language for the designers to explain how a particular element was to be solved. The concept of using ADT as an intermediate language for different disciplines which otherwise have trouble communicating is discussed in [19].

At the beginning of the design, often the voice of the customer is translated into a set of Customer Needs. In the case of Chessmate, this domain was not investigated deeply due to the developers being the main customers. The primary Customer Need (CN0) was simply "Play chess with someone who cannot touch the pieces on my board."; no further decomposition was performed. Rather, the focus was placed on developing comprehensive FR and DP lists, then evaluating the coupling between them. This coupling is symbolized in a design matrix, which is a Cartesian product of all FR and DP combinations [20,21]. Where there is an interaction between an FR and DP, this is denoted by a non-zero coefficient, or in the case of the value being unknown, simply a placeholder variable X. Minor levels of coupling, often considered higher-order effects, are annotated with x to show their lessened effect. A diagonal matrix is "uncoupled" and satisfies the Independence Axiom: "to maintain the independence of the functional requirements (FRs)" [9]. Such a design can be easily optimized by adjusting a particular FR or DPs without affecting others. A diagonal matrix indicates a "decoupled" or "path-dependent" solution, which can still be optimized, but the ordering of parameter choice selection becomes important. All other design matrices are "coupled" and may have a usable local solution but usually resist modification and optimization [9]. Needless to say, the focus is on minimizing coupling wherever it may appear.

ADT's second axiom is "minimize the information content of the design." Simply put, ensure that the design has the highest probability of meeting the stated FRs. When systems are not able to meet FRs all of the time, this is denoted in ADT as "complexity" and is deeply explored in [8]. As will become apparent in the next section, this axiom became integral to the design of the interaction between the robot and its chess pieces. Finally, any factors to be considered that are not functional are categorized as "Constraints." These are often resource-focused and affect all of the design decisions; they need to be revisited often especially when choosing between otherwise equivalent implementations.

2. Design

The design of Chessmate essentially had 4 elements to it, a software component, an electrical component and two physical components: the epitome of mechatronics. As previously mentioned, it was critical at the beginning to develop a comprehensive set of requirements before proceeding. Discussion within the team including some general concepts developed the CN0 into primary requirement and the mechanism for satisfying it:

FR_0: Synchronize chessboards in two locations without touching pieces

DP_0: Arduino-controlled robotic arm moves pieces on board based upon user input into a serial terminal (remote or local).

With the top-level complete, it was time to employ ADT's process for decomposition called "zig-zagging." The process simply states that each pair of domains completes a full "mapping-decomposition" process before moving to the next level.

2.1. Top-level Decomposition

For the Chessmate project, two levels of Functional Requirements (FRs) and their corresponding Design Parameters (DPs) were developed. The first iteration of top-level FRs and DPs can be seen in Table 1.

Upon later review, it was discovered that these FRs and DPs were heavily coupled, redundant, and incorrectly formulated as described by Thompson [22]. Applying AD best practices allowed the designers to heavily refine the FRs and DPs to their core as shown in Table 3. Similar to Bragason et al. [14], some of the originally conceived FRs were actually constraints (Table 2) due to their effect on the system as a whole.

After this analysis had been completed the next step was to specify the robot and any related geometrical constraints. The MeArm platform from MeArm Robotics in the UK was chosen due to its affordability (9000 ISK), availability (2 weeks), and suitability for interfacing with an Arduino micro-controller (existing library and connectors). Once the MeArm arrived, it was

Table 1. First draft of top level Functional Requirements and Design Parameters

ID	Functional Requirement	ID	Design Parameter
FR_1	Receive user input as to what move to make.	DP_1	Employ the serial console to receive user input.
FR_2	Translate user input to robot movements.	DP_2	Map input from the user to a predefined two-dimensional matrix in the software which corresponds to each chessboard square
FR_3	Know whether there is a piece already in the square to move the new piece to.	DP_3	Use software to determine whether there is a piece already present.
FR_4	Pick-up/put-down chess pieces without knocking over other pieces.	DP_4	Use a magnet to pick up the pieces.
FR_5	Update chessboard status in software.	DP_5	Read input and update the position matrix mentioned in DP3.
FR_6	Discard pieces.	DP_6	Use a box to use as a bin for the discarded pieces.
FR_7	Display current turn and current status of the game.	DP_7	Use the serial console and LED's to display current chessboard status.
FR_8	Have an on/off switch.	DP_8	Implement a hard wired switch connected directly to the robot.
FR_9	Reach all pieces and be able to lift them up vertically.	DP_9	The chessboard will be 12×12 cm in size, this means the squares will be 1.5×1.5 cm and the chess pieces will have diameter 12 cm.

Table 2. List of constraints (C) including decomposition

ID	Constraint
C_1	Schedule: 12 weeks
C_2	Budget: 20000 ISK
$C_{2.1}$	Robot platform: MeArm 2-axis RC-servo arm:
$C_{2.1.1}$	Max size of chessboard: 12×12 cm.
$C_{2.1.2}$	Maximum mass of the chess pieces: 10 g
$C_{2.2}$	Arduino UNO maximum program size: 32KB.

put through performance tests to determine the detailed constraints (Table 2). The reach of the arm (4–16 cm) defined the size of the chessboard ($C_{2.1.1}$). This board size of 12×12 cm indicated that the squares are 1.5×1.5 cm, so 1.2 cm was chosen as the piece diameter. Putting the arm in the fully extended position determined the maximum mass of the chess pieces ($C_{2.1.2}$). Knowing the maximum mass was critical to ensuring pieces could be moved reliably anywhere on the chessboard.

The designer considers how each of the FRs and DPs interacts to develop a design matrix as shown in Equation 1.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \end{Bmatrix} = \begin{bmatrix} X & X & 0 & 0 \\ x & X & 0 & 0 \\ 0 & 0 & X & X \\ 0 & 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \end{Bmatrix} \qquad (1)$$

This design is decoupled or "path-dependent" meaning that the order of implementation is important. There is a low degree of unavoidable coupling (x) between FR_1 and FR_2 due to the inability to completely separate the user interface from the application of chess rules. That said, by careful Application Programming Interface (API) and choice of abstraction, the Chessmate design was able to avoid having these elements be completely coupled. In addition, there is coupling between FR_3 and DP_4 due to the need to lift the pieces (in Z-axis) during pick-up before moving them to another location (X- and Y-axis)[1]. This is a minor concern due to the simplicity of programming this "path-dependence."

One demonstrable benefit of developing the design criteria before implementation is shown in Fig. 1 and Fig. 2: the difference between the concept versus the reality is very small.

Any mechanical solution must meet functional requirements FR_3 and FR_4: the pieces must be picked up and put down without knocking over other pieces and can reach all squares and pick up all pieces vertically. A great deal of effort was focused on ADT Information Axiom with respect to movement; it was critical to have consistent and stable piece placement at the end of each turn. Incorrectly placed pieces, pieces too close to an edge, and pieces that fell over were all unacceptable outcomes due to the impact on gameplay and challenge of properly restoring the board state after a failure.

3. Second Decomposition

The next phase of the "zig-zag" was to return to the Functional domain and further expand the requirements in depth and detail. The designers did not expand FR_3 any further because its needs had already been addressed completely by DP_3. For brevity, we omit any design matrices that are uncoupled (diagonal) or single-element.

The second-level FRs and DPs are much more application specific (see Table 4). Of note, the software implementation features much more heavily at this phase as it becomes the common communication medium between the various functions. In the next section, we will discuss the implementation details for each top-level FR in more detail. The instantiation of the system can be seen in Fig. 3.

[1]Many below-board robots simply nudge the other pieces out of the way, then correct the placement after.

Table 3. Top-level Functional Requirements (FR)

ID	Functional Requirement	ID	Design Parameter
FR_1	Interact with remote or local user.	DP_1	Arduino serial user interface and LED's
FR_2	Maintain state of chessboard according to chess rules	DP_2	Board configuration and chess rule-set evaluated against piece location matrix
FR_3	Move chess piece in X-, Y-, and Z-axis.	DP_3	Arm servo θ, extension ω, and lift ϕ mapped to chess board geometry using MeArm IK library
FR_4	Pick-up/put-down chess piece.	DP_4	Modulate End-effector magnet on rounded steel cap of standardized piece

Table 4. Level 2 Functional Requirements and Design Parameters decomposition. FR_3 omitted because it does not require any additional decomposition.

ID	Functional Requirement	ID	Design Parameter
$FR_{1.1}$	Parse user commands on serial terminal.	$DP_{1.1}$	`ParseInt()`: Read four integers from serial console as command.
$FR_{1.2}$	Display current turn on LED's.	$DP_{1.2}$	`showTurn()`: A yellow LED is white's turn, a blue LED is black's turn.
$FR_{2.1}$	Prevent user from playing illegal move	$DP_{2.1}$	Chess rule set library checked against requested move
$FR_{2.2}$	Synchronize internal and physical board state	$DP_{2.2}$	`updatePos()`: Movement commands queued to resolve differences between states
$FR_{2.3}$	Place captured pieces in "graveyard"	$DP_{2.3}$	`capture()` knows position of "graveyard" and executes movement to place
$FR_{4.1}$	Lower end-effector to bring within grasping and release range	$DP_{4.1}$	Z-position within 2 mm of metal cap
$FR_{4.2}$	Modulate magnet	$DP_{4.2}$	Magnet in closed plastic tube lowered towards piece via servo-actuated string.
$FR_{4.3}$	Lift bottom of piece above top of other pieces	$DP_{4.3}$	Relative Z-position target is piece height+10 mm

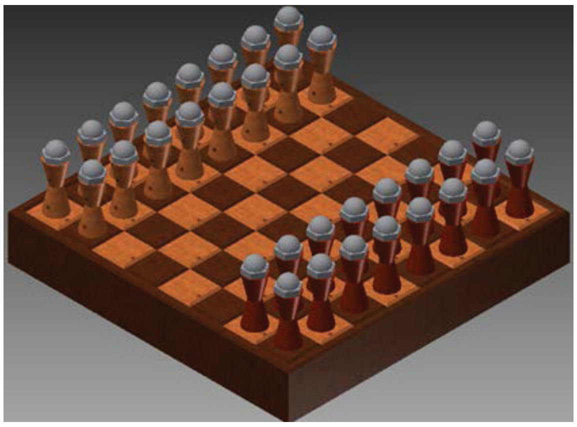
Fig. 1. Chessboard and pieces designed in CAD (AutoDesk Inventor)

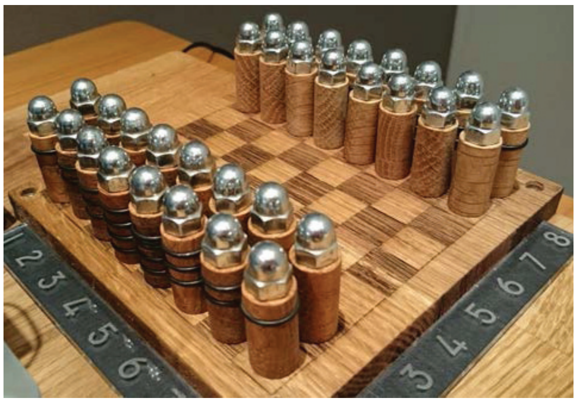

Fig. 2. Chessboard and pieces in implementation

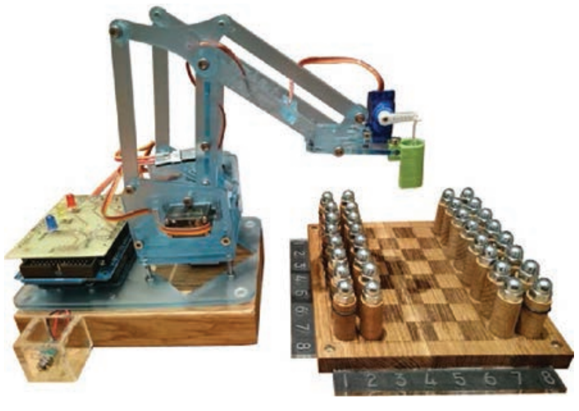

Fig. 3. Chessmate: the final product

3.1. FR$_1$ Interact with user.

A user who is next to Chessmate interfaces with it via a serial-USB connection. Commands are typed in as a series of 4 integers, which are translated to chess moves. After each command is executed, the board state is rendered on the screen using text characters. This interface was chosen to be simple so that Assisted Input methods (for visually impaired users) could easily be integrated. Existing Assisted Input methods assume that all content is text; rendering graphical content is still an open problem. In addition, this textual display can easily be presented as a service for a remote player. Depending upon the visualization desired, it might be web-based or as simple as an SSH connection. For convenience to the local player, an LED turn indicator duplicates this information. These two functions were uncoupled from each other. All functions regarding interactivity are chessboard agnostic to ensure independence as will be described in the next section.

3.2. FR$_2$ Maintain state of chessboard according to chess rules

As previously mentioned, an internal matrix is constructed that is a representation of the board state. This representation would be heavily affected by FR$_1$ except that it is decoupled using an abstraction barrier in the form of an Application Programmer Interface (API). The user interface routines call matrix-representation functions for state changes. The matrix routines call robotic functions to implement the changes in the physical board. Each of these functions can be easily changed modify the data representation without affecting the calling routines. This underlying representation has an understanding of chess rules, especially legal moves and when a piece has been captured. When a command has been given for a move, the difference in board states is determined, which then produces a queue of operations to be applied (`updatePos()`). If this move results in a capture, `capture()` indicates the destination for the displaced piece to be the "graveyard", a separate bin on the side. This small connection between updating state and capturing a piece results in a decoupled design as shown in Eq. 2

$$\begin{Bmatrix} \mathrm{FR}_{2.1} \\ \mathrm{FR}_{2.2} \\ \mathrm{FR}_{2.3} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & X & X \end{bmatrix} \begin{Bmatrix} \mathrm{DP}_{2.1} \\ \mathrm{DP}_{2.2} \\ \mathrm{DP}_{2.3} \end{Bmatrix} \tag{2}$$

3.3. FR$_3$ Move chess piece in X-, Y-, and Z-axis

Implementation of chess piece movement in the physical realm requires translating the abstract board layout into robotic arm movements. The MeArm vendor provided a custom library "IK" to translate x, y, z coordinates into robot poses. It was discovered that the library did not perform as advertised. The z-value changed non-linearly for each chess board position, as did the x and y values. This surprise coupling between the Cartesian coordinates created a large amount of complexity. The performance of the arm was repeatable enough that the developers were able to overcome this complexity by calibrating each board position individually and building a map of the inaccurate values. FR$_3$ was not decomposed, so no design matrix was generated.

3.4. FR$_4$ Pick-up/put-down chess piece

The design of the end-effector was heavily influenced by the robotic arm constraints as discussed in Section 2.1 A number of experiments were performed to find the minimal information content geometries for piece manipulation.

The main concern was to ensure the magnetic force was strong enough to be able to pick up a single piece but also light enough to be handled by the MeArm. The size constraint of the chessboard defined the piece separation so X-Y expansion could not be used to improve selectivity. Experiments with a commercially available 9 mm diameter neodymium magnet determined 10 mm of clearance between the magnet and the top of the piece resulted in a reliable release. With the existing end-effector servo and linkage, a 50-degree rotation provided this magnet translation.

To minimize strain on the servomotor and conserve energy, the distance between the magnet and metal cap needed to be further controlled during operation. If the cap was too close to the magnet, it took a lot of torque to release it; a spacer chosen to keep the magnet distance consistent When the gap was 2.5 mm thick, the piece was held securely without straining the motor: this became the tube's bottom dimension. During the pickup operation, the servo lowers the magnet to the bottom of the tube and allows a small amount of slack. This means that the servo can be inactive at that time. Even if it is active, the slack prevents small "jerks" that occur often in the RC servos from causing an accidental release. The ideal distance from the piece for the tube to during pick was determined to be between 3–5 mm. Having a small gap between the pieces and the tube allows for unintended servo movement without knocking the piece over or those around it. In addition, there was a concern that pieces with flat top surfaces might cantilever or shift sideways during pick-up. The simple solution was to have a small contact point to allow gravity to pull the piece plumb. This results in a need to have a rounded top on the chess pieces.

The original MeArm included a small servo-actuated gripper. The gripper mechanism was removed, but the servo was used in the final end-effector design as previously described. At full extension, the arm could only support a mass grasped with the provided gripper. The new end-effector needed to be no more than the gripper's weight to ensure that payload was not significantly reduced. The custom tube was 3D printed and consisted of the geometries previously determined. The magnet is the heaviest component of the end-effector assembly but only 2.9 ± 0.1 g in mass, below the mass of the previous gripper.

The only source of coupling is due to $FR_{4.1}$ and $FR_{4.3}$ being dependent upon height of the piece. This decoupled design matrix is shown in Eq. 3.

$$\begin{Bmatrix} FR_{4.1} \\ FR_{4.2} \\ FR_{4.3} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ X & 0 & X \end{bmatrix} \begin{Bmatrix} DP_{4.1} \\ DP_{4.2} \\ DP_{4.3} \end{Bmatrix} \tag{3}$$

4. Conclusion

ADT has proved quite helpful in the Chessmate project for understanding the interaction between the elements of hardware and software. The independence axiom was a constant reminder of proper abstraction design to prevent the user interface and robot control elements from becoming non-modular. The information axiom guided the development of the end effector to ensure reliable lifting (FR_4) and placement (FR_3) of pieces. Functional Requirements enabled systematic implementation improving the chances of on-time completion of a project. Chessmate is ready to play!

4.1. Future Work

For a complete chess telepresence experience, additional work needs to be performed on the UI to make it more intuitive for remote and local players. One concept would be to leverage video game matchup websites designs such as `battle.net` to develop an equivalent robotic chess matchup site. Players would pick an opponent (which might even be an AI) to play against on Chessmate.

It is well known that serious chess players use the standard shapes to get a feel for the condition of the board. With the current pieces, this is not possible; further investigation is needed on how to make the more standard chess piece geometries compatible with a magnetic pickup mechanism.

Acknowledgments

Special thanks also go to Mechatronics 1 TA Björn Breiðfjörð Gíslasson who was always ready to answer questions and help at a moments notice. We are especially thankful to our Ben Gray of MeArm Robotics for prompt help with the MeArm. We must thank our good friend Björn Þór Sigurðsson helped us design the circuit for the MeArm. We wish to thank Halldór Guðfinnur Svavarsson for 3D printing the end-effector and Ingi Níels Karlsson for helping us with the laser printer. Last but not least, we wish to thank our colleagues for their moral support.

References

[1] Suh NP. The Principles of Design. Oxford University Press; 1990.
[2] Schaffer S. Automata and the Proto-Industrial Ideology of the Englightement – History. In: Clark W, Golinski J, Schaffer S, editors. Enlightened Automata. Chicago University Press; 1999. p. 126–165.
[3] maxjus. How to Build an Arduino Powered Chess Playing Robot [Internet]; 2016 [cited 2016 Mar 31]. Available from: `http://www.instructables.com/id/How-to-Build-an-Arduino-Powered-Chess-Playing-Robo`.
[4] Thornhill T. Robot chess set allows players to physically make their move from different sides of the world. Daily Mail. 2012 Jun 28;Available from: `http://www.dailymail.co.uk/sciencetech/article-2126184/Robot-chess-boards-allow-players-physically-make-moves-different-sides-world.html`.
[5] Chhangani MA. Arduino based Wireless Powered Chess. International Journal of Innovative REsearch in Computer and Communication Engineering. 2015 Apr;3:3187–3194.
[6] yousifnimat. Chess Playing Robot [Internet]; 2014 [cited 2015 Aug 30]. Available from: `http://www.instructables.com/id/Chess-Robot/`.
[7] Hatch D. Chess Playing Robot [Internet]; 2014 [cited 2015 Nov 02]. Available from: `http://www.chessplayingrobot.com/index.html`.
[8] Suh NP. Complexity. Oxford University Press; 2005.
[9] Suh NP. Axiomatic Design - Advances and Applications. Oxford University Press; 2001.
[10] Altshuller G, Clarke DW. 40 Principles: TRIZ Keys to Technical Innovation. Technical Innovation Center, Inc.; 2005.
[11] Brown T. Design Thinking. Harvard Business Review. 2008 Jun;86(6):85–92.
[12] Liu A, Lu SCY. A new coevolution process for conceptual design. CIRP Annals – Manufacturing Technology. 2015;64:153–156.
[13] Thompson MK, Foley JT. Coupling and Complexity in Additive Manufacturing Processes. In: 8th International Conference on Axiomatic Design (ICAD 2014). vol. 33. CIRP. Lisboa, Portugal: Axiomatic Design Solutions, Inc.; 2014 Sep. 24–26. p. 177–182.
[14] Bragason G, Þorsteinsson S, Karlsson RI, Grosse N, Foley JT. Heat-activated Parachute Release System. In: 9th International Conference on Axiomatic Design (ICAD). vol. 34. Procedia CIRP. Florence, Italy: Elsevier ScienceDirect; 2015 Sep 16–18. p. 131–136.
[15] Sölvason GO, Foley JT. Low-cost spectrometer for Icelandic chemistry education. In: 9th International Conference on Axiomatic Design (ICAD). vol. 34. Procedia CIRP. Florence, Italy: Elsevier ScienceDirect; 2015 Sep 16–18. p. 156–161.
[16] Jónsson BL, Garðarsson GO, Pétursson O, Hlynsson SB, Foley JT. Ultrasonic gasoline evaporation transducer — reduction of internal combustion engine fuel consumption using axiomatic design. In: 9th International Conference on Axiomatic Design (ICAD). vol. 34. Procedia CIRP. Florence, Italy: Elsevier ScienceDirect; 2015 Sep 16–18. p. 168–172.
[17] Suh NP. Challenges in dealing with large systems. In: 9th International Conference on Axiomatic Design (ICAD). vol. 34. Procedia CIRP. Florence, Italy: Elsevier ScienceDirect; 2015 Sep 16–18. p. 1–15. Keynote.
[18] Cochran DS, Li J, Vanover K, Foley JT. A System Design of a Rural Hospital Operating Room. In: 26th CIRP Design Conference. Stockholm: CIRP; 2016. p. 6.
[19] Foley JT, Harðardóttir S. Creative Axiomatic Design. In: Proceedings of the 26th CIRP Design Conference. Stockholm; 2016 Jun. 15–17. p. 6.
[20] Cochran DS, Foley JT, Bi Z. Use of the Manufacturing System Design Decomposition for Comparative Analysis and Effective Design of Production Systems. International Journal of Production Research. 2016;p. 24.
[21] Benavides EM. Advanced engineering design - An integrated approach. Woodhead Publishing; 2012.
[22] Thompson MK. A classification of procedural errors in the definition of functional requirements in Axiomatic Design theory. In: 7th International Conference on Axiomatic Design (ICAD 2013). vol. 32. CIRP. Worchester, MA: Axiomatic Design Solutions, Inc.; 2013. p. 1–6.
[23] Procedia CIRP. 9th International Conference on Axiomatic Design (ICAD). vol. 34. Florence, Italy: Elsevier ScienceDirect; 2015 Sep 16–18.