INTERNATIONAL DESIGN CONFERENCE - DESIGN 2010
Dubrovnik - Croatia, May 17 - 20, 2010.

DESIGN 2010

CHARACTERISING THE IMPACT OF LEGACY ARCHITECTURES ON COMPLEX PRODUCTS

D. F. Wyatt, D. C. Wynn and P. J. Clarkson

Keywords: product architecture, incremental design, constraints, computational design synthesis

1. Introduction

The architecture of an engineering product has a strong influence on its quality, across many different lifecycle stages. For instance, Ulrich states that product architecture can influence product change, product variety, component standardisation, product performance and product development management [Ulrich, 1995]. In principle, it is therefore important to tailor a product's architecture to the requirements placed on it. However, many complex products are designed incrementally, based on previous versions [Eckert et al. 2004] – carrying over elements of their architecture. This paper attempts to investigate how such carry-over of "legacy" architectures can affect the quality of new designs, using computational design synthesis techniques to simulate real design.

Product architecture has been defined as "the scheme by which the function of a product is allocated to physical components" [Ulrich 1995]. Despite its importance, the qualitative nature of product architecture means that it is not easy to define a "design space" of architectures, in comparison with numerical design problems. Therefore, it may be difficult to identify alternatives to a given architecture, even if the alternatives are superior.

This is particularly problematic in incremental design, where parts of the design of a complex product may have remained unchanged across multiple generations, spanning extended periods of time. In many cases such design carry-over may be intentional, due to the complexity of the product and the resulting risk and cost of modifying it. However, with time such design decisions can become "fossilised" into "fictitious constraints" [Pahl & Beitz 1996] on the form of a solution. Many design approaches, such as Systematic Design, specifically suggest looking beyond such "fictitious constraints", but without considering that there may be underlying reasons for the constraints [Pahl & Beitz 1996]. In addition, from a psychological point of view, design has been demonstrated to be subject to "fixation" effects [Purcell & Gero 1996], where knowledge of a past solution to a problem makes it cognitively more difficult for designers to generate new solutions. Such effects may further hinder designers from thinking beyond known architectures.

Both rational architecture carryover and fictitious constraints were observed in a case study of diesel engine design [Wyatt et al. 2009a]. For the former, the interviewees in the case study (engineers involved with the new product development process within the case study company) described situations where specific engine components were intentionally reused from previous designs to reduce development time, cost and risk; however, one of the interviewees also explained that design relied on a "shared vision" of the engine derived from previous versions, suggesting a route for unintentional carryover. There was also evidence that standardisation of business processes in the company may reduce architectural innovation.In order to provide guidance to designers on the importance of these effects, this paper investigates the effect of architecture carry-over on the quality of a product. The remainder of this paper is structured as follows. Section 2 states the research

questions investigated in this paper, and justifies the use of computational design synthesis to investigate the effect of architecture carry-over. Section 3 explains the specific computational design synthesis method employed. Section 4 presents the example design problem, the experiments carried out, and the results obtained. Section 5 discusses the results in the context of the research questions and suggests directions for future research. Finally, section 6 draws brief conclusions.

2. Research questions and approach

2.1 Research questions

This paper aims to answer two research questions:

How does incremental design (using a previous design as the basis for a new product architecture) affect the achievable product quality, compared with original design (designing without preconceived ideas of the solution's form)?

If incremental design does limit achieveable product quality, can the severity of the limits be reduced by allowing greater design freedom?

2.2 Approach

The most accurate answers to the research questions posed above would be obtained through empirical studies of real design processes. However, it is difficult to obtain useful results from such studies, due to the duration of real design processes, their uniqueness, and the difficulty of precisely measuring and controlling cognitive parameters such as design freedom. This paper therefore takes a simulation approach, modelling real-life engineering design using computational design synthesis.

Table 1. The aspects of computational design synthesis used to simulate aspects of real engineering design, and their limitations

Engineering design	Simulation	Potential limitations of model
Product architecture design	Constraint-based network synthesis	*Computational methods do not reflect cognitive processes in design, and are fundamentally naïve. Thus, the range of results may not explore the full range of possibilities and/or may not respect actual constraints, depending on how the synthesis is set up. However, if the design problem is correctly formalised the results from computational synthesis should be equivalent to those that could be found by conventional methods.*
Wide range of design problems	Single design problem based on a real example	*The results from the example problem may be of limited validity, especially for problems of significantly greater or lesser complexity.*
Original design	Starting synthesis from an empty product architecture model	*Even in original design, engineers may have some insight into the structure of the solution based on their previous experience.*
Incremental design	Starting synthesis from a model of an existing product's architecture	*This may be too rigid a representation of what is reused in incremental design.*
Varying degree of design freedom	Varying maximum backtrack depth in synthesis	*This is a simple model of "design freedom". In particular, some "backtrack" steps may be more costly (thus less likely to be undertaken) than others.*
Design quality	Structural metrics applied to architectures	*These metrics are not direct measures of top-level business considerations, e.g. profit or sales. However, the metrics used are intended to capture some of engineering-related costs of developing a product.*

Computational design synthesis involves the use of computational methods to solve problems in design. It is distinguished from optimisation by the relatively unstructured problems to which it is applied: in optimisation the design representation is frequently a finite-dimensional vector of real

numbers, whereas in computational design synthesis the representation may consist of more complex information structures. Computational design synthesis has successfully been applied to engineering problems ranging from structural design of trusses [Shea et al. 1997] to design of robots and their controllers [Lipson & Pollack 2000]. Such methods have a variety of advantages over conventional design: they are able to approach problems systematically, without suffering from cognitive biases or fixation, and generate large numbers of potential solutions rapidly.

However, the use of computational methods has the additional advantage of allowing full control over the process of finding design solutions. Thus, it is possible to compare objectively the results obtained from different "parameter settings"; if the parameter settings are chosen to correspond to different design approaches, insights may be gained into the behaviour of design practice.
In this work, we use constraint-based network synthesis [Wyatt et al. 2009b] to generate product architectures. The strength of this method is that it can model both original design, by starting synthesis from an empty architecture, and incremental design, through starting from a model of an existing product. The amount of "design freedom" allowed in incremental design can also be modelled using the amount of "backtracking" from the synthesis start point, i.e. the number of elements in the start point that may be removed before "forward" synthesis; increasing the amount of "backtracking" will lead to architectures that are less similar to the start point. The architectures resulting from synthesis are evaluated for "quality" using structural metrics of the networks of which they consist. Table 1 summarises how different aspects of computational design synthesis are used to simulate real engineering design, and some of the limitations inherent in doing so; the limitations are discussed further in subsection 5.2. The following section gives more detail about the product architecture synthesis method (subsections 3.1 and 3.2) and the metrics used for evaluating architectures (subsection 3.3).

3. Using computational design synthesis to explore spaces of architectures

3.1 Representing product architecture design problems

In the computational design synthesis method used in this paper, a specific product architecture is represented as a network (or *graph*) consisting of *nodes* (components) linked by *edges* (physical connections or other relations). Both components and connections have *types* assigned to them. This makes it possible to indicate different categories of element and show where components or connections are equivalent within the scope of product architecture design (although, for example, the detailing of one "injection-moulded plastic component" may be very different from that of another). Connection types may be directed or undirected, according to whether the relevant connections are symmetric (e.g. "attached to") or asymmetric (e.g. "contains"). Both component and connection types can be hierarchically organised, and a given type can have multiple "parents". Only the types at the lowest level of the hierarchy are "concrete" (i.e. instantiable), while the others are abstract (i.e. not instantiable). The types of connections/relations modelled are at the discretion of the modeller, but may include structural, behavioural, geometrical and/or assignment relations.
Using this representation, a particular product architecture can be viewed as a point in a "space of product architectures" that includes the architectures of other "equivalent" products. In this context, "equivalence" can be interpreted more or less broadly: the space of architectures for the set "products that clean floors" contains more possibilities than the space for the set "cylinder vacuum cleaners", which is in turn larger than the space for the set "cylinder vacuum cleaners made by [Company X]".
The extent of a space of architectures is defined by the different arrangements of components and connections, but not every arrangement of components and connections "makes sense". More formally, an arrangement of components and connections is defined to be "realisable" if and only if it forms a model of the product architecture of a product that could fulfil the desired function. The set of "realisable" product architectures can then be modelled using constraints on the components and the connections between them. In this method, four types of constraints are used:

- Component number constraints (CNCs) indicate how many components of a given type may be present in a design.

- Direct connection constraints (DCCs) indicate which component types may be connected together by which connection types. In addition, they state the required cardinality of the connection – how many components of the second type may be attached to a component of the first type, and vice versa.
- Fan-out constraints (FOCs) indicate how many connections of a certain type that components of a certain type may have in total (incoming and/or outgoing, as appropriate).
- Indirect connection constraints (ICCs) indicate that there must be a continuous path from every component of one type to every component of another type, of specified connection type(s), and optionally including other components en route.

These constraints may refer to component and connection types at any level in the appropriate hierarchy, allowing for both "OR" and "AND" constraints. Numerical aspects of these constraints are specified as ranges with a minimum and a maximum for the quantity concerned. The set of component types, connection types and constraints for a particular problem is collectively termed a "schema".

Figure 1 shows a graphical representation for the entities and relations used in this method. The lower section of the diagram shows components and connections that define a particular architecture. Components are represented as cubes, while connections are represented as concentric circles; arrows link the component at the "source" end of the connection to the connection itself, and the connection to the "sink" component. The directionality of a connection is recorded in the corresponding connection type. Components and connections are linked by arrows to (concrete) component and connection types in the top section of the diagram. Since models of specific architectures often appear on a separate diagram from the schema, in the figure the components and connections are linked back to their types using hyperlinks, small white numbered circles (hyperlinks with matching numbers indicate the two ends of the same link). The top section also shows the constraints that, together with component and connection types, define a schema. Component number constraints are by default incorporated into component types; other constraint types are linked by arrows to the relevant component and connection types. Numerical ranges use "*" to indicate an unlimited upper bound. For instance, the direct connection constraint in Figure 1 states that every component of type "Component Type 1" must have exactly 1 connection of type "Connection type" to a component of type "Component type 2", but the cardinality in the reverse direction is unrestricted. In this case, the architecture in the lower part of the diagram satisfies all the constraints.

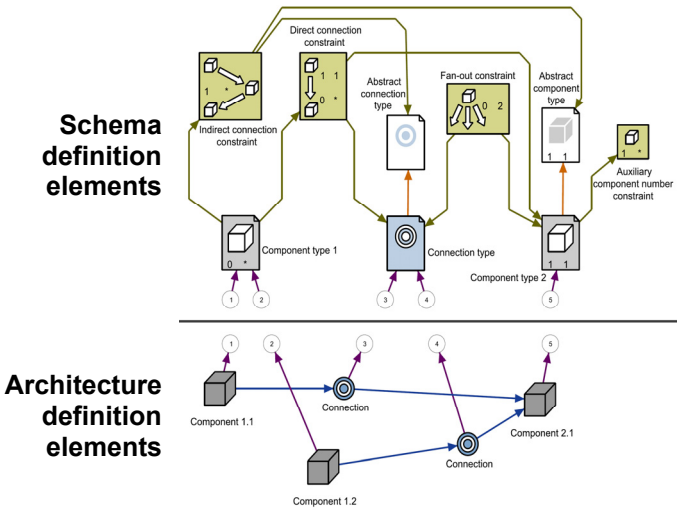

Figure 1. Elements of the graphical representation for architectures and schemas

3.2 Computational design synthesis algorithm

If represented in the form described in the previous subsection, alternative product realisable architectures may be synthesised using computational methods. The component types and connection types in a schema define a set of potential architectures (each of which may or may not be realisable). This set forms a *state space*, since any architecture can be reached from any other architecture

(including the "empty" architecture containing no components or connections) by sequences of elementary operations of four types:

- Removing an existing connection;
- Removing an existing component (in which case all connections associated with it must also be removed);
- Adding a component of one of the types defined in the schema;
- Adding a connection of one of the types defined in the schema, between two existing components.

Synthesising realisable architectures thus involves searching this state space to find goal states – architectures that satisfy all the constraints in the schema. In practice, since there may not be restrictions on the number of components in an architecture, the state space may be infinite – thus it is necessary to specify a maximum search depth for each type of elementary operation (a maximum number of those operations to allow when carrying out the search).

State space search problems can be solved exhaustively using systematic search methods [Russell & Norvig 2003]. Alternatively, stochastic methods can be used to sample the space of architectures: a random number of random elementary operations up to the specified maximum search depth are performed, after which the resulting architecture is tested against the constraints. This is less efficient than systematic search, but in the case of large problems the overhead of systematic search can become prohibitive and outweigh its advantages in efficiency.

Depending on the specific problem, either the complete set of elementary operations may be used or a subset may be chosen. For instance, if it is desirable to conserve the set of components in the architecture, synthesis may be restricted to only adding and removing connections.

The starting point for the state-space search may be:

- The minimal set of components that satisfies the CNCs, in the case of original design; or
- A model of an existing architecture, potentially modified before synthesis by manually removing components or connections, in the case of incremental design.

3.3 Evaluation of generated architectures

Once generated, the architectures may be evaluated objectively using metrics to support the designer in choosing which should be taken forward to detailed design..

Table 2. Metrics used to evaluate product architectures

D-complexity	M-complexity	C-complexity
$D(A) = \frac{1}{n}\sum_{i=1}^{n} z_i$	$M(A) = n + \frac{1}{2}\sum_{i=1}^{n} z_i$	$C(A) = \frac{n(n-1)}{2} \Big/ \sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}$
Components with more interfaces have been shown to require more redesign work on average [Sosa *et al.*, 2005]. Thus, a metric for the design complexity, *D(A)*, of an architecture *A* containing *n* components is: *the mean of the number of connections per component z_i for all components i in A*.	The manufacture cost of a product may be divided into the manufacture cost of its components and the cost of assembling them. Without additional information about the relative costs of parts and assembly operations, a metric for the manufacture complexity, *M(A)*, of an architecture *A* containing *n* components is: *the sum of the number of components n and the number of connections per component z_i for all components i in A (halved to avoid double counting)*.	The likelihood of change propagating between two components in a product is inversely related to the graph distance between the two components in the architecture [Keller *et al.*, 2006]. Therefore, a metric for the change complexity, *C(A)*, of an architecture *A* containing *n* components is: *the reciprocal of the mean of the graph distances d_{ij} between all pairs of components i,j in A*.

A metric is defined as a function that assigns a numerical value to an architecture; by convention, "better" architectures return smaller numerical values. In this paper, architectures are evaluated using three graph-theoretical metrics, similar to those used by Lindemann [Lindemann et al. 2008], corresponding to different aspects of quality important to the product's manufacturer: the complexity

of designing the product (D-complexity), complexity of manufacture (M-complexity) and the complexity of making changes to it (C-complexity). These metrics are described in Table 2

4. Assessing the impact of legacy architectures on new designs

4.1 Design problem used in computational experiments

To investigate the influence of legacy architectures, the synthesis method described in section 3 was applied to a realistic design problem: redesigning a domestic "cylinder" or "canister" vacuum cleaner. Until recently, most domestic vacuum cleaners used physical filtration to separate dust from the stream of air, either in the form of a reusable cloth filter or a disposable bag. However, in recent years vacuum cleaners have begun to be equipped with cyclonic dust separators which use centrifugal effects in a "cyclone" of rapidly-rotating air to separate and collect the denser dust. Such technology has a number of advantages and is attractive to consumers: thus, many vacuum cleaner manufacturers who formerly made filtration-based products have brought out models that use cyclonic dust separators. This example considers the design of the product architecture for a cyclone-based cylinder vacuum cleaner, and the effect of basing it on the architecture of an earlier filtration-based cylinder vacuum cleaner. Such a problem is an example of "technology infusion", where a new technology is added to a product class of which instances have already been designed using a previous technology, and is of a level of complexity similar to design problems that might be encountered in practice.

In order to construct a schema and "starting point" architecture for this research, 4 vacuum cleaners were acquired and dismantled to gain an understanding of the space of possible architectures, as described in [Wyatt et al., 2009b]. One in particular, the Sanyo SC-N200 (a filtration-based cylinder vacuum cleaner), was chosen to represent the original product to be modified through incremental design. The representation described in subsection 3.1 was then used to construct a model of the SC-N200 architecture and a schema for the problem, shown in Figure 2 and Figure 3 respectively (using the graphical notation of Figure 1).

In principle, there are many ways in which the architecture of a vacuum cleaner can vary – from the order in which air flows through different components to the way in which the cable is stored. In this case, to keep the problem reasonably tractable it was assumed that only the architecture of the front section of the vacuum cleaner base unit requires redesign to accommodate the cyclone; the design of the architecture of the hose and tools on the one hand, and the motor and power supply on the other, were assumed to be separable from the cyclone incorporation. Thus the air flow starts at the "Hose socket" and finishes at the "Rest of vacuum cleaner". The "bag" component was removed from the model used as the synthesis start point, and to avoid unnecessary computational expense a "cyclone" component was connected into the air flow path in its place.

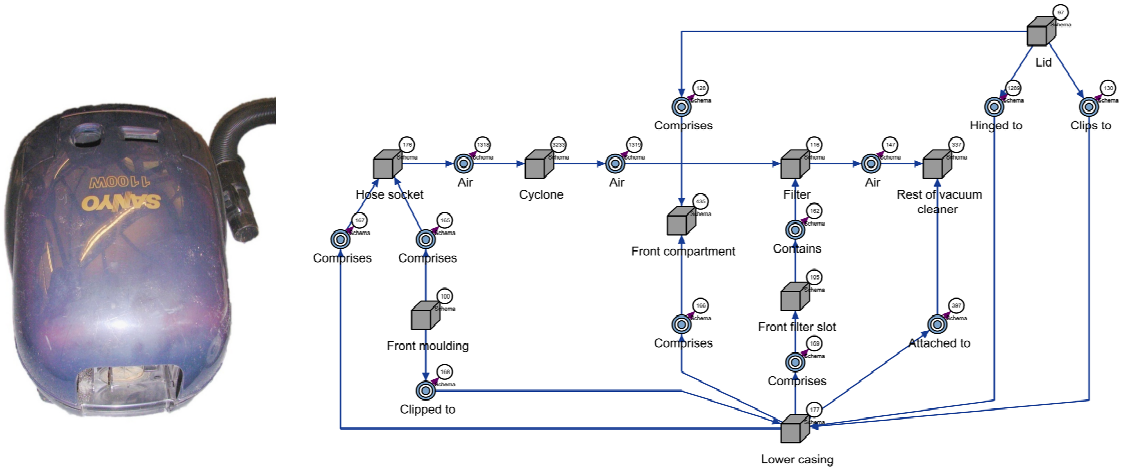

Figure 2. (Left) A picture of the Sanyo SC-N200. (Right) The starting architecture used for the experiments

However, the "cyclone" component represents the circulating volume of air itself, and must be "comprised" by at least two "mouldings" (injection-moulded plastic components); the synthesis of this arrangement was left as the main problem for the computational methods to solve. Even this relatively constrained architecture design problem required significant effort to formalise, as shown by the complexity of Figure 3 for instance, while most of the component types correspond to physical entities, the "Compartment" component type was found necessary to model the airtight sections of the vacuum cleaner.

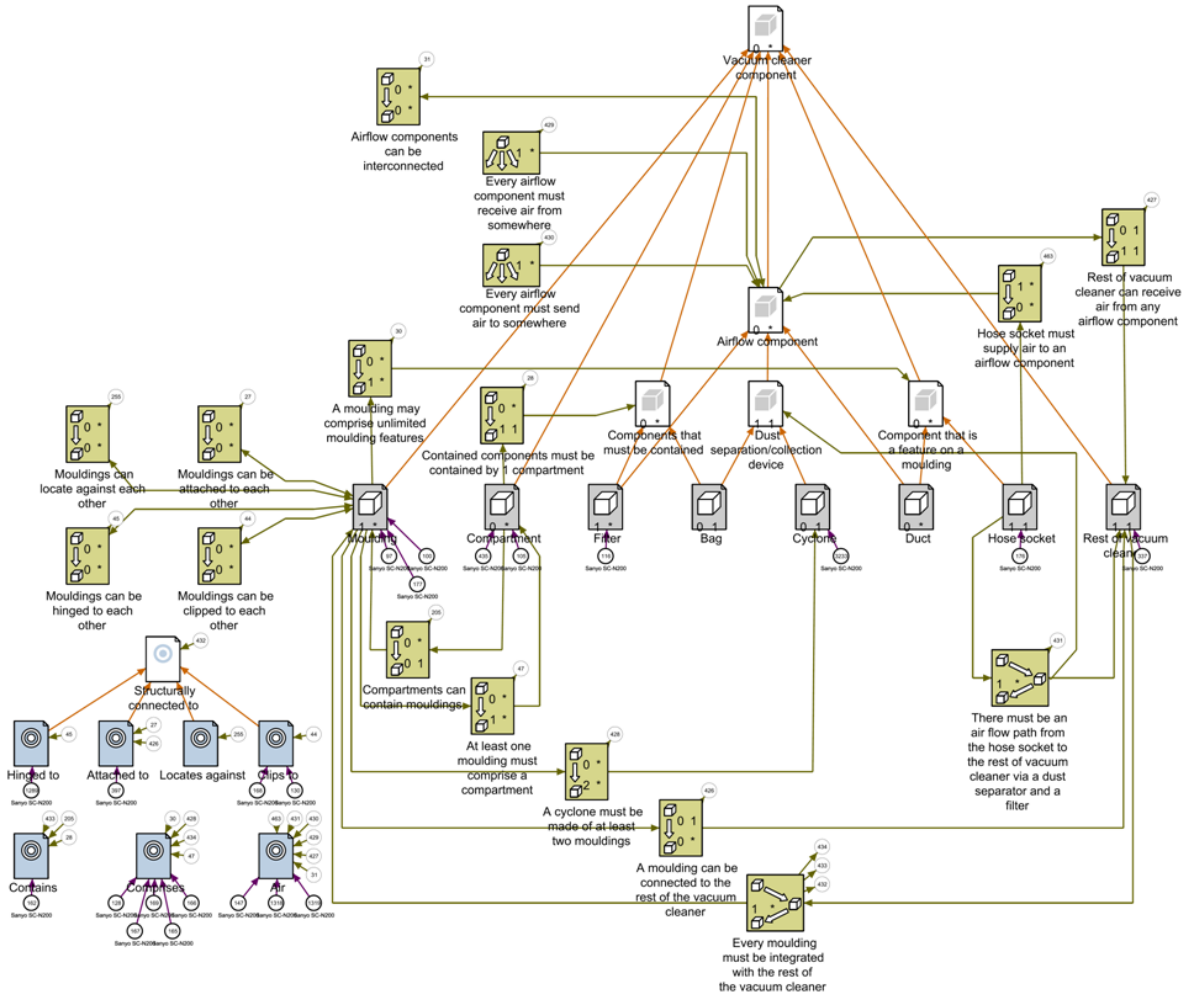

Figure 3. The schema used for the experiments

4.2 Experiments performed

Four experiments were carried out, as shown in Table 3. In experiments 1-4, simulating incremental design with increasing amounts of design freedom, synthesis started from the SC-N200 model shown in Figure 2 (with 9 components and 13 connections), with amounts of backtracking ranging from none in experiment 1 to removal of up to 3 components and 6 connections in experiment 4. Experiment 5, simulating original design, used synthesis from a "minimal" start point – in this case, a 4-component architecture consisting of 1 "Moulding", 1 "Filter", 1 "Hose socket" and 1 "Rest of vacuum cleaner".

The maximum forward search depths were chosen based on an architecture observed in one of the other vacuum cleaners dismantled during the composition of the problem, which had 11 components and 18 connections. To allow for greater variety in the generated designs, the overall maximum component and connection numbers were defined to be 1 greater than this size (i.e. 12 components and 19 connections). Where synthesis was carried out with no backtracking (runs 1 and 5), the search depths were therefore set to allow architectures up to this size. In runs 2, 3 and 4 varying amounts of backtracking were allowed, and the forward search depths were increased accordingly.

The software used to perform the experiments was a Java implementation of the computational design synthesis method described in section 3, using the CAM (formerly P3) platform [Wynn et al. 2009]. CAM provides a user interface for constructing general network models, with an extensible plug-in architecture to allow analysis functionality to be added to the software. The categories of element described in subsection 3.1 and the graphical notation shown in Figure 1 were specified using a meta-model, while the synthesis and evaluation algorithms were implemented as executable plug-ins.
Stochastic synthesis was used in all cases, as the scale of the problem made systematic synthesis impractical. 500,000 synthesis attempts were performed for each experiment. Experiments 1-4 each took approximately 10 hours on a workstation PC, while experiment 5 took approximately 22 hours.

Table 3. Synthesis experiments performed

#	Synthesis start point	Maximum search depth				Number of synthesis attempts	Number of architectures produced
		Removing components	*Removing connections*	*Adding components*	*Adding connections*		
1	SC-N200	0	0	3	6	$5x10^5$	1101
2	SC-N200	1	2	4	8	$5x10^5$	368
3	SC-N200	2	4	5	10	$5x10^5$	296
4	SC-N200	3	6	6	12	$5x10^5$	292
5	Minimal	-	-	8	19	$5x10^5$	835

4.3 Results

The synthesised architectures were evaluated using the metrics described in subsection 3.3. Table 4 shows the minimum, maximum, mean and standard deviation of the values of each metric for each experiment; these are also plotted in Figure 4. Figure 5 shows three-dimensional plots of the results from each experiment.

Table 4. The ranges (minimum, maximum, mean and standard deviation) of D-, M- and C-complexity of the architectures found in each experiment

#	Synthesis start point	D-complexity				M-complexity				C-complexity			
		Min	Max	Mean	Standard deviation	Min	Max	Mean	Standard deviation	Min	Max	Mean	Standard deviation
1	SC-N200	1.55	1.90	1.82	0.09	26	31	28.5	0.9	0.33	0.38	0.36	0.01
2	SC-N200	1.40	2.00	1.81	0.15	22	29	25.7	1.3	0.32	0.39	0.37	0.01
3	SC-N200	1.25	2.25	1.94	0.18	18	31	26.0	2.2	0.30	0.41	0.38	0.01
4	SC-N200	1.38	2.57	2.08	0.24	19	34	26.7	3.1	0.32	0.43	0.38	0.02
5	Minimal	1.25	2.71	2.19	0.30	14	29	24.2	2.6	0.32	0.43	0.40	0.02

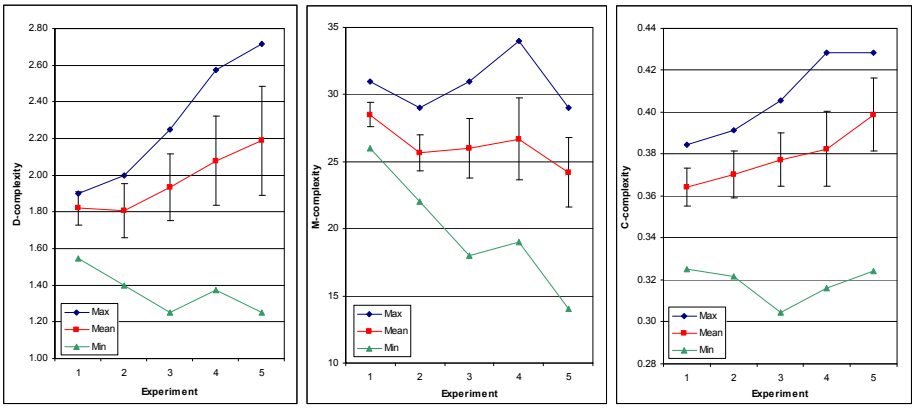

Figure 4. The variation of D-complexity, M-complexity and C-complexity in the synthesis experiments. Error bars on the mean series show the standard deviation

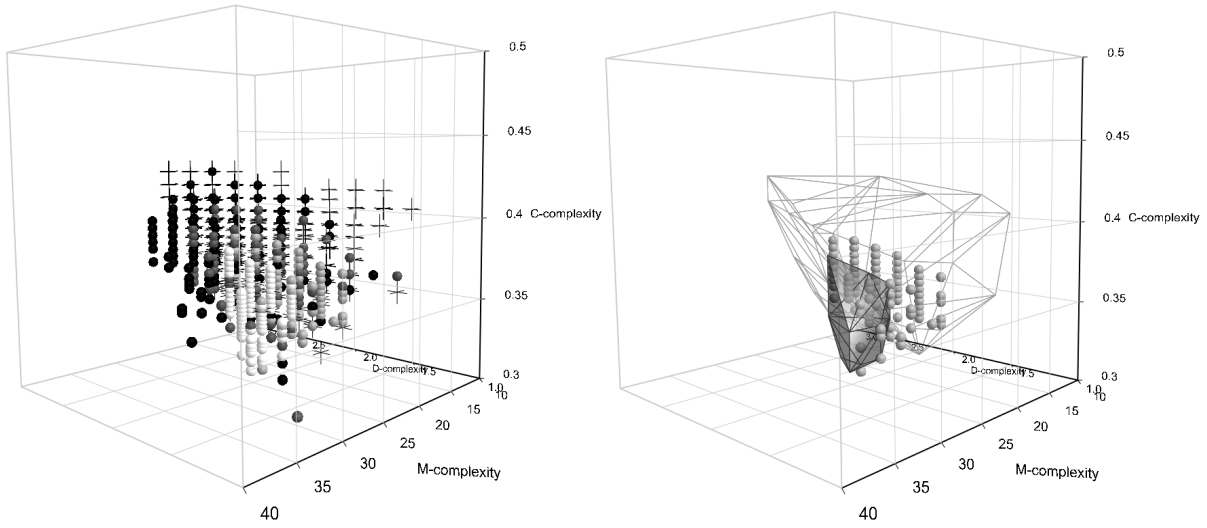

Figure 5. Three-dimensional plots of the D-complexity, M-complexity and C-complexity of the architectures from each experiment. (Left) All results; spheres indicate experiments 1-4, light grey for experiment 1 shading to dark grey for experiment 4; black crosses indicate experiment 5. (Right) Polyhedra show convex envelopes of experiment 1 results (shaded, black wireframe) and experiment 5 results (grey wireframe), and spheres show all individual results from experiment 2 (grey spheres)

4.4 Analysis of results

For the case of synthesis starting from a given architecture, Table 4 and Figure 4 show that the ranges of metric values exhibited by the synthesised architectures usually increase with increasing backtrack depth. In particular, when comparing experiment 4 (greatest backtrack depth) with experiment 1 (smallest backtrack depth), there is an 11% improvement in minimum D-complexity (1.38 vs. 1.55), a 27% improvement in minimum M-complexity (19 vs. 26), and a 3% improvement in minimum C-complexity (0.32 vs. 0.33). In addition, the range of D-complexity is more than 3 times larger in experiment 4 than experiment 1 (1.38-2.57 vs. 1.55-1.90), the range of M-complexity is 3 times larger (19-34 vs. 26-31), and the range of C-complexity is more than double the size (0.32-0.43 vs. 0.33-0.38). This can also be seen graphically in Figure 5: in the left-hand plot the black spheres (results of experiment 4) are much more widely dispersed than the white spheres (experiment 1), and the right-hand plot shows that adding a single backtracking step between experiment 1 and experiment 2 allows a much wider range of results. In addition, the left hand plot suggests that increasing the backtrack depth increases the range of architectures found "anisotropically" (increasing variety without favouring particular types of architecture). While the boundaries of the metric ranges do not expand monotonically from experiment 1 to experiment 4, in the case of the minimum values for all metrics and maximum M-complexity, this may be due to the stochastic nature of the synthesis and the decreasing number of successful results as the search space expands (1101 in experiment 1 down to 292 in experiment 4).

Based on the trends observed in experiments 1-4, it would be expected that synthesis "from scratch" would be the least constrained out of all the experiments, and thus would explore the widest range of architectures and return the widest range of metric values. This is reasonably well supported by the results of experiment 5: the minimum value of D-complexity from the results of experiment 5 (1.25) equals the minimum from experiments 1-4, and the minimum M-complexity (14) is lower than those found in all other experiments; however, the minimum C-complexity (0.32) is only lower than that found in experiment 1. Again, Figure 5 shows graphically that the results of experiment 5 cover a wider range than the results of experiments 1 and 2. Although the ranges of values obtained from experiment 5's results do not encompass all the ranges from experiments 1-4, this may be due to the limited number of architectures obtained from the significantly larger search space to be explored in this experiment. In addition, the nature of the stochastic search algorithm unavoidably biases the

results towards architectures with fewer components and connections – thus the lower M-complexity values at both ends of the range are not unexpected, since this metric measures the number of components and connections in the architecture, while the long paths needed to obtain low C-complexity values are not as common in small architectures.

5. Implications, limitations and future work

5.1 Impact of legacy architectures on quality of synthesised architectures

The results reported in subsection 4.3 allow the research questions posed in subsection 2.1 to be answered as follows:

1. *To what degree does incremental design (using a previous design as the basis for a new product architecture) limit the achievable quality, compared with original design (designing without preconceived ideas of the solution's form)?*

- Comparing the results of experiment 1 (pure incremental design) with experiment 5 (original design) shows that basing an architecture on a previous design can indeed constrain the maximum achieveable quality.

2. *If incremental design does limit achieveable quality, can the severity of the limits be reduced by allowing greater design freedom?*

- Comparing the results of experiments 1-4 (increasing backtrack depth in incremental design) shows that increasing the degree of design freedom in incremental design can allow the maximum quality to increase to values comparable to those attainable through original design.

However, comparing the numbers of architectures synthesised in the different experiments highlights the fact that increasing the degree of design freedom (either within incremental design or by switching to original design) can increase the difficulty of finding solutions. Starting from an architecture that is known to be realisable facilitates finding other feasible architectures, although their quality may be reduced.

The results suggest that in product architecture design, in addition to the tradeoffs between competing design objectives, there is also a tradeoff between the benefits of architecture conservation [Wyatt et al. 2009a] and the quality of the design: while restricting the scope of redesign can reduce costs and risks of development, it can also reduce the achievable quality. The effects of this tradeoff should therefore be borne in mind when making decisions about the redesign of complex products.

5.2 Limitations and future work

As noted in Table 1, the results were obtained from a simulation of a real design situation with a number of limitations. Firstly, structural metrics were used to evaluate the resulting architectures as a proxy for "design quality". The structural metrics have the advantage that they may be calculated for a product architecture without further information about the architecture's constituent elements. However, although they theoretically capture aspects of the process used to design and construct the resulting product, they are not readily interpretable in concrete terms. Calculating concrete values from an architecture is challenging, since the architecture itself does not contain the information required to do so (thus values must be estimates). Nonetheless, an important future direction for this work is to investigate methods for estimating such concrete performance values in order to demonstrate effects in a measureable way.

In addition, the analysis in this paper has used backtrack depth within computational design synthesis as a proxy for "design freedom". In particular, this incorporates an assumption that increasing backtrack depth increases the maximum "distance" of the resulting architectures from the starting point. Although the results in subsection 4.3 suggest that the diversity of architectures increases with increasing backtrack depth, ideally this should be demonstrated explicitly, for instance by using graph edit distance as a similarity measure. Combined with concrete performance evaluation described in the previous paragraph, this would enable true quantification of the degree to which novelty allows better performance, as described in subsection 5.1.

While this work has considered the range of attainable metric values at different backtrack depths, it has not attempted to measure whether increasing the backtrack depth provides a "better tradeoff" between the metrics (although Figure 5 suggests that this is the case). This is important because in general there will be many objectives in a design problem and the overall solution will be a tradeoff between them; thus, to show that increasing the backtrack depth would improve quality in a real design problem, it is necessary to demonstrate that doing so improves not only the attainable values of individual metrics but also the tradeoff between them. This could be formally defined using concepts of Pareto dominance: if Set A provides a "better tradeoff" between a set of metrics than Set B, Set A's Pareto front with respect to those metrics should dominate Set B's Pareto front with respect to the same metrics. The quality of the tradeoff improvement could then be measured by the fraction of Set B's Pareto front dominated by Set A's Pareto front. Such a definition is objectively measurable, and carrying out this analysis represents another future direction for this research.

To improve the quality of the data used in this experiment, in particular the variation due to its probabilistic nature as noted in subsection 4.4, larger numbers of stochastic synthesis runs could be carried out. Alternatively, systematic search could be used (with potentially some form of decimation to reduce the fraction of the search space explored), or optimisation against one or more of the metrics, for instance using simulated annealing [Shea et al. 1997]. In addition, the formalisation of the design problem could be improved to allow for a wider range of solutions and to reduce the incidence of non-realisable architectures, and similar analyses could be conducted on other example problems to test generalisability.

Finally, there is the issue of how the insights gained from this analysis may be made more useful for design practice. Improving the concreteness and generalisability of the analysis, as described above, may allow estimation of rule-of-thumb values for the quantitative impact of legacy architectures on new designs. Such values may be used to improve decision-making when planning new product development. At a more detailed level, in cases where design effort is limited and must be deployed in the most efficient manner, applying the methods from this paper to the design problem in question may be able to identify the specific architecture changes that would give the greatest quality improvement.

6. Conclusions

Many new designs are based on previous products, especially at the product architecture level, due to the costs and risks of making changes to the design and cognitive difficulties in generating alternatives to known architectures. However, such reuse of legacy architectures can potentially impact the quality of the design of new products. This paper has investigated the potential impact of incremental rather than original design, and whether it can be mitigated by increasing "design freedom", using computational design synthesis of product architectures for a consumer vacuum cleaner. The results show that incremental design restricts the achievable "quality" (evaluated using structural metrics) compared with original design, but increasing the backtrack depth (increasing "design freedom") produces architectures that perform better against all metrics individually and potentially improves the overall tradeoff between metrics. Future work is needed to increase the concreteness of the analysis, by using technical or process performance evaluation and direct measurements of "distance" of architectures from the starting point, and to generate general insights and specific methods that may be used to guide decision-making during the design process.

Acknowledgements

This work was funded by the George and Lilian Schiff Studentship.

References

Eckert, C. M., Clarkson, P. J., Zanker, W., "Change and customisation in complex engineering domains", Research in Engineering Design, Vol.15, No.1, 2004, pp. 1-21.

Keller, R., Eckert, C. M., Clarkson, P. J., "Heuristics for Change Prediction", Proceedings of the International Design Conference (DESIGN 2006), 2006, pp. 873-880.

Lindemann, U., Maurer, M., Braun, T., "Structural Complexity Management: An Approach for the Field of Product Design", Springer, Berlin, Germany, 2008.

Lipson, H., Pollack, J. B., "Automatic design and manufacture of robotic lifeforms", Nature, Vol.406., 2000, pp. 974-978.

Pahl, G., Beitz, W., "Engineering Design: A Systematic Approach", Springer, London, UK, 1996.

Purcell, A. T., Gero, J. S., "Design and other types of fixation", Design Studies, Vol.17, No., 1996, pp. 363-383.

Russell, S. J., Norvig, P., "Artificial Intelligence: A Modern Approach", Prentice Hall, Englewood Cliffs, NJ, USA, 2003.

Shea, K., Cagan, J., Fenves, S. J., "A shape annealing approach to optimal truss design with dynamic grouping of members", ASME Journal of Mechanical Design, Vol.119, No.3, 1997, pp. 388-394.

Sosa, M. E., Agrawal, A., Eppinger, S. D., Rowles, C. M., "A network approach to define modularity of product components", Proceedings of the ASME Design Engineering Technical Conference and Computers and Information in Engineering Conference (IDETC/CIE 2005), 2005.

Ulrich, K., "The role of product architecture in the manufacturing firm", Research Policy, Vol.24, No.3, 1995, pp. 419-440.

Wyatt, D. F., Eckert, C. M., Clarkson, P. J., "Design of product architectures in incrementally developed complex products", Proceedings of the International Conference on Engineering Design (ICED'09), 2009a, pp. 167-178.

Wyatt, D. F., Wynn, D. C., Clarkson, P. J., "A computational method to support product architecture design", Proceedings of the ASME International Mechanical Engineering Congress and Expo (IMECE 2009), 2009b.

Wynn, D. C., Nair, S. M. T., Clarkson, P. J., "The P3 Platform: An approach and software system for developing diagrammatic model-based methods in design research", Proceedings of the International Conference on Engineering Design (ICED'09), 2009, pp. 559-570.

David F. Wyatt
PhD student
University of Cambridge, Department of Engineering
Trumpington Street, Cambridge, United Kingdom
Telephone: +44 (0)1223 748 564
Email: dw274@eng.cam.ac.uk
URL: http://www-edc.eng.cam.ac.uk/people/dw274.html