Fundamentals for the Design of Products for a Circular Economy

Examples from Software Engineering to Motivate Design of Physical Products

Clemons, Eric; Teilmann-Lock, Stina

Document Version
Final published version

Published in:
Proceedings of the 54th Hawaii International Conference on System Sciences

DOI:
10.24251/HICSS.2021.790

Publication date:
2021

Citation for published version (APA):
Clemons, E., & Teilmann-Lock, S. (2021). Fundamentals for the Design of Products for a Circular Economy: Examples from Software Engineering to Motivate Design of Physical Products. In *Proceedings of the 54th Hawaii International Conference on System Sciences* (pp. 6573-6582). Hawaii International Conference on System Sciences (HICSS). Proceedings of the Annual Hawaii International Conference on System Sciences https://doi.org/10.24251/HICSS.2021.790

Proceedings of the 54th Hawaii International Conference on System Sciences | 2021

Fundamentals for the Design of Products for a Circular Economy: Examples from Software Engineering To Motivate Design of Physical Products

Eric K. Clemons
The Wharton School
clemons@upenn.edu

Stina Teilmann-Lock
Copenhagen Business School
stte.mpp@cbs.dk

Abstract

Often research in information systems looks for reference disciplines, like information economics or game theory, that can inform and motivate our research. Here we reverse that paradigm and offer an area in which information system provides a reference discipline for the design of physical products.

Design for the Circular Economy is a green initiative that goes beyond recycling and focuses on the design of products that can remain in use almost indefinitely, and thus are not replaced and are not recycled. This leads to products for which maintenance, repair, upgrades, and style enhancements are less wasteful. This usually requires breakthroughs in design and in manufacturing processes.

There is a small set of design principles that enable Design for the Circular Economy, and that yield long-term benefits in the ownership and operation of products. Green design for the Circular Economy becomes relevant even for products with shorter lifetimes and lower costs.

1. Motivation

This paper was motivated by a request from a prestigious Danish architectural design firm to help them assess the economic value of commercial real estate design for the circular economy. Although there is a large literature on architecture, smart homes, and the green economy, neither they nor we could find any literature on commercial construction for the circular economy.

The circular economy represents a green initiative, and like all green programs it seeks to reduce waste. The best known green initiatives strive for energy efficiency, reducing reliance upon fossil fuels, and reducing waste through recycling. The circular economy seeks to reduce waste by maximizing the lifetime of all durable objects. Rather than salvaging the metals used in a laptop the circular design would ensure that chips, motherboards, buses, and disc drives could be repaired or replaced flexibility. Almost would need to be recycled, because nothing would be taken out of service and scrapped.

We started by creating a *metaphor,* an example of circular design that was selected from a discipline we understood well. We chose software engineering because we could design and create systems easily and we could display them easily. It thus provided a metaphor ideally suited to rapidly creating and discarding bad designs, and rapidly creating, updating, and displaying good designs. Software engineering has embraced modularity and reuse almost since its inception [16]; the Y2K crisis was a result of reuse, at the turn of the century, software designed in the 1960s and 70s. Using a domain that had long embraced reusability allowed us to explain a small set of principles that we knew would be essential to circular design in commercial real estate.

Finally, we will offer the architects a tool to calculate the financial value of the flexibility generated by circular design. The future value of commercial real estate is determined by the configuration of the building and the tenants' usage case at the time each tenant signs a lease. When the configuration of the building matches the use case of the most demanding future tenants, the building's owner can charge the highest rent. Flexibility provides *a real option* over the sequence of scenarios that emerges [4], allowing the owner to charge higher rents because the building's design is appropriate for the tenants who would value it the most.

We are now actively engaged in three additional stages for the architect. (i) We are working with them to design future environmental scenarios. Although architects have always designed for flexibility, they have not designed for unlimited flexibility, because of the higher cost. We are helping them design scenarios that they had not previously anticipated. (ii) We are helping them design use cases for these scenarios. For example, if the entrance to an office suite included rooms with the same aggressive ventilation as a commercial kitchen, it might be possible for consulting firms to have face-to-face meetings with clients, without concern for social distancing. (iii) We are actively engaged in gathering data with the architects, so they can assess the rent that could be charged for different building configurations, coupled with different tenant usage scenarios and different environments.

2. Flexible Design in the Circular Economy

There is a significant literature on what the circular

URI: https://hdl.handle.net/10125/71410
978-0-9981331-4-0

economy is and why green designers, green consumers, and green governments want to embrace it [3, 5, 6, 9, 12]. There is a shortage of specific guidance on how it can be implemented. This paper makes a first contribution by drawing lessons from software engineering, which is one of the first industries to master design complexity for long term flexibility, reuse, enhancements, maintenance and repair. We explore how these principles can be used in the design of complex physical systems. Additionally, our work includes developing software to calculate the option value of flexibility. by examining different rent streams enabled by different levels of flexibility under different environmental conditions.

Flexible design supports maximum ease of use, maximum ease of maintenance, maximum reuse of design elements, and longest possible useful system lifetime. This fits seamlessly into the concept of the *Circular Economy,* where to the greatest extent possible no object produced within the economy ever leaves the economy. This concept is far more comprehensive than recycling. We do not discard and scrap an object, separate it into its different raw materials, melt them down, and reuse them, as we would with recycling. To the greatest extent possible we reuse the entire object in its original form, repairing, updating, and upgrading only the smallest possible subunits.

As a concrete example, imagine a centuries-old manor house. The façade may have evolved from half-timbered Tudor to red brick Georgian. The gardens certainly have been enhanced over the centuries. Plumbing has been added, and then improved many times. Gas heating and lighting has replaced fireplaces and candles, which in turn was replaced with electrical lighting and central heating and central air-conditioning. The house remains. The house gets better and better, and its appearance changes to match the style of the times in which its owners live. Now go a step further and assume that the few critical components of the furnace, like the fan motor or the heat exchanger, can be replaced without replacing any other elements of the heating / ventilation / air conditioning system. Now go a step beyond that and imagine that the heat exchanger can be repaired forever and nothing ever needs to be discarded when it fails. We have the right to expect everything we own to be repairable [7]. All of these ideas together represent the ideal description of the circular economy.

We are using design of flexible extensible software as a metaphor to explain and motivate flexible design of durable physical structures. In this case we are focusing on the design of commercial real estate. Today the most complex systems that humans create are the software systems that control physical objects like stealth aircraft and physical systems like transportation and power grids. The principles of extensible and flexible design first emerged in software. As green consciousness and the conservation of physical resources becomes more critical, we seek to apply these same design principles to physical systems.

Software engineering has long embraced Modular Design [11, 14]. Each module performs a well-defined function. Modules have well-defined interfaces, and interact with other modules in fully predictable ways.

Modular design permits the overall structure of the system to be maintained without change even when individual modules need to be updated or replaced entirely. To achieve this, modules need to interact flawlessly for the system to function, but interactions are highly localized and well-defined, so that individual modules can be replaced or repaired without replacing or repairing the entire system. Updating the plumbing system should not require updating the electrical system or the gardens in our manor house.

There is a small set of key design principles, all of which are essential to design for the circular economy. We describe each below.

There are clear competitive advantages that firms will master when they master design for the circular economy. This is the motivation for our work. We will discuss competitive advantage in more detail below.

3. Literature Review on the Circular Economy

We start with a definition of the Circular Economy. *"A circular economy is based on the principles of designing out waste and pollution, keeping products and materials in use, and regenerating natural systems."* [9]

This can go beyond traditional recycling, melting down and reusing the plastic in bottles and the aluminum in cans. AMG goes beyond recycling catalysts for oil refineries, and extracts vanadium from the spent filters [2]. Vanadium has previously been a toxic waste produced by oil refineries; vanadium is essential to storage batteries needed to deal with the time-varying behavior of solar and wind-powered electricity generation, and AMG is now the largest producer of vanadium in the United States.

We are not the first to consider the use of design theory to implement the concepts of the circular economy; see Andrews [3]. But we have been unable to locate any prior work that explicitly focuses on the use of modular design and the advantages that are generated by modular design.

We focus here on design of complex products, to extend their lifetimes and reduce or ultimately eliminate waste. Design for the circular economy has become a national priority for Denmark [1, 5, 8] which is one of the reasons that we became interested in the research area.

Some authors focus on a broad set of design objectives, ranging from design for durability to design for ease of recycling [6]. Others focus in depth on the mathematical implications for modular design throughout the supply chain [10]. There numerous conferences held annually, many sponsored by the Ellen MacArthur Foundation, and the Foundation has published numerous studies. The work that comes the closest to our own (https://www.ellenmacarthurfoundation.org/explore/circular-design) is discussed in one of the foundation's web page on design for the circular economy, where modularity was the 11th of 13 design principles and was described in 110 words. We believe that the lessons from modular design of software systems allows us to go beyond what is already known about circular design of existing physical products.

Other authors have observed that modularity offers competitive advantage in dynamic and changing environments [15], or for design of specific physical products in such environments [13]. However, while these authors have demonstrated the value, they have not provided any guidance on how to achieve modular design.

We note that while the concept of modular design for the circular economy appears to us to be very promising, it is far easier to find companies offering consulting services than it is to find refereed academic publications.

4. Key Concepts

We see five key concepts that underlie extensible, reusable, and maintainable design in software engineering, and we believe that the same five concepts underlie extensible, reusable, maintainable design in all problem areas within the circular economy. For brevity we will refer to this as *sustainable design.*

Modularity requires dividing design into separate components, with clear boundaries between them, limited interactions with other modules, and well-defined interactions with other modules. The well-defined and limited interactions can be considered as having simple standard interfaces between all modules.

Recursive Modularity requires that each module be structured with modular design, built on simpler modules, again as separate components with clear boundaries, limited interaction, and standard interfaces.

With separate modules, each with well-defined interfaces and with interaction solely through those interfaces, it is easy to remove components and replace them with alternative components with different capabilities, often from different suppliers. The use of standard interfaces leads to easy replacement from any supplier, one of the defining characteristic of *Open Design*.

Transparency allows the purpose and the structure of the object and of its modules to be readily be apparent to professionals with experience in the design of similar objects.

Intergenerational Compatibility ensures that changes are possible when the need for them should have been foreseeable. The architects who designed 18th century manor houses can be forgiven for failing to anticipate gas fired central heating, and for failing to make provisions for aluminum duct work and electrically powered forced-air heating.

Our colleague Jason Woodard suggested two related but separate concepts. The first is *informed flexibility.* It is not possible to prepare for everything, and an informed designer will have a better sense of how much flexibility to provide. It probably does make sense to design commercial real estate with the option of sufficient ventilation to enable safe mask-less meetings and safe mask-less shopping. It probably does not make sense to design center city real estate to serve as aircraft hangars or shipyards in the future. The second, related concept, is *intuitive modularity*. What, exactly, should a module be, and how much flexibility should a module be designed to support?

Our Philadelphia apartment was designed by the same firm that designed the Philadelphia Museum of Art. Windows are the same size today that they were when the building opened and the passenger elevator shafts likewise are the same size and in the same locations. Many of the bricks in the exterior façade are original. The interior weight-bearing structures are unchanged. And virtually nothing else from the original building remains. With standard sizes for bricks, windows, and elevator shafts, the building has evolved over the nearly one century it has been occupied.

However, it is not possible to increase the size or change the location of the windows in the building's façade because they are located between weight-bearing structures in the building. The size and the locations of the window gaps cannot be changed. In contrast, new construction a few blocks from our apartment has entire walls bolted into place, attached from the outside to the building's weight bearing structures. It is now possible to replace bricks, or to replace windows, or to replace entire segments of the building's façade with new structures that include larger windows.

5. Benefits that Result from Key Concepts

We see four benefits that arise immediately from these four key concepts. The first three offer enormous competitive advantage to firms that master them. The fourth offers advantages to societies that embrace them.

The first source of competitive advantage is *Constant and Continuous Improvement,* which enabled by the easy replacement of one or modules without affecting anything else in the complex system and without requiring a cascade of additional changes. At any point in the life cycle of a product the manufacturer can easily introduce a superior model simply by replacing a single module with a superior one, without needing to alter anything else in the product's design. When new storage batteries are developed with a longer life or longer driving range on a single charge, the manufacturer can make that change. Additionally, owners of existing products can make the same change; as soon as one manufacturer abandons traditional planned obsolesces, all we need to follow. But the first company that embraces modularity to achieve continuous improvement will gain competitive advantage over all other players in its industry.

Additionally, the principles of circular design *Facilitate Breakthrough Innovation.* With carefully defined interfaces it is possible to make dramatic improvements to one element of technology to reflect dramatic improvement without needing to discard and replace anything else. Adding self-driving capability requires upgrading software so that the navigation system can control the vehicle and upgrading software so that the navigation system is linked to image processing systems that monitor the positions and velocities of other vehicles close enough to be interesting. Those are relatively simple software upgrades. New actuators will need to be added, so that the vehicle can autonomously steer, accelerate, and brake. But the rest of the vehicle can remain unchanged. Again, the first company that embraces modularity will always offer products that remain profoundly superior to those of its competitors.

Our design principles *Enable Incremental Change* to accommodate evolving style. If the exterior, user-facing shell is also a separate module, then it can be replaced. Again, this is the opposite of planned obsolescence, and again the first company that embraces modularity will always offer products that remain profoundly superior to those of its competitors.

Perhaps most significantly from an environmental perspective, our principles *Enable Maintenance with Minimum Waste* — With fully modular design repairs can be made to the smallest sub-module that contains the defect, allowing virtually all of the rest of the product to remain unchanged. Again, when the vehicle's electric motor needs to be repaired, owners will not need to replace the drive-train, or the motor, but will replace the smallest components within the motor that need to be replaced. This reduces or eliminates the problems caused by products that are designed to fail, and will appeal to consumers both because of reduced expense for maintenance and reduced environmental impact from being forced to scrap vehicles or large modules within a vehicle.

6. A Visual Example from Software Engineering

We construct a single software system modeling complex service operations to make each of the design principles more clear. We will use simulation, because high level graphical simulation languages allow simple visual presentation, which allows us to provide visual examples of all of the principles.

We will start by constructing a simulation of a barbershop. My simulation language of choice is Goldsim because of its modeling power, and because Goldsim is provided without charge to academic institutions for classroom use.

The design is *modular*, with independent units performing separate functions.

The design is *recursively modular*, with modules separable into smaller independent units.

The design is *open*, and modules can be modified, augmented, or replaced without cascades of unanticipated effects in other segments of the model.

The design is *transparent*. In the final version, shown in section 5.6, critical data is both visible to users and controllable by users. Users can see and can control all data that critically influences the results obtained.

The design exhibits *intergenerational compatibility*. Resetting parameters to the values used in model 6.2 will allow model 6.6 to produce the same results as model 6.2.

Each module makes intuitive sense; each component, at its most basic level, models the behavior of well-defined entities, like customers, service personal, or resources needed to perform service. We have *intuitive modularity*. Additionally, we have *informed flexibility* because fully understand the range of behaviors that these modules will need to support. Customers arrive, leave because the queue is too long or decide to wait for service, continue to wait until they are served, or get impatient and leave. Servers start a service or are idle while waiting for a customer to arrive. Additional servers can join at times of peak demand or leave when they are no longer needed. We can create additional customer types and additional types of service.

Adherence to these design principles allow us to make continuous improvements, like giving the user more control over parameter settings. They allow us to make dramatic changes, like allowing us to add new types of customers, who enjoy higher priority, and who can demand additional types of services. They allow us to make changes to the user interface.

We start with a simple disposable prototype, which is inflexible, cannot be extended or maintained, and lacks transparency. We then introduce a model that exhibits our design principles, and show how adherence to these principles allows us to extend the model, dramatically improve its capability, and enhance its appearance.

6.1. Visual Example 1 — The Simplest Simulation in Goldsim

Below is the simplest Goldsim code we feel comfortable presenting. It handles the arrival of a random sequence of customers, arriving randomly, and served by a single barber. The shop closes at 10 pm. Any customer who happens to be waiting is not served. Any haircut that has begun but didn't finish before 10 pm doesn't finish and is not counted. It's not elegant, but we are assuming that over the course of a long day the inaccuracy introduced by these simplifying assumptions can be ignored. It's the simplest way we could build the simulation. See figure 1A.

Arrivals occur randomly, controlled by the Clock-Element *ArrivalArr*, and the arrival rate of 2 per hour is hidden inside the ArrivalArr element; it can only be changed by a Goldsim programmer. *ArrivalArr* automatically places the customer in the *Queue* if there is room; the *Queue* has a maximum size of 5 customers waiting, and this is also hidden inside the *Queue* element and can only be changed by a Goldsim programmer. Whenever a customer arrives or a haircut completes, the Gold ?-element *DecideIfStart* decides if a haircut can now start. When the necessary conditions are met, that is, when a customer is waiting, it attempts to trigger the Gold Hourglass-Element *PerformHaircut*. Haircuts take exactly 1 hour to complete. Since we have 1 barber, there can be at most one haircut performed simultaneously, which is once again controlled by a setting hidden within *PerformHaircut*.

The model works. But it cannot be extended, and it is not designed to be used except by the programmer who wrote it. We allow Goldsim to control the queue and the order in which customers are served; We cannot create different customer types with different priorities. We allow Goldsim to determine when the barber is free; We cannot readily change the number of barbers over time or vary the assignment of barbers to customers. Customers cannot easily observe the values We assumed for each parameter, so they cannot easily change any of them. The model can either be used in its current form or discarded. This is not circular design.

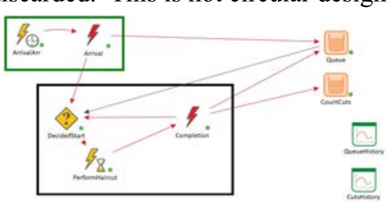
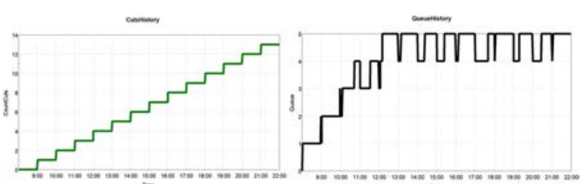


Figure 1A—A simple disposable model simulating a barbershop with one barber.



Figures 1AB and 1C—Outputs from the simple disposable model simulating a barbershop with one barber.

6.2. Visual Example 2 — Adding Explicit Control over the Number of Barbers and More Complex Customer Behavior

This is our first software model that can arguably be described as following sustainable design principles. We made several changes to the code, to give ourselves more control and more flexibility, and to make things more transparent to a user who is not a Goldsim programmer. The code is modular and extensible. There is an arrival process and a haircut process, and they are clearly separate; the only interaction is that the arrivals place customers in the queue and the haircut process removes them. The only way modules communicate and the only data they change is assumed to be the Orange Bucket-container boxes. Because of the clear separation either process can be extended without making any changes to the other. See figure 2A below.

Arrival rates are now specified as varying over time, and are determined by the Blue Selector-box *ArrivalLambda*.

We now manage the queue of waiting customers explicitly. When a customer arrives the arrival is counted. If the customer decides to stay because the *Queue* is below the maximum queue length specified for customers of the shop, then the customer stays and is added to the queue of waiting customers. This is all determined by the processing within the green border box we drew around it.

The maximum queue length MaxQueue is a Green Pencil-input box, and it is not hidden inside a processing box.

The number of barbers can be altered, and is specified through another Green Pencil-input box.

The Gold ?-Box *DecideIfStart* now checks to ensure that a barber is available and that a customer is waiting before starting a haircut; we can easily expand the functionality of the box to check if different categories of customer are waiting, and we can impose a priority ordering to determine which type of customer gets served first. This will be useful later.

Because we are managing the pool of barbers explicitly, the haircut process appears more complicated than it did before. When *DecideIfStart* activates *PerformHaircut*, it also engages a barber and reduces the pool of free barbers. When *PerformHaircut* completes, the completion returns a barber to the pool of free barbers. This is all contained in the black border box drew around this bit of processing. Again, this additional complexity will give us the ability to modify or to extend functionality later.

The length of a haircut is now a random variable, specified in the Green Distribution-box *LengthofHaircut*

The Orange Buckets each has a purpose. *Queue* tells us how many customers are waiting, *BarbersFree* tells me how many barbers are available, *AllArivals, Exits, and Completions* allow me to track the performance of the shop.

We've added three output graphs, collected together in the green dashed box.

And we've added a *dashboard*, shown in in the box on the upper right, which gives our users easy ability to alter data and easy access to all the graphs.

The most important changes we made to processing were to make the model extensible in a variety of ways, which is central to sustainable design for the circular economy.

The use of a dashboard also adheres to sustainable design principles. It provides the user explicit control over critical inputs and it provides convenient access to all of the outputs. It allows the user to make many modifications directly, without a programmer, and it allows the user to see the results of those modifications instantly. And it allows the user to change the façade of the model without changing any internal processing.

This design allows us to implement much more reasonable assumptions than we used in our primitive disposable first model. By turning off the arrival of new customers late in the day, that is, by setting the arrival rate to zero after 6 pm, we can ensure that no customers are still in the shop waiting for the completion of their haircuts when the shop closes.

It's clear that in some sense the structure of figure 2A is the same as the structure of figure 1A. Arrivals are still processed within the green border box above and scheduling and performing haircuts is still processed in the black border box below. But it's clear that the structure is both more complicated and more transparent. There are inputs on the left that determine the number of barbers, the maximum queue length, arrival rate parameters, and the length of a haircut. Since these are no longer hidden inside processing details they are subject to control by users. As importantly, they can be found and altered by subsequent developers responsible for enhancing the model. Critical elements, the queue of customers and the pool of free barbers, are visible, as are the counters that total up events as they occur.

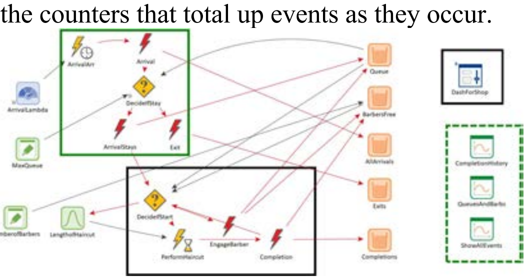


Figure 2A—A simple model simulating a barbershop with one or more barbers, exhibiting sustainable design principles.

The dashboard allows the user to set the number of barbers and the maximum queue length of waiting customers. It also provides ready access to the program's outputs. We can see that customer arrivals starts slow, gains speed throughout the day, peaks after lunchtime, and stops when the shop closes at 6.00. Looking at the graph on the lower left we can see that the mean number of barbers who are free approaches zero during the period of peak arrivals and the queue increases in length. Comparing the graph on the upper left with the graph on the lower right we can see that the shop is losing customers who leave the shop because they are discouraged by the long queue and the number of haircuts performed is less than the number of customers who arrive as a result. See figure 2B below.

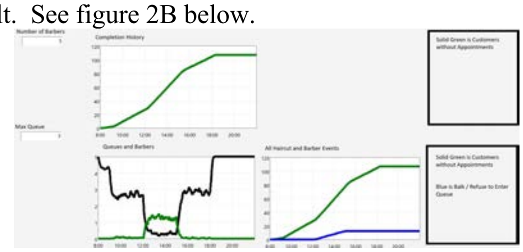


Figure 2B—Display from our simple model that follows sustainable design principles.

6.3. Visual Example 3 —Allowing for More Complex Customer Behavior, And Adding Part-time Barbers

Moving from the model in figure 1A to the model in figure 2A required that we discard the model and start over. With modular design and sustainable design principles exploited in the model of section 5.2, we can extend the model from figure 2A almost effortlessly. Here we have added two new functions to the model.

We have added the capability to hire part-time barbers. All of the logic for adding staff is handled within the new dashed purple box. The model adds barbers when it is time to add them, triggered by the Gold Bolt-? box *AddStaff*. Likewise, the model removes barbers when it is time to remove them, triggered by the Gold Bolt-? box *RemoveStaff*. Nothing in our code to process the arrival of customers or the process of performing haircuts needs be altered to in any way to accommodate the process that changes the number of barbers.

We have added the data elements needed to control the hiring of part time barbers. To accommodate additional barbers we also needed to create three new Green Pencil-box inputs, for the time at which we add staff (*AddStaffTime*), the number of staff we add (*AddStaffCount*), and the length of time the additional staff work (*AddStaffLength*). None of the existing inputs need to be changed in any way.

We have increased the range of customer behaviors that we can model. We have added the ability for customers to get frustrated by the length of their wait. Every couple of minutes customers get fidgety, triggered by the Gold Bolt-clock box. They check the length of the *Queue* against the number of barbers, and decide whether to remain in the shop or to renege. Customers who renege leave, and are counted as *Reneges* and removed from the *Queue*. Once again, nothing outside this box needs to be altered to accommodate adding the possibility of reneging.

It was not necessary that the initial programmer was able to anticipate these changes. Our design made it easy to accommodate changes to customer behavior and the operation of the shop.

No one today expects that an architect in the 1700s should have anticipated central forced air heating, with aluminum ductwork, gas-fired furnaces, and electric fans. But programmers should anticipate changes in customer behavior and in barber employment. See figure 3A below.

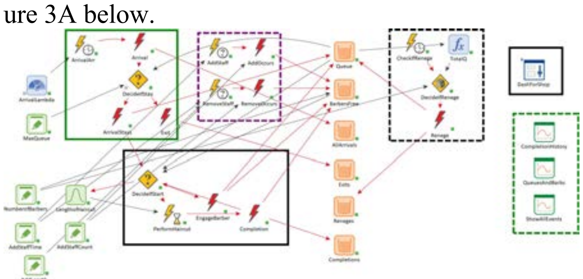


Figure 3A—Our simple but well-designed model, which allows us to add part time barbers and more complex customer behavior

As is clear from the dashboard below, these changes have produced two visible impacts on the operations of our shop. First, there is a new source of lost customers. In addition to losing customers who balk and choose not to enter the queue we are also losing customers who renege and choose not to stay. Second, the addition of part-time barbers results in the shop completing more haircuts and serving more customers. The total loss of customers due to balking and reneging combined is less than the loss we previously experienced to balking alone.

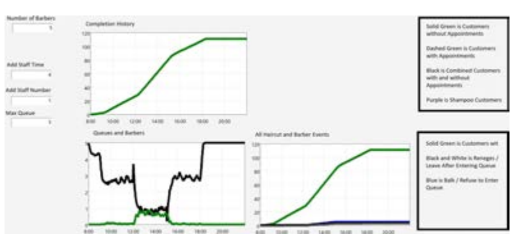


Figure 3B—Display from the model in figure 3A.

6.4. Visual Example 4 — Adding a Second Customer Type with More Complex Service Possibilities

Once again, we need to make changes to our model, to reflect changes in the operation of our barbershop. And once again, we see the advantage of adhering to sustainable design principles. We need to add a second type of customer, customers who make appointments before arriving. These customers are different from our original customers in several ways, including their higher priority, their longer service times, and their willingness to request additional services. Because of our modular design, we can make these additional changes easily.

First, we need to add a new arrival process, the teal border box that contains the Gold Bolt-clock box that is triggered when a customer arrives with an appointment. These customers never balk, so the arrival process for these customers is easy.

We need to add a new input, the number of customers per hour who arrive with appointments, the Green Pencil-input box *ApptLambda*.

Since these are not new customers, merely a changed behavior in existing customers, we need to subtract *ApptLambda* from *ArrivalLambda* to create *AdjustArrLambda*.

We need create an additional service type for these new customers, represented by the Gold Hourglass-box *PerformAppointment*, which has a slightly longer service time than our original customer type.

We modify the Gold ?-box *DecideIfStart*, to start an appointment haircut if a customer is waiting and a barber is free. If so, by analogy with *PerformHaircut*, the model removes an appointment customer from his queue and assign a barber. Again, by analogy, the model returns the barber to the barber pool when the appointment haircut is completed. And they are higher priority, and we always serve this customers before serving customers without an appointment; *DecideIfStart* will only start a regular haircut if no customer with an appointment is waiting. These are the only changes we make to existing code.

Appointment customers often request an additional service, a shampoo, which doesn't take very long. The shampoo is quite profitable, as we will see later when we add cost and revenue information to the model. The shampoo is performed by a barber, so it does not require an additional type of server, but it requires a dedicated shampoo station, so there is one more resource that must be available and must be controlled, just the way we controlled the pool of barbers. If a customer wants a shampoo and a shampoo station is available the model removes one station from the pool of available stations and starts *PerformShampoo*; when the shampoo is complete the model returns one station to the pool of available stations and free sup the barber. Customers will not wait for a shampoo. If a shampoo station is not available, they will leave without waiting for a shampoo.

And because we will want to observe the total number of haircuts, which is the sum of the haircuts given to the two types of customers, we need to add a single Blue Fx-calculation box to calculate the *CombinedTotal.*

Significantly, because of the modular design we have adopted, it is easy to continue to add functionality without cascading changes throughout the existing model. Note that once again it is possible that at the time of the construction of initial model, as shown in figure 2A, the programmer did not anticipate any of these future changes. Our initial modular design makes it possible to make extensions without modification to existing processes. This is, indeed, sustainable design. See figure 4A below.

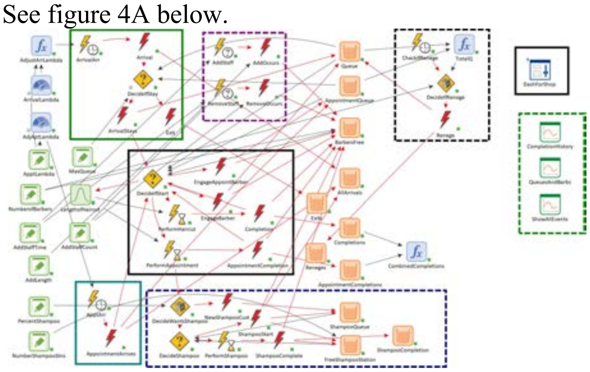


Figure 4A—Our model allows us to add another customer type and more complex customer services.

Again, note that moving from 3A to 4A requires two new boxes, a teal boundary box for arrivals of the new customers and a dashed purple boundary box to process the new customers. There is one place and one place only where we needed to make changes to existing code, *DecideIfStart*, which now needs to determine which type of event starts. As we would hope, design that embraces our principles accommodates numerous enhancements to functionality with minimal changes to existing design elements.

The dashboard in figure 4B shows very little change in the number of total number of haircuts performed. It does show a significant increase in the number of shampoos performed. The utilization of barbers has increased, which is good. But shampoo stations add another fixed cost. We cannot determine if the new configuration is profitable until we add costs and revenues and calculate profits. That will be the next set of changes we make to the model.

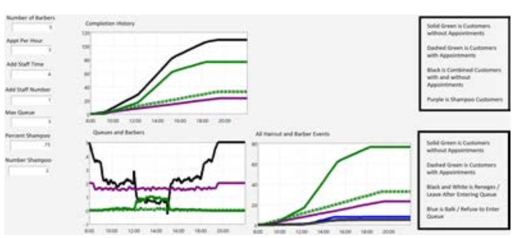


Figure 4B—Display from model in figure 4A.

6.5.Visual Example 5 — Adding Profits

We now decide to extend the model to calculate profits. This will allow the user to assess which combination of full-time and part-time barbers, appointment and regular customers, and shampoo stations, results in the highest profits. Because of our modular design, we merely add a new set of functionality, the calculation of profits, which is performed in the solid blue boundary box.

Because of the modularity of our design, no other changes are required. See figure 5A below.

Again, we were able to make the necessary additions, without any changes to any of the existing elements of our code. Think of this as being able to add navigation control and self-driving capability to an existing car with additions but no changes to any existing element of the vehicle.

When we compare profitability with and without appointment customers, we see that the addition of appointment customers with shampoo does increase profits. Compare figures 5B and 5C.

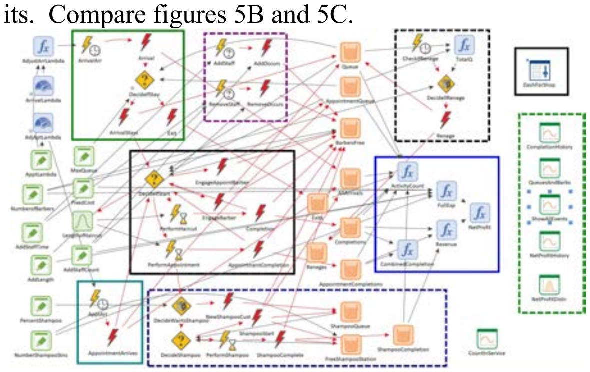


Figure 5A—Adding the calculation of profits.

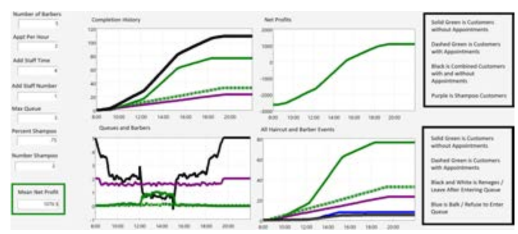


Figure 5B—Displaying profits when the model has two customer types and the possibility of shampoo.

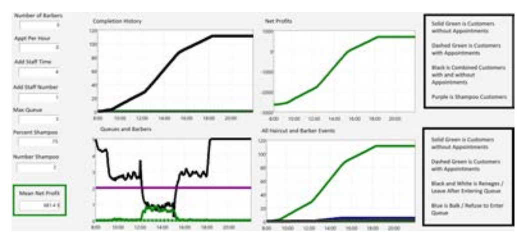


Figure 5C—Displaying profits when the model has one customer type without the possibility of shampoo.

6.6.Summary of Design Principles in Software Engineering and Relevance to Physical Product Design

We have shown how principles such as recursive modularity and transparency allow us to take the model of section 5.2 and add more complex behavior. We modeled more complex behavior, allowing customers to make appointments, get frustrated and leave the shop after waiting, and choose shampoos. We allowed "repairs" to be made to arrival times. We allowed additional functionality, adding profits. Finally, we showed how code for the arrival of customers waiting for a haircut could easily be copied and reused, to model customers arriving for a manicure.

7. Application to Physical Design

Recently, an office building was constructed in Philadelphia using modular design. First, the weight-bearing skeleton of the building went up. Then cranes lifted modular slabs of siding into place. Each slab was one story high by one unit-room wide; that is, it contained two large windows and appeared to be constructed from individual bricks, but it did not require a window installer or a brick-layer. The building's sheath was applied slab by slab.

The design is recursively modular. It is possible to replace a slab's window assembly with one of the same standard dimensions, and that it is possible to replace individual panes of glass in each window assembly as well. The building has a small number of weight-bearing internal structures, which means that each floor can be reconfigured as needed, into individual offices for a single company or separate suites for separate companies branching off a central hallway. Bathrooms are already recursively modular, since sinks have standard holes for tap handles and faucets. Moreover, the faucets are likewise modular, and defective washers can be replaced without replacing the entire faucet, the entire faucet and spout assembly, or the entire sink.

Likewise, elevators have been recursively modular almost since their inception. Elevator cars are standard sizes; it is possible to replace an elevator cab without recutting an interior elevator shaft. Control systems can be replaced without changing the elevator cabs. Control systems can be re-programmed without replacing any other elements. My new elevator system at home has one elevator always stationed at the lobby, because half of all traffic originates in the lobby. The other two elevators are prepositioned at 1/3 and 2/3 of the height of the building.

We believe there are four factors that determine when circular design can be employed. First, the item has to be *expensive* enough to be worth maintaining; buildings are expensive! Second, the item has to have a *long useful lifetime*, long enough to require changes in functionality; building space needs to be configured for new tenants, since a restaurant requires very different ventilation and lighting than a gymnasium or an executive office floor. Third, the item needs to be intended to be built to last long enough to *require enhancements in technology during its useful lifetime*; the airconditioned and self-prepositioning elevators in my home run in the same shafts that were in use when the building was first constructed, but not one piece of the original elevator technology remains in place. And finally, the *underlying technologies need to progress slowly enough to have predictable changes*.

We will see the same attention to maintenance, change in functionality, and technical upgrades in home appliances. A high definition Panasonic flat-screen TV from a dozen years ago still offers a sharp, crisp high definition image, and it has all the outputs needed to connect it to a great stereo; however, its processor cannot accommodate the most recent operating system upgrades needed to run Netflix, and the processor cannot be replaced. At some point the waste associated with this kind of design flaw will be unacceptable both to consumers and to regulators.

Can we explain why we replace our cellphones so often, rather than upgrading them [13]? Cellphones do not satisfy our fourth requirement for design for the circular economy, predictable rates of technological progress. We can anticipate that screens will become larger, and higher resolution, and chips will become faster, and batteries will become better, but often each new release of an iPhone or a Samsung represents a quantum leap in all areas at once. That will cease. Phones cannot get larger or they will not fit in a pocket or a human hand. Screens will not become higher resolution because they already exceed the visual acuity of the human eye. When progress slows and becomes predictable upgrading a processor, replacing a battery, or increasing memory will become as easy as inserting a SIM card.

8. Conclusions

8.1. Contributions

We never had unlimited ability to anticipate future requirements when we designed software, and it is unreasonable to assume that we will ever have unlimited ability to anticipate future requirements when we build homes, office tours, or appliances. But for anything that is expensive, long lasting, and going to need maintenance and upgrades, our ability to design future products is going to have to be influenced by the lessons we have learned in software engineering.

We have extracted a set of five principles that are common in large software projects and shown how they can be applied in the design of physical products for the Circular Economy. We believe that this goes beyond what has already been learned about this form of design.

8.2. Limitations and Future Research

Our observations are based on comparing the attributes of physical products that do and do not already exhibit design for the circular economy. We compare the attributes of those products to the attributes of software projects large enough and complex enough to have followed the design principles that lead to software with a long useful lifetime. But we have not yet provided quantitative estimates of the value of flexibility.

We have begun cooperation with a Danish architectural firm to explore use of these principles in the design of commercial real estate. The firm is especially interested in how flexible modular design can be valued. They will obtain data on how different businesses use different configurations of floor space, which are most in demand, and which can attract the tenants willing to pay the highest rent. We are working with them to create a set of scenarios describing future business environments. For each environment, different building configurations will have different uses, and different values for those uses. And the modular design we are exploring together allows buildings to be reconfigured on demand, allowing the building owner to adjust the utilization of interior floor space to meet the demands of tenants who are willing to pay the highest rent.

We have completed implementation of a simple tool to calculate the value of flexibility. It combines scenario analysis to manage long term uncertainty with valuation of specific actions under each scenario [4]. It compares the 30-year rental streams generated by a fixed traditional design and a flexible design. Each design has a different cost to construct. We assume that if the architect builds a ***traditional*** design it is optimized for the current set of operating environments. We assume that if the architect builds a ***flexible*** design it can be reconfigured, at a known cost, for whatever usage generates the highest revenue under the conditions in effect when leases are signed, over the lifetime of the building.

We then run the model for 30 years, under a random sequence of usage scenarios. At each lease renewal the owner of the ***traditional building*** earns rents corresponding to the maximum any tenant would pay for any usage scenario supported by the building's initial configuration. In contrast, at each lease renewal the owner of the ***flexible building*** earns rents corresponding to the maximum any tenant would pay for any usage scenario supported by any possible usage configuration supported by the building in its present form or after reconfiguration.

We compare the net profits at the end of 30 years, where profits are determined by initial construction costs, reconfiguration costs if any, and optimal revenue streams enabled by the design in the environments that evolve. The difference is the option value of flexibility. Goldsim provides means and a range of statistical information. This option value should result in higher sales price when the architectural firm completes and sells the building.

References:

[1] 3daysofdesign, "Eco Conscious Concepts", https://3daysofdesign.dk/

[2] AMG Press Release, AMG Advanced Metallurgical Group N.V. and Shell Catalysts & Technologies Agree to form Shell & AMG Recycling B.V.", Amsterdam, 2019, https://amg-nv.com/feed-posts/amg-advanced-metallurgical-group-n-v-and-shell-catalysts-technologies-agree-to-form-shell-amg-recycling-b-v/.

[3] Deborah Andrews, "The circular economy, design thinking and education for sustainability", Local Economy. Vol. 30, No. 3, 2015, pp 305-315.

[4] Eric K. Clemons and Bin Gu, "Justifying Contingent Information Technology Investments: Balancing the Need for Speed of Action with Certainty before Action", Journal of Management Information Systems, Vol. 20, No. 2 (Fall, 2003), pp. 11-48.

[5] Danish Design Council, "Circular economy", http://danishdesigncouncil.dk/en/circular-economy/

[6] Brett Fifield and Katerina Medkova, "Circular Design — Design for Circular Economy ", Conference Proceedings: Smart Cities in Smart Regions, Lahti 2016, https://www.researchgate.net/publication/313771263_Circular_Design_-_Design_for_Circular_Economy.

[7] Ifixit, "We Have the Right to Repair Everything We Own", https://www.ifixit.com/Right-to-Repair/Intro.

[8] Ministry of Environment and Food Of Denmark, "Towards Circular Business Models: Experiences in Eight Danish Companies", April 2018, pp. 82, https://www2.mst.dk/Udgiv/publications/2018/04/978-87-93614-97-0.pdf.

[9] Ellen MacArthur Foundation, "What is the circular economy?", https://www.ellenmacarthurfoundation.org/circular-economy/what-is-the-circular-economy.

[10] Thomas Nowak, Fuminori Toyasaki, and Tina Wakolbinger, "The Road Toward a Circular Economy: The Role of Modular Product Designs in Supply Chains", Innovative Solutions for Sustainable Supply Chains (Understanding Complex Systems), Springer 2018, pp 111-133.

[11] John Ousterhout, A Philosophy of Software Design, Yaknyam Press (2018) pp. 190.

[12] Tim Price, "'Generation Green' : how millennials will shape the circular economy", Environmental Journal, 2018, https://environmentjournal.online/articles/generation-green-how-millennials-will-shape-the-circular-economy/.

[13] Karsten Schischke, Marina Proske, Nils Nissen, and Klaus-Dieter Lang, "Modular products: Smartphone design from a circular economy perspective", Conference Proceedings Electronics Goes Green, 2016, pp 1-8, https://www.researchgate.net/publication/312964839_Modular_products_Smartphone_design_from_a_circular_economy_perspective.

[14] Kevin Sullivan, William Griswold, Yuanfang Cai, Ben Hallen,. (2001). The Structure and Value of Modularity in Software Design. ACM SIGSOFT Software Engineering Notes, 2001, 26.

[15] C. Jason Woodard and Eric K. Clemons, "The Evolution of Modular Product Architectures and the Emergence of Platform Ecosystems", 2013, https://pdfs.semanticscholar.org/fcc3/ec2dedadbb016f818049831e4eed0afa707e.pdf

[16] S. Younoussi and Ounsa Roudies, "All about software reusability: A systematic literature review". Journal of Theoretical and Applied Information Technology. (2015) Vol. 76 No. 1, pp. 64-75.