Andy Hoffner
Assignment 4

Intelligent, computerized critiquing systems is definitely an intriguing idea. It's more than likely that as our understanding of the design process increases so will the astuteness of our software and operating systems. While Kufmann discuss many aspects of the computerized critic, I found his most interesting points concerned using a critic to increase the designer's knowledgebase. I think Kaufmann was right in assuming most designers are unwilling to disrupt the design process to seek out information, even when they know more information exists, and furthermore that designers may not even know when they need more information. I can recall a recent experience of my own – when we wrote a Java program to parse VXML – in which if we had known more about Java's built-in classes for handling we might have them, instead of manually writing a parser using regular expressions. If our programming environment was smarter, we could avoid writing more code than we had to. There is plenty of information on using Java's classes to parse VXML online, but we simply didn't know to look for them. It would be unbelievably helpful if the coding environments were more helpful in this respect, and maybe one day they will be. I think right now the best we can do is to identify *how* a system like this could be built, and make incremental steps to developing intelligent critiquing systems.

Kaufmann lays out a clean generic solution, but far more design work would be needed to adapt this system to specific areas of computing, whether it be coding, digital art, business software or whatever – really all areas of computing could benefit from an intelligent critiquing system. The problem really lies in definition and expandability. Some concepts are as clear-cut as designing a kitchen. If a critic was developed for Photoshop, how can you determine what makes a good image? There might be some generic artistic concepts that could be applied (rule of thirds, complementary colors), but art is in many ways a subjective science, not an objective one. Critics in coding developments would have difficulties as well. You may be able to specify some generic goals ahead of time (like in Kaufmann's example of the left-handed cook), but telling a computer what your trying to do *before* writing code can be a logistic impossibility. Really the only way to tell a computer what you're going to code is by showing it the computer the code structure itself.

The question that remains then, I think, is what we can and cannot at this point define as rules for critics. If computational systems arise that allow for some kind of derived intelligence, an *artificial* intelligence, then we can see the full benefits of this type of electronic critiquing design structure. Until then, these systems are forced by practicality into niche areas of design in which structured rules can be inherently derived.

That's not to say there isn't a good deal of these 'niche areas' available. And I think that almost all types of design software have at least *some* so-called generic rules. HYDRA was an interesting example of the critic-design software, but I would be interested to know of other examples of computerized critics that are already in use, and how users feel about their assistance in relation to the design process.

Kaufmann's ideas really epitomize the basic necessary structure a computerized critiquing system would need to have to truly be useful. Much of the work we do on computers is design work, and the remaining part deals with knowledge acquisition. Computerized critics facilitate a means to combine the two, extending available knowledge to you on demand (even when you didn't think you needed to demand it). They are in essence collaborating with you, allowing your to refine your design and guiding you through the process of 'reframing the problem', as Kaufmann puts it. If a critic can be extendable, and later sharable, then it too can benefit from your knowledge and bring the knowledge of other real experts too you, effectively expanding your knowledgebase in the hopes to better your designs.

I would like to see software development environments become more intelligent. I think we are just starting to see some emergence of computational critics, like the Eclipse IDE, which can provide suggestions on how to correct errors and warnings in your code. But there is a lot more it could do for you.