Choose one (1) of the following prompts and provide your opinion, experience and/or ideas.

* In the article, the author states " I can imagine a not-too-distant future in which the line between design and development completely fades away, and we are collaborating using the same language, abstractions, and tools." What do you think he means and what would it take to get there?
* The article is written by a software designer.  The author states "We also knew that, as a product, it was crucial to delight our customers, who were mainly application developers."  Did that statement surprise you? Isn't the customer supposed to be the end user?  Why is the application developer a customer in this article? Should it be that way?
* The author states "The problem is that these cutting-edge methods and tools target only one side of the craft. A gap continues to plague Web design and development. And in the middle is a one-way hand-off that lacks consideration for both sides."  The author lists several things that are needed to close the gap.  Did you agree or disagree with any of them? If so, which one(s)?
* The author quotes Facebook in saying "“Components let you split the UI into independent, reusable pieces, and think about each piece in isolation". (source: Facebook). Is it important to do that? Why or why not?

The author states "The problem is that these cutting-edge methods and tools target only one side of the craft. A gap continues to plague Web design and development. And in the middle is a one-way hand-off that lacks consideration for both sides."  The author lists several things that are needed to close the gap.  Did you agree or disagree with any of them? If so, which one(s)?

* Express design as working components
  + Convergence of language
    - Upside: This would assist in bridging the gap between the design and the development. More clarity in the intents and requests of the designers would lead to easier development processes and a lower chance of the designer adding a component that designer did not have in mind. It would effectively decrease development time.
    - Downside: This would increase the barrier for entry of the development field on the front of design teams. If they wish for a new component, they must identify the more technical name of the component and more so learn to remember these names.
    - Overall: Close, but advantage to upside.
  + Distinguishing of component types
    - Upside: Members of teams can know how big and useful specific parts of the applications are and allocate priority to them appropriately. They can also make more abstract classes that can be adapted for individual classes.
    - Downside: Priority may be allocated incorrectly. Importance may be misallocated to specific components.
    - Overall: Heavy advantage to upside.
  + Overall: Upside advantage
* Component-centric workflow
  + Focused design
    - Upside: Focus may be given to more important parts of project. May assist with distinguishing the bigger parts of the application from the smaller parts.
    - Downside: Order in which best to take on list items may be harder to figure out. Parts may be harder to test due to heavy use of interfaces or abstract classes rather than full classes.
    - Overall: Advantage upside, dependent on the team.
  + Isolate and elaborate
    - See “Focused design”
    - Overall: Advantage upside, dependent on the team.
  + Build
    - Upside: Grants greater independence to users. Allows users to work on different things at different times and at the same time as other team members. Ideal for projects using Git.
    - Downside: Necessity to go through each individual part to check for compatibility and add use of new classes before final product is completed.
    - Overall: Advantage upside more than anything else. No-brainer.
  + Integrate and learn
    - Upside: Fixes the downside of Build part.
    - Downside: I don’t think any. This is practically just an amendment to Build. Let’s just say it takes time, but everything does.
    - Overall: Advantage upside.
  + Overall: Heavy upside advantage.
* Treat foundation components as a product
  + Upside: Users will likely generate better-quality products. Users may give priority to fixing bugs of completed parts.
  + Downside: Completed quality of the entire product may be lowered.
  + Overall: Advantage downside, unless the group is really and sincerely committed to the project.
* Downside: May be difficult to integrate all at once.
* Overall: Mid-range upside advantage.