2. Explain the steps involved in designflow. (16) [CO4-L1]

A design flow is a sequence of steps to be followed during a design. Some of the steps can be performed by tools,such as compilers or CAD systems;other steps can be performed by hand. In this section we look at the basic characteristics of design flows.

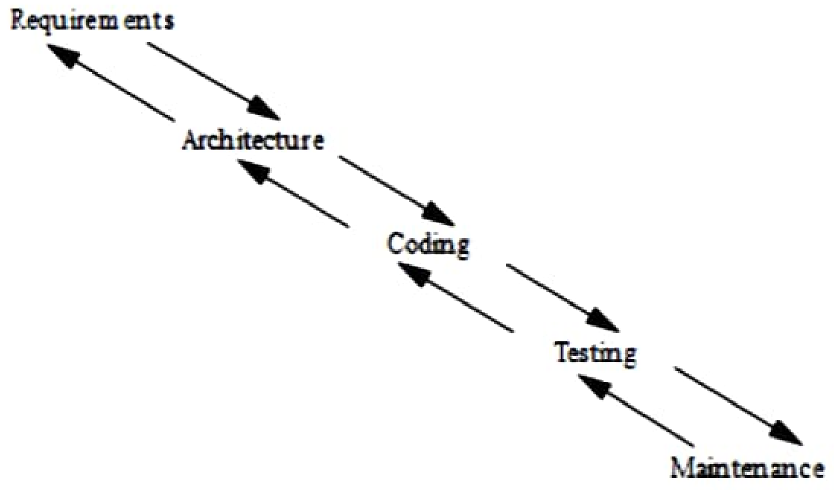
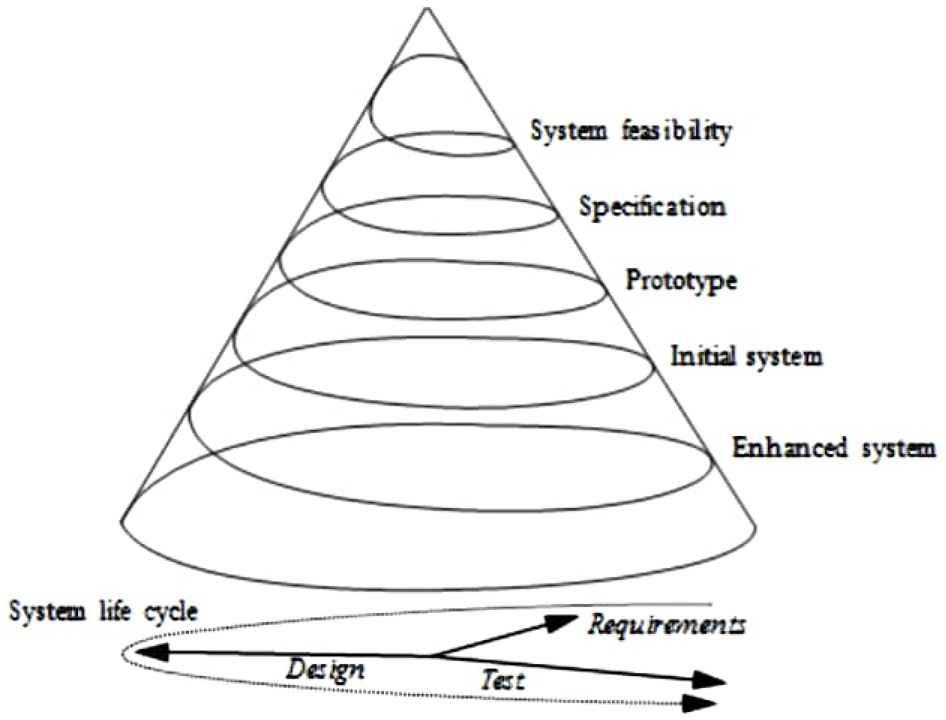


Figure **waterfall model**

Figure shows the **waterfall model** introduced by Royce [Dav90], the first model proposed for the software development process. The waterfall development model consists of five major phases: requirements analysis determines the basic characteristics of the system; architecture design decomposes the function-ality into major components; coding implements the pieces and integrates them; testing uncovers bugs; and maintenance entails deployment in the field, bug fixes, and upgrades. The waterfall model gets its name from the largely one-way flow of work and information from higher levels of abstraction to more detailed design steps (with a limited amount of feedback to the next-higher level of abstraction). Although top–down design is ideal since it implies good foreknowledge of the implementation during early design phases, most designs are clearly not quite so top–down. Most design projects entail experimentation and changes that require bottom–up feedback. As a result, the waterfall model is today cited as an unrealistic design process. However, it is important to know what the waterfall model is to be able to understand and how others are reacting against it.

Figure illustrates an alternative model of software development called the **spiral model** [Boe87]. While the waterfall model assumes that the system is built once in its entirety, the spiral model assumes that several versions of the system will be built. Early systems will be simple mock-ups constructed to aid designers' intuition and to build experience with the system. As design progresses,more com-plex systems will be constructed. At each level of design, the designers go through requirements,construction,and testing phases. At later stages when more complete versions of the system are constructed, each phase requires more work, widening the design spiral. This successive refinement approach helps the designers under-stand the system they are working on through a series of design cycles.



spiral model

The first cycles at the top of the spiral are very small and short, while the final cycles at the spiral's bottom add detail learned from the earlier cycles of the spiral. The spi-ral model is more realistic than the waterfall model because multiple iterations are often necessary to add enough detail to complete a design. However, a spiral methodology with too many spirals may take too long when design time is a major requirement.

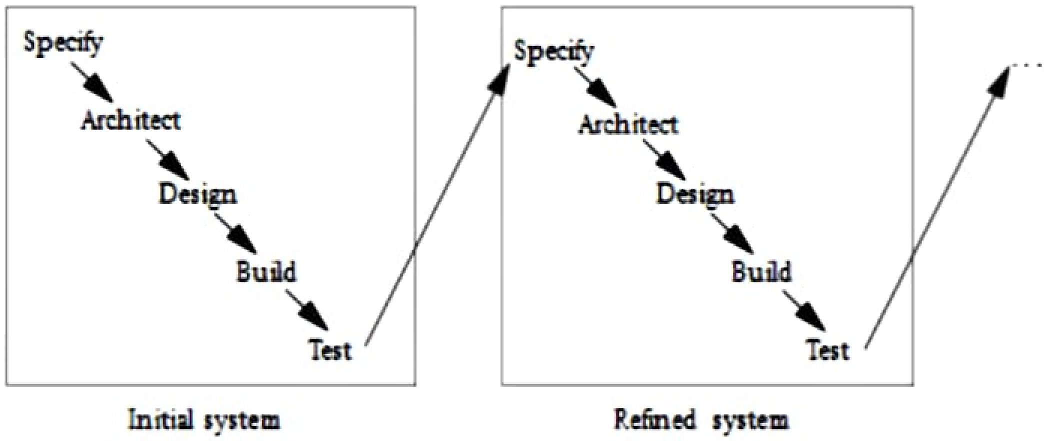


Figure **successive refinement**

Figure shows a **successive refinement** design methodology. In this approach, the system is built several times. A first system is used as a rough proto-type, and successive models of the system are further refined. This methodology makes sense when you are relatively unfamiliar with the application domain for which you are building the system. Refining the system by building several increas-ingly complex systems allows you to test out architecture and design techniques. The various iterations may also be only partially completed;for example,continuing an initial system only through the detailed design phase may teach you enough to help you avoid many mistakes in a second design iteration that is carried through to completion.

Embedded computing systems often involve the design of hardware as well as software. Even if you aren't designing a board, you may be selecting boards and plugging together multiple hardware components as well as writing code.

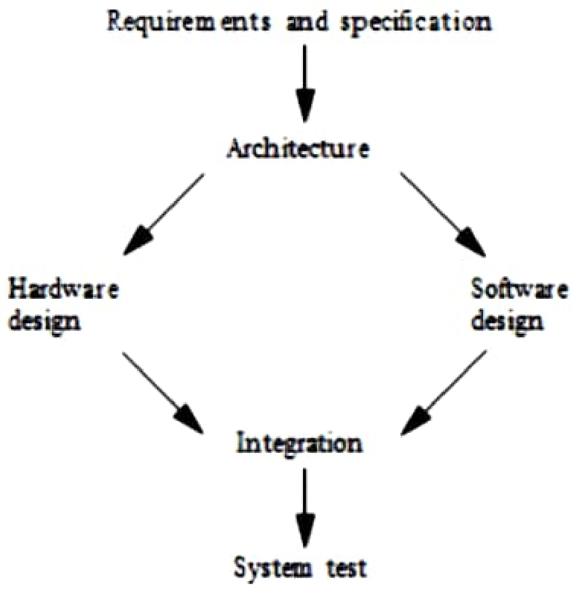
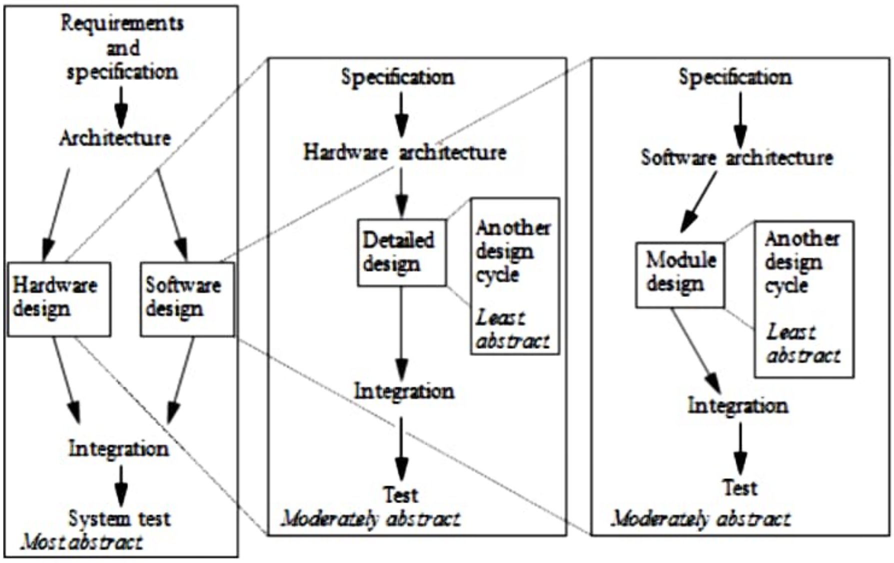


Figure shows a design methodology for a combined hardware/software project. Front-end activities such as specification and architecture simultaneously consider hardware and software aspects. Similarly, back-end integration and testing consider the entire system. In the middle, however, development of hardware and software components can go on relatively independently—while testing of one will require stubs of the other, most of the hardware and software work can proceed relatively independently.

In fact, many complex embedded systems are themselves built of smaller designs. The complete system may require the design of significant software com-ponents, custom logic, and so on, and these in turn may be built from smaller components that need to be designed. The design flow follows the levels of abstrac-tion in the system, from complete system design flows at the most abstract to design flows for individual components. The design flow for these complex sys-tems resembles the flow shown in Figure 9.5. The implementation phase of a flow is itself a complete flow from specification through testing. In such a large project, each flow will probably be handled by separate people or teams. The teams must rely on each other's results. The component teams take their requirements from the team handling the next higher level of abstraction, and the higher-level team relies on the quality of design and testing performed by the component team. Good communication is vital in such large projects.

When designing a large system along with many people, it is easy to lose track of the complete design flow and have each designer take a narrow view of his or her role in the design flow. **Concurrent engineering** attempts to take a broader approach and optimize the total flow. Reduced design time is an important goal for concurrent engineering, but it can help with any aspect of the design that cuts across the design flow, such as reliability, performance, power consumption, and so on. It tries to eliminate "over-the-wall" design steps, in which one designer performs an isolated task and then throws the result over the wall to the next designer, with little interaction between the two. In particular, reaping the most benefits from concurrent engineering usually requires eliminating the wall betwee n design and manufacturing. Concurrent engineering efforts are comprised of several elements.



Cross-functional teams include members from various disciplines involved in the process, including manufacturing, hardware and software design, marketing, and so forth.

Concurrent product realization process activities are at the heart of concurrent engineering. Doing several things at once, such as designing various subsystems simultaneously, is critical to reducing design time.

Incremental information sharing and use helps minimize the chance that concurrent product realization will lead to surprises. As soon as new information becomes available, it is shared and integrated into the design. Cross-functional teams are important to the effective sharing of information in a timely fashion.

Integrated project management ensures that someone is responsible for the entire project, and that responsibility is not abdicated once one aspect of the work is done.

Early and continual supplier involvement helps make the best use of suppliers' capabilities

Early and continual customer focus helps ensure that the product best meets customers' needs