

Software Design and Commercial Software Development

Anthony I. Wasserman
Software Methods and Tools
tonyw@acm.org

The design activities of software development involve making decisions and tradeoffs in the presence of constraints as part of building a system that meets a set of system requirements. The design activities cover many aspects of system development, including:

- architectural design – the creation of subsystems, components, and other program units, along with the mechanisms for communication among the various pieces
- algorithm design – the creation of routines for efficient calculations and data manipulation, based on representations of mathematical formulae and data structures
- database design – the creation of logical and physical data structures to be accessed from secondary storage devices
- human interaction design – the creation of the style of user interaction with the system and the mechanics of the user interface
- robustness design – the assurance of graceful responses to exceptional conditions

Taken collectively, these choices influence everything from aesthetics to performance of the program, as well as the magnitude of the development effort, the difficulty of making changes, and the ease of integrating the program with others.

The design decisions are often influenced by the intended user community of the application. For instance, applications that are intended for use by the development team and their close colleagues may need less attention to robustness and user interface than applications designed for a large, dispersed community of users.

One example of the latter is commercial software, which include software systems (packaged products and online services) used by commercial enterprises and those sold for money. This definition is intentionally broad, and is intended to include packaged software, shareware, components, commercially funded web sites, and open source products, such as those developed through the Apache Software Foundation.

Important characteristics of such products include user expectations of reliability, correctness, robustness, scalability, and ongoing support, including the regular release of updates that add new features and support for new platforms and standards, as well as fixing bugs from previous releases.

Assuring these qualities takes extensive effort. Furthermore, the commercial developer must continue to produce products that deliver value to its users. Development of a commercially-oriented software product introduces additional considerations for the design process, including:

- time to market issues – Commercial developers want to bring their products to market ahead of competitors. To do so, they may favor the use of existing components, licensing software from others rather than writing their own versions. They draw heavily on commonly used architectures, patterns, and frameworks.
- performance design – The vendor must assure acceptable performance for users through

optimizing algorithms and data structures and/or designing web sites for scalability under heavy load conditions. For example, developers of high traffic web sites rely on load balancing and scalability solutions from an application server vendor, as well as clustering solutions from a database vendor and the hardware vendor. These large-scale components are typically assembled into multi-tier architectures deployed across a network of computers augmented by hardware firewalls, routers, and load balancers.

- support for industry “standards” – Software products must “play well with others.” Customers are looking for compliance with particular industry standards, e.g., SOAP and SQL. It is in the vendor's interest to comply, both to meet customer requests and to simplify integration and coexistence with complementary products.

- global market requirements – Many products must be designed so that they can be easily localized and internationalized.

- usability design – This activity goes beyond human interaction design to address the complete set of issues that create a positive customer experience with the product, including such things as a product installer, documentation, examples, online tutorials and help

- product packaging – This includes the company's branding and “look-and-feel”, such as logos, startup splash screens, and other graphical design elements

With all of these requirements and constraints, it is very complex to make the appropriate design decisions and tradeoffs, particularly since individual designers tend to make decisions focused specifically on their area of responsibility. By contrast, the product team must identify the key design issues across the entire product and resolve the tradeoffs to create the best possible overall solution, balancing technical and business/market issues. At this point, there are very few people who have the depth and breadth of experience to do this, so it is usually a team effort or a set of arbitrary decisions by the product leadership.

Much of today's research and education in software design is based on optimizing decisions in a single area, such as database design. While it is essential for a software professional to have that specialized knowledge, it is also important for such a person to have a perspective on how design decisions in a single domain interact with the myriad other decisions that go into product development. In today's commercial environments, the product manager in the marketing organization frequently lacks the technical knowledge needed to architect and build the product, while the engineering manager often fails to fully understand the business issues that impose additional constraints on software development.

These issues have implications for creating a science of design focused on software. In building a taxonomy and a body of knowledge for this emerging area, it is important not only to include specific fields of design, but also to address tradeoffs across these fields, the effect of global constraints, and the influence of external issues, e.g., cost and time to market, on the design and development processes.

Tony Wasserman has divided his career between academia and the software industry. As a University of California, San Francisco, professor, he developed the User Software Engineering methodology, which addressed both HCI and SE issues. He edited (with Peter Freeman) *Software Design Techniques: a Tutorial* (IEEE Press, 4th ed., 1983) and *Software Engineering Education: Needs and Objectives* (Springer Verlag, 1976). Tony was founder and CEO of Interactive Development Environments, which built the multi-user Software through Pictures modeling environment. More recently, he was VP of Engineering for a dotcom startup, and then as VP of Bluestone Software (subsequently acquired by HP), with responsibility for their mobile middleware products. He is now an independent consultant on software product development.