Construction and Deployment

Week 11

Agenda (Lecture)

- Construction and Deployment

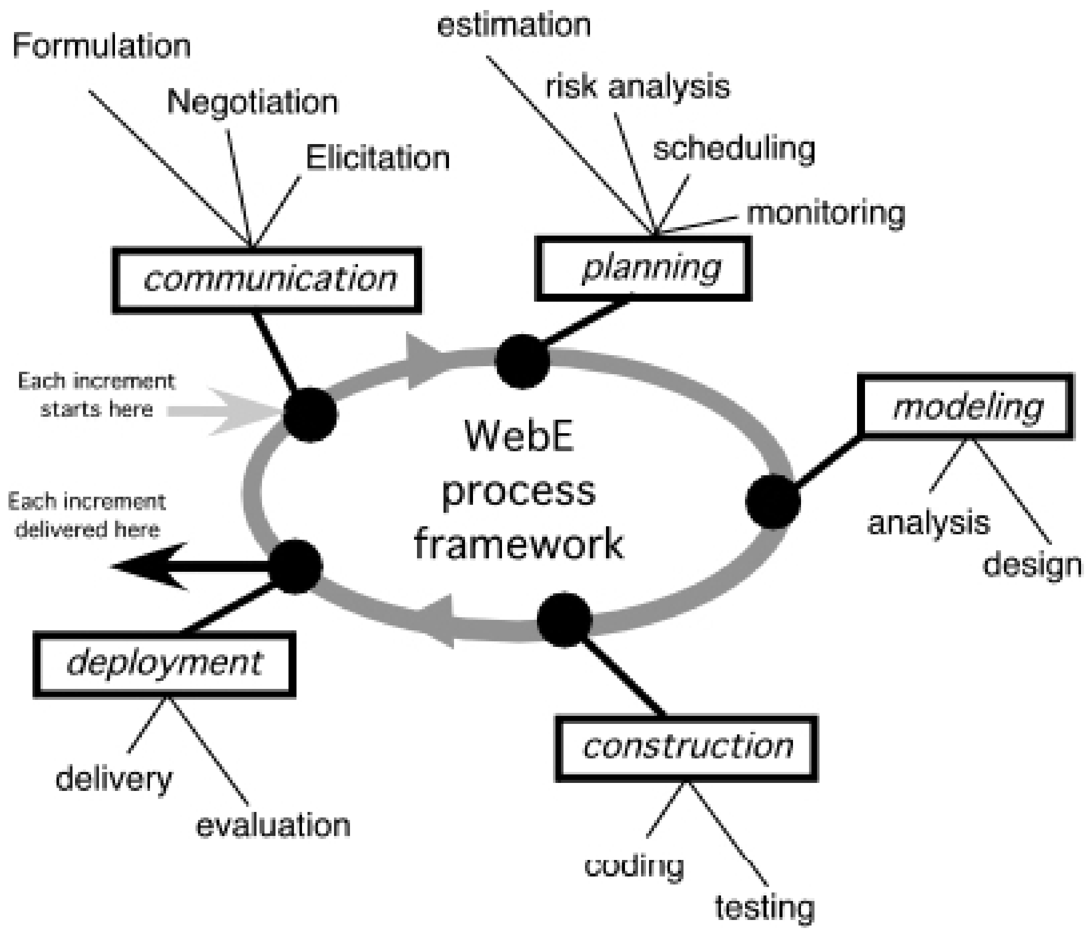
Agenda (Lab)

- Implement your web apps based on design documents
- Weekly progress report

Announcement

- Midterm II
  - Monday, 5/2
  - Short-answer questions
  - Scope
    - Chapters 8, 9, 10 and 11
    - Your project

WebE Process Activities & Actions



Some Basic Principles

- Keep the development environment and the production environment separate. Do *not* develop directly on the servers that are accessible to your users!
- Provide the developers with an environment that facilitates their productivity.
- Where possible, undertake testing in the same environment that your users will see.
- Particular development and production environments should be integrated with the functional architecture.
  – For example, if a content management system (CMS) is adopted (Chapter 16), then the way in which the CMS is used by developers to create, integrate, and publish components will need careful consideration.

Construction - I

- Encompasses a set of tasks that lead to an operational WebApp that is ready for delivery to end users.
  - *Selection* involves the identification of relevant preexisting components (or objects within components) that can be reused within the proposed design
  - *Coding* covers the adaptation of existing components or creation of new components and may involve the direct creation of HTML or scripting-language source code or the automatic generation of code using an intermediate design representation of the component to be built

Construction - II

- *Content management* (Chapter 16) involves the creation, migration, and structuring of content. This encompasses content creation, implementation of database schemas, or conversion of legacy content into, say, XML.
- *Authoring* involves the integration of raw content with the graphic design (layout) and the mapping of the content into the screens and pages.
- *Integration* comprises the linking of the code, content, and presentation into the final components to be released.
- *Refactoring* is an iterative action that "polishes" the implemented components to improve their structure and clarity and remove redundant code.
- *Testing* involves the verification that the various components and objects are correct.

Construction: Preparation Principles

- Before you create a single element of content, design a single Web page or write one line of code, be sure you:
  - Understand the problem you’re trying to solve.
  - Understand basic WebApp design principles and concepts.
  - Pick a language that meets the needs of the component to be built and the environment in which it will operate.
  - Select an environment that provides tools that will make your work easier.
  - Create a set of unit tests that will be applied once the component you create is completed.

Construction: Selection Principles

- As you select existing, reusable components and objects, be sure you
  - Take into account the constraints of the technical environment.
  - Match the components to the information and functional environments.
  - Consider the skills and knowledge of both the developers and likely maintainers.
  - Consider issues of IP **(**Intellectual Property), the proprietary nature of the components, and whether they are portable.

Construction: Coding Principles

- As you begin writing code, be sure you
  - Write code that is self-documenting.
  - Constrain your algorithms by following structured programming practices.
  - Select data structures that will meet the needs of the design.
  - Understand the functional architecture and create interfaces that are consistent with it.
  - Keep conditional logic as simple as possible and ensure it is testable.
  - Adopt coding styles that aid in readability (e.g., select meaningful identifier names and follow other local coding standards). A wide variety of links to coding standards can be found at the Literate Programmer website **www.literateprogramming.com/fpstyle.html**.

Construction: CM Principles

- As content is managed, be sure you
  - Select data structures that will meet the needs of the design.
  - Understand the information architecture and create content and navigational structures that are consistent with it.
  - Ensure consistency in the formats and data structures.
  - Avoid reliance on proprietary data formats.
  - Treat your content as publishable material—not as software.

Construction: Authoring and Integration Principles

- **Authoring:** As you create Web pages (or templates for pages), be sure you
  - Continually consider issues of usability.
  - Remember to address issues of accessibility.
  - Understand how your users *will* react, not how *you want* them to react.
  - Learn from competitors.
- **Integration**: As you integrate your components and objects, be sure you:
  - Keep backups – preferably in some form of version control system. You need to be able to rewind the WebApp to earlier versions.
  - Look for component interface mismatches or inconsistencies.
  - Take the opportunity to identify components that need refactoring.

Construction: Refactoring and Testing Principles

- **Refactoring principles.** As you refactor your WebApp be sure you
  - Understand common refactorings (see the list of example refactorings on the Refactoring Homepage at www.refactoring.com/catalog/index.html)
  - Refactor often, and in small steps, when the opportunity arises (but don't change unnecessarily—see the section on not changing things if they are working on the Cunningham & Cunningham, Inc. website at http://c2.com/cgi/wiki?IfItIsWorkingDontChange
  - Make sure the implementation communicates the design in an obvious way.
- **Testing principles.** After you've completed your first components, be sure you
  - Conduct a walkthrough when appropriate.
  - Perform unit tests and correct errors you've uncovered.
  - Select tests that are most likely to locate errors rather then to hide them.

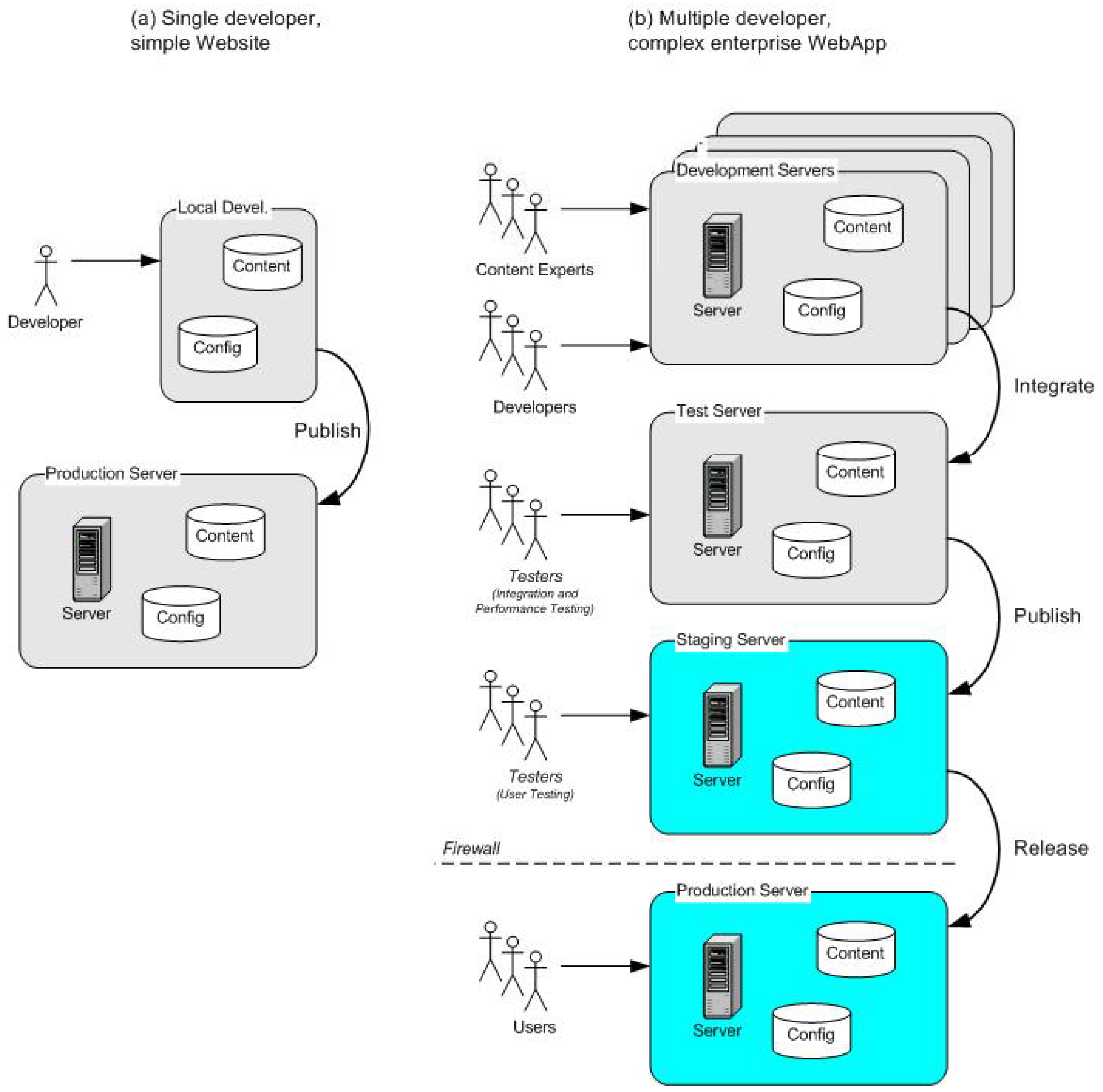
Deployment

- Encompasses three actions: packaging, release, and evaluation
- Deployment happens not once, but a number of times as the WebApp moves toward completion
- Can be accomplished in a very fine-grained manner (not always advisable) by releasing new components from the staging server to the production server after the individual components have been tested
- Each package-release cycle provides end users with an operational WebApp increment that provides usable functions and features.
- Each evaluation cycle provides the WebApp team with important guidance that results in modifications to the content, functions, features, and approach taken for the next increment.

Deployment Environments

- Will often be multiple environments for use during deployment
  - **Development servers.** Developers and authors use these servers to perform all authoring and unit-level testing.
  - **Test server.** Once developers complete the unit-level testing of their components, they can integrate them for verification within the full WebApp environment.
  - **Staging server.** This server is intended to provide a mirror of the full production environment.
    - comprehensive user testing occurs without having to release the WebApp to the production server and hence the full user population.
  - **Production server.** When the WebApp is ready to be released for use by all users, it is placed on this server.

Deployment Environments



Deployment Principles

- *Principle 1:* Customer expectations for the WebApp increment must be managed.
- *Principle 2:* A complete delivery package should be assembled and tested.
- *Principle 3:* A support regime must be established before the WebApp is delivered.
- *Principle 4:* Buggy WebApps should be fixed first, delivered later
  - (except where being first really is more important than the possibility of adverse customer reactions)!

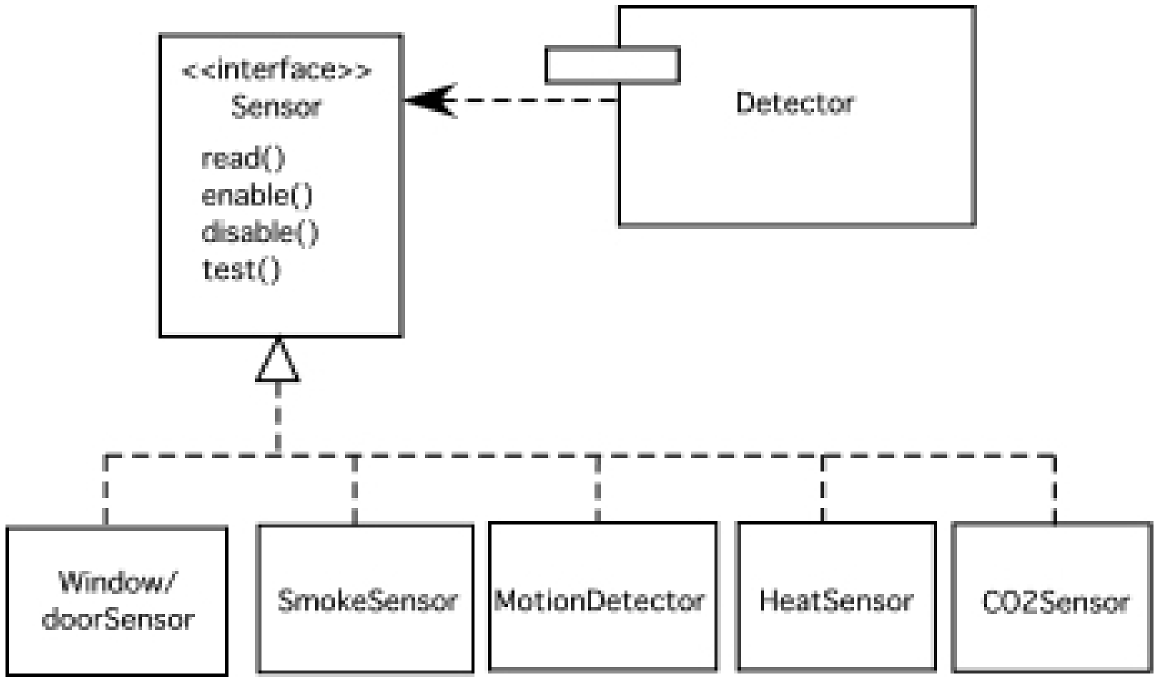
Using CMS during Deployment

- Content Management Systems are change management tools. They provide the following generic benefits [Dar00]:
  - Ensure that published content is correct and consistent with other content
  - Control and track changes to content including the implementation of mechanisms that enforce who may make a change
  - Verify that the correct version of a function has been implemented and that the version corresponds properly to versions of related functions
  - Enable the WebE team to rebuild a WebApp rapidly if the system fails or crashes
  - Allow the team to roll back to a previous version if serious unforeseen errors are encountered in the most recent version
- As the size and complexity of a WebApp grows, the need for comprehensive version control and a robust CMS at every stage of construction and deployment also grows.

Components

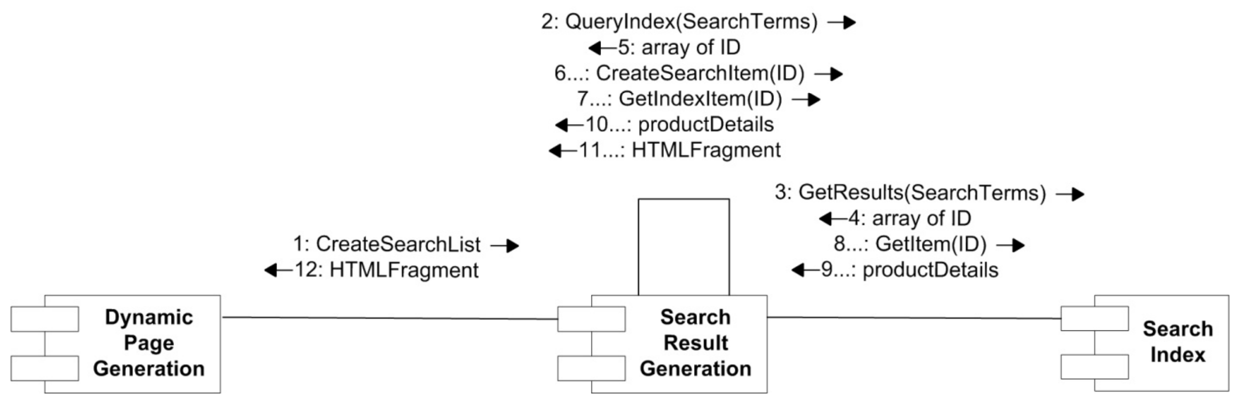
- *Principle 1:* A designer should specify a WebApp component in a way that allows it to be extended (within the information or functional domain that it addresses) without the need to make internal (content, code, or logic) modifications to the component itself.
- *Principle 2:* A component that uses a base class should continue to function properly if a class derived from the base class is passed to the component instead.
- *Principle 3:* The more a component depends on other concrete components (rather than on abstractions such as an interface), the more difficult it will be to extend.
- *Principle 4:* The designer should create a specialized interface to serve each major category of clients.

Extensibility



Component Design Steps - I

- Step 1. Identify all information and functional design classes that correspond to the problem domain.
- Step 2. Identify all information interaction and functional design classes that correspond to the infrastructure domain.
- Step 3. Elaborate all design classes that are not acquired as reusable components.
- Step 4a. Specify message details when classes or components collaborate.



Component Design Steps - II

- Step 4b. Identify appropriate interfaces for each component.
- Step 4c. Elaborate attributes and define data types and data structures required to implement them.
- Step 4d. Describe processing flow within each functional component in detail.
- Step 5. Describe which data are to be persistent data (Web pages, databases, etc.) and which are to be dynamically constructed.
- Step 6. Develop and elaborate behavioral representations for a class or component.
- Step 7. Elaborate deployment diagrams to provide additional implementation detail.
- Step 8. Factor every component-level design representation and always consider alternatives.