Project Status Report
Due Sundays by 11pm
One per Project

• Link on the Calendar page of the class website: http://www.cs.washington.edu/education/courses/cse403/11wi/weekly-status.html

• Why is it useful (WIFM)?
  • Brings your team, your customer, and the executives up to speed on your status, to provide an opportunity for reflection, adjustment and feedback
Highlights
• <list your biggest accomplishments of the week, at most 3>

Lowlights
• <list any major problems you ran into>

Goals for next week
• <what’s your week plan, list 3-5>

Issues
• <what issues are facing you right now; list at most 3>

Risks
• <what are your biggest risks in the future; list at most 2>

Staff help required
• <list any questions for/help you need from the 403 staff>
Highlights (of last week)
• Created a project wiki
• Established regular group meeting times
• Formed a plan and schedule for the SRS assignment

Lowlights (of last week)
• Didn’t complete the customer meetings and have to get that information asap

Goals (for this coming week)
• Commit to a feature set
• Have draft SRS in hand by Wed, and complete by Friday
• Get familiar with dev tools and resources

Issues
• Dividing the work between the team is a challenge – perhaps creating subgroups will help

Risks
• We don’t share a common vision and will diverge during the design step
• Handhelds aren’t available for us to use

Staff help required
• Need accounts and space on cubist
• Need answer on access to mobile devices
Readings

• Why?
• Why the summaries?
• 1 summary per week!
• Papers available on campus.
Notes on the UI prototyping

• The prototype should not be better than the final product

• First impressions are important
Architecture

MIT Stata Center by Frank Gehry
Why architecture?

“Good software architecture makes the rest of the project easy.”
Steve McConnell, Survival Guide
The basic problem

How do you bridge the gap between requirements and code?
One answer

Requirements

Code

a miracle happens
A better answer

Requirements

Provides a high-level framework to build and evolve the system

Software Architecture

Code
What does an architecture look like?
Box-and-arrow diagrams

Very common and hugely valuable. But, what does a box represent? An arrow? A layer? Adjacent boxes?
An architecture: components and connectors

- **Components** define the basic computations comprising the system and their behaviors
  - abstract data types, filters, etc.
- **Connectors** define the interconnections between components
  - procedure call, event announcement, asynchronous message sends, etc.
- The line between them may be fuzzy at times
  - Ex: A connector might (de)serialize data, but can it perform other, richer computations?
A good architecture

• Satisfies functional and performance requirements
• Manages complexity
• Accommodates future change
• Is concerned with
  – reliability, safety, understandability, compatibility, robustness, ...
Divide and conquer

• Benefits of decomposition:
  – Decrease size of tasks
  – Support independent testing and analysis
  – Separate work assignments
  – Ease understanding

• Use of abstraction leads to modularity
  – Implementation techniques: information hiding, interfaces

• To achieve modularity, you need:
  – Strong cohesion within a component
  – Loose coupling between components
  – And these properties should be true at each level
Qualities of modular software

- decomposable
  - can be broken down into pieces

- composable
  - pieces are useful and can be combined

- understandable
  - one piece can be examined in isolation

- has continuity
  - change in reqs affects few modules

- protected / safe
  - an error affects few other modules
Interface and implementation

• **public interface**: data and behavior of the object that can be seen and executed externally by "client" code

• **private implementation**: internal data and methods in the object, used to help implement the public interface, but cannot be directly accessed

• **client**: code that uses your class/subsystem

  – Example: *radio*
    • public interface is the speaker, volume buttons, station dial
    • private implementation is the guts of the radio; the transistors, capacitors, voltage readings, frequencies, etc. that user should not see
UML diagrams

• UML = universal modeling language

• A standardized way to describe (draw) architecture

• Widely used in industry
Properties of architecture

• Coupling
• Cohesion
• Style conformity
• Matching
• Errosion
Loose coupling

- *coupling* assesses the kind and quantity of interconnections among modules

- Modules that are loosely coupled (or uncoupled) are better than those that are tightly coupled

- The more tightly coupled two modules are, the harder it is to work with them separately
Tightly or loosely coupled?

[Diagram of software components and their relationships]

- User Interface
- Graphics
- Data Storage
- Application Level Classes
- Business Rules
- Enterprise Level Tools

[Lines connecting the components with arrows indicating relationships]
Tightly or loosely coupled?

Diagram:

- User Interface
- Graphics
- Data Storage
- Application Level Classes
- Business Rules
- Enterprise Level Tools

Relationships:

- User Interface related to Graphics
- Data Storage related to Application Level Classes
- Business Rules related to Enterprise Level Tools

End Points:

- End1
- End2
- End3
- End4
- End5
- End6
- End7
- End8
- End9
- End10
- End11
- End12
- End13
- End14
- End15
Strong cohesion

• *cohesion* refers to how closely the operations in a module are related

• Tight relationships improve clarity and understanding

• Classes with good abstraction usually have strong cohesion

• No schizophrenic classes!
Strong or weak cohesion?

class Employee {

public:

    ... 
    FullName GetName() const;
    Address getAddress() const;
    PhoneNumber GetWorkPhone() const;
    ... 
    bool IsJobClassificationValid(JobClassification jobClass);
    bool IsZipCodeValid(Address address);
    bool IsPhoneNumberValid(PhoneNumber phoneNumber);
    ... 
    SqlQuery GetQueryToCreateNewEmployee() const;
    SqlQuery GetQueryToModifyEmployee() const;
    SqlQuery GetQueryToRetrieveEmployee() const;
    ... 
}
An architecture helps with

- System understanding: interactions between modules
- Reuse: high-level view shows opportunity for reuse
- Construction: breaks development down into work items; provides a path from requirements to code
- Evolution: high-level view shows evolution path
- Management: helps understand work items and track progress
- Communication: provides vocabulary; pictures say $10^3$ words
Architectural style

• Defines the vocabulary of components and connectors for a family (style)

• Constraints on the elements and their combination
  – Topological constraints (no cycles, register/announce relationships, etc.)
  – Execution constraints (timing, etc.)

• By choosing a style, one gets all the known properties of that style (for any architecture in that style)
  – Ex: performance, lack of deadlock, ease of making particular classes of changes, etc.
Styles are not just boxes and arrows

• Consider pipes & filters, for example (Garlan and Shaw)
  – Pipes must compute local transformations
  – Filters must not share state with other filters
  – There must be no cycles
• If these constraints are violated, it’s not a pipe & filter system
  – One can’t tell this from a picture
  – One can formalize these constraints

![Diagram](scan -> parse -> optimize -> generate)
The design and the reality

• The code is often less clean than the design

• The design is still useful
  – communication among team members
  – selected deviations can be explained more concisely and with clearer reasoning
Architectural mismatch

• Mars orbiter loss

NASA lost a 125 million Mars orbiter because one engineering team used metric units while another used English units for a key spacecraft operation.
Views

A view illuminates a set of top-level design decisions:
• how the system is composed of interacting parts
• where are the main pathways of interaction
• key properties of the parts
• information to allow high-level analysis and appraisal
Importance of views

Multiple views are needed to understand the different dimensions of systems

- Functional Requirements
- Performance (execution) Requirements
- Packaging Requirements
- Installation Requirements

- Use Case View
- Design View
- Implementation View
- Process View
- Deployment View

Booch
Web application (client-server)
Model-View-Controller

Separates the application object (model) from the way it is represented to the user (view) from the way in which the user controls it (controller).
Pipe and filter

Pipe – passes the data

Filter - computes on the data

top | grep $USER | grep acrobat

Each stage of the pipeline acts independently of the others.
Can you think of a system based on this architecture?
Blackboard architectures

- *The knowledge sources*: separate, independent units of application dependent knowledge. No direct interaction among knowledge sources.

- *The blackboard data structure*: problem-solving state data. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.

- *Control*: driven entirely by state of blackboard. Knowledge sources respond opportunistically to changes in the blackboard.

Blackboard systems have traditionally been used for applications requiring complex interpretations of signal processing, such as speech and pattern recognition.
Hearsay-II: blackboard

Hearsay-II Instance of Blackboard