Topics for Today

- Course Overview
  - Introductions
  - Course Syllabus
  - Learning Mode
- What to expect
- Intro to software architecture
- Principles of sound documentation
- Team building activity
Introductions

- Tell us two things about you. More if you’d like
- What do you hope to get from this course?

Course Syllabus
Learning Modes (Flipped Classroom)

- Lecture
- Readings/HW
- Group Discussions
- Projects and Writing Assignments

Prerequisites

- Understanding of basic software development life-cycle
- Understanding of UML notation (you might need to review as we go)
  - Used throughout the semester
- Ability to read/understand source code in some high-level development language
Assignments and Grading

- Tests: midterm and final
  - In class – open book (Maybe)
    - Go to lecture
    - Keep up with the readings and course work
- Project
  - Define and critique software architecture for a software-intensive system
    - We will discuss the project at a later date
    - Can be work-related; can be open-source
  - Describe likely extensions
  - Evolve the architecture: either extend it or refactor it.
- Communication
  - IBM’s SmartCloud

Expected Workload

- The project will be a significant component of the course
  - Builds on previous coursework
  - This class might be quite time consuming
  - I will try to help you out, but…
- Planning and scheduling your time is essential
- Make sure you spread out the work
  - You will have problems trying to “cram”
  - I am available to help
Communicating

- Problems with course material
  - Talk to me
- Problems with course delivery
  - Talk to me
- Problem with workload
  - Talk to me
- Problem with your team
  - Talk to me

It is your responsibility to address problems early on

Architecture Overview
What is Architecture?

- Let’s read the definitions
  - SEI website contains over 150 of those: http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

- What do you think?

- Is there a difference between architecture and design? Explain

What is Architecture?

- [Garlan & Perry]: Software architecture is "the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."

- [Bass, Clements, and Kazman]: The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

- [IEEE 1471]: Software architecture is the fundamental organization of a system, embodied in its components, their relationships to one another and the environment, and the principles governing its design and evolution.
Why Should You Care?

- Good Things are Well Architected
  - Internet
  - World Wide Web
  - Airplanes
  - ATMs
- Good architecture is hard
- But you have a role to play
Project Failure Rate (by Size)

IT Project Failure Rates by Size and Methodology

Data: Scott Ambler “The July 2010 State of the IT Union Survey”

Earlier Architectural Decisions do Make a difference
A good architecture definition

- Set of design decisions (both structural and behavioral) that if done incorrectly may cause the project to be cancelled.

Clements, et. Al,

Misconceptions about architecture

- Architecture is just paper
- Architecture and design are the same things
- Architecture and infrastructure are the same things
- `<my favorite technology>` is the architecture
- A good architecture is the work of a single architect
- Architecture is simply structure
- Architecture can be represented in a single blueprint
- System architecture precedes software architecture
- Architecture cannot be measured or validated
- Architecture is a science
- Architecture is an art
**Architecture is just paper**

- A system’s architecture ultimately resides in executable code
- A system’s architecture may be visualized in models
- Every system has an architecture; some architectures are made manifest and visible, many others are not

**Architecture is design**

- All architecture is design, but not all design is architecture
- Architecture focuses on *significant* design decisions, decisions that are both structurally and behaviorally important as well as those that have a lasting impact on the performance, reliability, cost, and resilience of the system
- Architecture involves the how and the why, not just the what
Architecture is design

- So when do you stop architecting and start designing?

- What is the difference between architectural design and detailed design?
  - You mean architectural design is not detailed?

Architecture is infrastructure

- Infrastructure is an integral and important part of architecture, but there is more to architecture than just infrastructure

- Significant patterns will manifest themselves at many different levels and dimensions of a system

- Too narrow a view of architecture will lead to a very pretty infrastructure, but the wrong infrastructure for the problem at hand
<my favorite technology> is the architecture

A given technology only serves to implement some dimension of an architecture
- The network is the architecture
- The database is the architecture
- The transaction server is the architecture
- J2EE is the architecture

Architecture is more than just a list of products

Technology shapes an architecture, but a resilient architecture should never be bound to all of the technologies that form it

Architecture is the work of a single architect

Conceptual integrity is essential, but the complexity of most interesting systems leads development to be a team sport

Fred Brooks (1975), but then Fred Brooks (1995)

The architecture team is not
- a committee
- a problem clearinghouse
- an ivory tower

The architecture team
- Needs a clear leader
- Requires a mix of specialties
- Manifests itself at many levels in the system
Architecture is structure

- Architecture does involve structure, decomposition, and interfaces
- Architecture also involves behavior
- A system’s architecture is always projected to a given context

Architecture is flat

- Architecture is flat only in trivial systems
- Multiple stakeholders with multiple concerns lead to multiple views with multiple blueprints
- Using a single blueprint to represent all or most of system’s architecture leads to a semantic muddle
- The 4+1 view model has proven to be both necessary and sufficient for most interesting systems
System architecture comes first

- Software has a longer life than hardware
- Complex systems require well-informed hardware/software tradeoffs, which cannot be made in a strict sequence
- Forcing a hardware-first process typically leads to problems

Architecture can’t be measured

- The very purpose of a blueprint is to provide a tangible artifact that can be used to visualize, specify, construct, document - and reason about - a system
- A system’s architecture can be used to
  - Mitigate technical risks through the release of a continuous stream of executables
  - Improve learning and understanding and communicate important decisions
  - Accelerate testing and attack integration risks
  - Set expectations
  - Break in the development environment and the team
Architecture is a science

There exists only a modest body of knowledge about software architecture

Scientific and analytical methods are lacking; those that do exist are hard to apply

There is no perfect design; architecture involves the management of extreme ambiguity and contradiction

Experience counts: the best architects are grown, not born

Architecture is an art

Even the best architects copy solutions that have proven themselves in practice, adapt them to the current context, improve upon their weaknesses, and then assemble them in novel ways with very modest incremental improvements

The “artsy” part of software architecture is minimal

An architectural process can be established with intentional artifacts, clear activities, and well-defined
Architectural Qualities

Performance
Ease of Maintenance
Testability
Security
Usability

Architectural Choices - 1

- What if your system requires high performance? What do you need to pay attention to?
- What if security is the essential driver?
- What if it is modifiability?
- High Accuracy?
- Incremental deployment?

Spend 3 minutes on each. You can do this alone or with the rest of the class.
What if your system requires high performance? What do you need to pay attention to?

- Exploit potential parallelism and decompose system into synchronizing processes
- Manage interprocess and network communication and data access frequency
- Pay attention to latencies and throughput
- Identify and solve potential performance bottlenecks

Security?

- Legislate usage relationships and communication restrictions
- Identify most sensitive parts
- Introduce special elements with high degree of trust
Modifiability?
- Carefully separate concerns so that a single change does not ripple through system

High Accuracy?
- Pay attention to how data elements are defined and used
- And how their values flow through the system
Incremental deployment?
- Keep dependency relationships among pieces untangled
  - Avoid “nothing works until everything works” syndrome

Stakeholders
- Manager; needs to know how to back up system data for disaster recovery
- Security and compliance: needs to know how data is logically and physically secured
- User: How is this going to make my life better?
- Developer: What are the system interfaces I need to respect?
- Management: what is the business case for the system?
Stakeholders

- Analyst
- Architect
- Business manager
- Conformance checker
- Customer
- DB Admin
- Deployer
- Designer
- Evaluator
- Implementer
- Integrator
- Maintainer
- Network admin.
- Product line manager
- Project manager
- Representative of external systems
- System engineer
- Tester
- User

Role of Software Architect

- Define the functional interface of the system
- Determine important qualities of the system
  - Especially, acceptance criteria
- Revise architecture as implementation issues arise
- *Explain and Persuade* different stakeholders on architectural details
The Views and Beyond Approach - 1

- Views provide the organizing principle for architecture documentation.
- Beyond views include additional information
- An architecture document should be helpful to the people who depend on it in order to do their work.

The Views and Beyond Approach - 2

- Views and Beyond is not a method, but a collection of techniques:
  1. Find out what architecture information stakeholders need.
  2. Provide that information to satisfy the needs.
  3. Capture the information in views, as well as beyond-view information.
  4. Package the information in a form useful to its stakeholders.
  5. Review the result to see if it has satisfied stakeholders’ needs.
Objectives

- help you decide what information to capture about an architecture
- provide guidelines and notations for capturing the necessary information
- provide examples of architecture documentation
- answer this question:
  - How do you record an architecture so that others can successfully use it, maintain it, and build a system from it?

The term document doesn’t necessarily mean information printed on paper. Documentation can take many forms.

Why Documenting a Software Architecture?

*Doing business without advertising [or designing an architecture without documenting it] is like winking at a girl in the dark. You know what you’re doing, but nobody else does.*

- Steuart Henderson Britt
Why Documenting a Software Architecture? - 1

- Architecture is the blueprint for the system and the project that develops it
  - It defines the work assignments.
  - It is the primary carrier of quality attributes.
  - It is the best artifact for early analysis.
  - It is the key to post-deployment maintenance and mining.

- To be useful, this blueprint must be understood.

- To be understood, it must be communicated.

- Documentation speaks for the architect, today, tomorrow, and 20 years from now.

Why Documenting a Software Architecture? - 2

- Architecture documentation contributes to architecture design
  - Documentation enables an artifact-driven approach to software design
  - Completing the design artifact means we’ve completed the design
  - Documentation establishes the set of design decisions needed to establish/maintain the architecture
  - Making those design decisions means completing the architecture.
### Why Documenting a Software Architecture? - 3

- Documentation clarifies the line between *architectural* and *non-architectural design* decisions.
  - *Non-architectural design* is preferred over *detailed design*. Architectural decisions can be quite detailed!
  - Architectural design decisions affect the system’s ability to deliver on its behavioral and quality goals.
  - Architectural design decisions are documented in the architecture document.

### Why Documenting a Software Architecture? - 4

- Architecture documentation has three fundamental uses:
  1. **Education**, introducing people to the system:
     - new members of the team, external analysts, the customer, or even a new architect
  2. **Communication**, as a vehicle among stakeholders and to/from the architect
  3. **Analysis**, especially for the quality attributes that the architecture design enables the system to deliver
Uses and Audience

- Architecture documentation must support these purposes:

  1. **Education**: It should be sufficiently abstract to be quickly understood by new team members.

  2. **Communication**: It should be sufficiently concrete to serve as a blueprint for construction.

  3. **Analysis**: It should provide enough information to serve as a basis for analysis.

Business Case for Architecture Documentation

- Project activities will be less costly with high quality, up-to-date documentation than they would be otherwise.

- The effort saved from architecture documentation should outweigh the cost to create it.
Principles of Sound Documentation

*I have made this letter rather long only because I have not had time to make it shorter.*

Blaise Pascal

Seven Principles of Sound Documentation

- These principles apply to all documentation, not just that for software architectures
  1. Write from the reader’s point of view.
  2. Avoid unnecessary repetition.
  3. Avoid ambiguity.
  4. Use a standard organization.
  5. Record your rationale.
  6. Keep documentation current, but not too current.
  7. Review documentation.
Write from the reader’s point of view

- Determine who the readers are
- Determine what readers will want to know
- Make the information concise and easy to find
  - Don’t make too many assumptions about what the readers know
- Your readers will appreciate your effort and be more likely to read your document
  - (Which will make the business case for architecture documentation)

Avoid Unnecessary Repetition

- Each kind of information should be recorded in only one place. This makes documents easier to use and change.
- Repetition often confuses the reader, especially when information is repeated in slightly different ways.
  - The reader wonders
    - “Was the difference intentional?”
    - “If so, why?”
    - “If not, which way is correct?”
- When is repetition okay?
  - Summary and overview
  - When not repeating will cause the reader to spend time flipping back through pages
    - Hyperlinks?
Avoid Ambiguity -1

- Documentation is for communicating information and ideas. If the reader misunderstands because of ambiguities, the documentation has failed.

- Even “simple” concepts can confuse. For example, what does the arrow above mean?
  - C1 calls C2?
  - Data flows from C1 to C2?
  - C1 instantiates C2?
  - C1 sends a message to C2?
  - C1 is a subtype of C2?
  - C2 is a data repository and C1 is writing data to C2?
  - C1 is a repository and C2 is reading data from C1?

Avoid Ambiguity -2

- Precisely defined notations/languages help avoid ambiguity.

- If your documentation uses a graphical language, *always include a key!*
  - It can point to the language’s formal definition.
  - It can give the meaning of each symbol. (Don’t forget the lines!). If color or position is significant, indicate how.

- Be sure to make the key meaningful: don't just say “element” and “relation.”
  - Different element and relation types should have different symbols.
Avoid Ambiguity -3

Example of Keys

Use a Standard Organization - 1

- Establish a standard organization, make sure that your documents follow it, and make sure that readers know what it is.

- A standard organization
  - helps the reader navigate and find information
  - tells the writer what to document, where it belongs
  - helps plan the work and measure the work left to be done
  - lets the writer record information as soon as it is known, in whatever order it is discovered
  - embodies completeness rules and helps with validation.
Corollary #1: Organize the documentation for ease of reference
- A document may be read from cover to cover only once, if at all
- A successful document will be referred to hundreds or thousands of times

Make information easy to find.

How do you do that?
- Comprehensive index
- Annotated TOC
- Reader’s guide
- Keywords, tables of figures…

Corollary #2: Don’t leave incomplete sections blank; instead, mark them “to be determined.”
- Better: “TBD by Revision 2.6”
- Better still: “TBD by 14 November 2011”

Why?

Corollary #3: If a section doesn’t apply, don't leave it blank or delete it;
- mark it “not applicable.”
- Better: “not applicable because…”

Why?
**Record Your Rationale**

- Why did you make certain design decisions?

- Next week or next year, how will you remember? How will the next architect know?

- Recording your rationale requires discipline but can save enormous time in the long run.

- Record significant rejected alternatives as well.
  - helps avoid wasting time on the same dead ends in the future
  - might explain when a dead end is no longer dead

**Keep Documentation Current, But Not Too Current.**

- This rule applies throughout the entire life cycle of the system

- Documentation that is incomplete or out of date
  - does not reflect the truth
  - disobeys its own rules about form and internal consistency
  - is not likely to be used

- Documentation that is kept current
  - can provide quick and efficient answers to questions about the software
  - is more likely to be used
Help instill a *documentation-based culture* in your organization by letting documents answer questions.

- The architect’s first answer should be “Here is where you can find that information in the documentation.”
- If the information is missing, update the document.
- Make sure the next release contains the information.
- Send the message that the *documentation* is the preferred, authoritative source for information.

Contrast that to the architect who happily answers questions every time the phone rings.

Don’t keep it *too* current:

- During the design process, decisions are considered and reconsidered frequently.
- Releasing too often will result in unnecessary expense and frustration among the readers.
- Determine points in the development process when up-to-date documentation will be released.
- Follow a release strategy or *rhythm* that makes sense for your project.
Review Documentation

- Only the intended users of a document can tell you if it contains the right information, presents the information in a useful way, and satisfies their needs.
- Plan to review your documents with representatives of the stakeholders for whom it was created.

Summary

- Certain principles apply to all documentation, not just to that for software architectures.
- Use them as guidelines to help you write high-quality documentation.
- Use them when you are reviewing other people’s documentation, to help suggest specific improvements.
- This semester, will continue to learn about the better ways of documenting software architecture using different views that are intended for describing different elements for different stakeholders.
Reading for next week

- DSA: Prologue
- DSA: Chapters 1 and 2

Marshmallow challenge
Things to Understand

- **Build the Tallest Freestanding Structure**: The winning team is the one that has the tallest structure measured from the table top surface to the top of the marshmallow. That means the structure cannot be suspended from a higher structure, like a chair, ceiling or chandelier.
- **The Entire Marshmallow Must be on Top**: The entire marshmallow needs to be on the top of the structure. Cutting or eating part of the marshmallow disqualifies the team.
- **Use as Much or as Little of the Kit**: The team can use as many or as few of the 20 spaghetti sticks, as much or as little of the string or tape.
- **Break up the Spaghetti, String or Tape**: Teams are free to break the spaghetti, cut up the tape and string to create new structures.
- **The Challenge Lasts 25 minutes**: Teams cannot hold on to the structure when the time runs out. Those touching or supporting the structure at the end of the exercise will be disqualified.
- **Ensure Everyone Understands the Rules**: Don’t worry about repeating the rules too many times. Repeat them at least three times. Ask if anyone has any questions before starting.
So...Go!

AND THE

TALLEST STRUCTURE IS...
Lessons from this exercise

- http://www.ted.com/talks/tom_wujec_build_a_tower.html

- Importance of:
  - Prototyping, especially when the domain is not well understood
  - Team Coherence and Facilitation
  - Experience
    - Much easier 2nd time around
    - Especially if study (postmortem) occurs in between