Deliverables: Zero-Feature Release

- **Build process, installation process, code repository, automated testing framework, bug tracking system**
  - Maybe no tests yet and no tickets in the bug tracking system
- **Installation package**
  - Includes all of the items below
- **Demo of one-step build and component communication**
  - Checks out all sources from repository, compiles and builds binaries, packages them along with all existing documentation and automated tests, and places the result on a known site ready for downloading
  - Shows that your main components identified in the design can successfully communicate (be integrated) with each other
- **Latest specification, design, and test plan documents**
  - Keep them short! Consider what is / isn't important for customers / devs.
- **Up-to-date schedule**
  - Includes what has been done and what remains to be done
- **Release notes**
  - Detailed instructions on how to run the demo
  - Known issues with prioritization

Questions to consider: Who is your audience – developers or end users? What do they expect? What defines success for them?

Deliverables (tentative list): Beta Release

- **Installation package**
- **Application sources and binaries**
  - One-step build for all sources
- **Latest spec & design documents**
  - Keep it short! Consider what is/isn't important for customers/devs.
- **Release notes**
  - Detailed instructions on how to run a (small) demo of your app
  - Known issues with prioritization, expressed in a bug tracking system
- **Up-to-date test plan**
- **Automated tests (unit and acceptance)**
- **Up-to-date schedule**
  - Including what has been done and what remains to be done

Questions to consider: Who is your audience – customers or developers? What do they expect from this release? What defines success for them?

Deliverables (tentative list): Final Release

- **Installation packages**
  - Including all of the items below
- **Application sources and binaries**
  - Separate distributions (installation packages) for customers and developers
  - One-step build – from compiling all sources to creating installation packages
- **User & technical documentation (separate)**
  - User doc: What does my mom need to know (and do) to run this product?
  - Technical doc: What does a support team need to know to work on version 2?
- **Release notes**
  - Known issues with associated severities & priorities
    - Include a link to your bug tracking system's tasks/tickets that reflect those issues
  - Specify where your current code repository is
  - Instructions on running the installer and your app are moved to the user doc.
- **Latest test plan**
- **Automated tests (unit and acceptance)**
  - Test coverage would be a very welcome addition.
- **Up-to-date schedule**
  - Things that have been accomplished (of those that were planned)
  - Features (of those initially planned) that are now pushed to version 2 or abandoned
    - How much would each such feature cost (in terms of dev effort)?

Questions to consider: Who is your audience – customers or developers? What do they expect from this release? What defines success for them?

Beware: Frequent Mistakes Made by Students in Previous SE Classes

Communication-related:

- Failing to submit (for the preliminary releases and even for the final release!) key required components
  - Missing documentation, tests, etc.
- Not having a backup person who knows how to put together deliverables and submit
- Submitting code without clear instructions about how to run it if one starts from scratch
  - Most customers aren't as tech-savvy as you are!
  - Customers/users aren't intimately familiar with your project and your way of doing things.

Lecture 10: Core Principles and Best Practices for Software Design (Part III)

"Treat design as a wicked, sloppy, heuristic process. Don't settle for the first design that occurs to you. Collaborate. Strive for simplicity. Prototype when you need to. Iterate, iterate, and iterate again. You'll be happy with your designs."

-- Steve McConnell, *Code Complete* (*2nd ed.*), Ch. 5

Outline

- Time-tested software design principles
  - Principle of Loose/Weak Coupling
  - Principle of Strong Cohesion / Single Responsibility
  - Open-Closed Principle
  - More coming today, with examples…

Resources

- "*Code Complete*", 2nd ed., by Steve McConnell
  - Ch. 5: http://www.cc2e.com/docs/Chapter5-Design.pdf
- "*The Pragmatic Programmer*", by Andrew Hunt and David Thomas
  - Ch. 2 (sections 7, 8, 9)
  - Ch. 5 (sections 26, 28, 29)
- "*Agile Software Development – Principles, Patterns and Practices*", by Robert C. Martin
  - See handout
- "*Design Patterns Explained*", by Alan Shalloway and James Trott

Principles for Good Design: Interface Segregation Principle

- "Clients should not be forced to depend on methods that they do not use."
- Example: Dogs jump but don't sing. J

Principles for Good Design: Dependency Inversion Principle (Example)

```
#define TEMP_GAUGE 0x86
#define FURNACE 0x87
#define ENGAGE 1
#define DISENGAGE 0

void Regulate(
  double minTemp, double maxTemp)
{
  for(;;) {
    while (read(TEMP_GAUGE)>minTemp)
      wait(1);
    set(FURNACE, ENGAGE);
    while (read(TEMP_GAUGE)<maxTemp)
      wait(1);
    set(FURNACE, ENGAGE);
  }
}
```

```
void Regulate(
    Thermometer t, Heater h,
    double minTemp,
    double maxTemp)
{
  for(;;) {
    while (t.Read() > minTemp)
      wait(1);
    h.Engage();
    while (t.Read() < maxTemp)
      wait(1);
    h.Disengage();
  }
}
```


Principles for Good Design: Dependency Inversion Principle

- (A) "High-level modules should not depend on low-level modules. Both should depend on abstractions."
- (B) "Abstractions should not depend on details. Details should depend on abstractions."
- Example: Separation of policy and mechanism
- Question: Is this principle an argument for structuring systems using layers?

Principles for Good Design: Liskov Substitution Principle

- "Subtypes must be substitutable for their base types."
- This is different from saying that there must be an IS-A relationship between the two types.
  - Example: Is *Square* always substitutable for *Rectangle*?

Symptoms of Bad Designs: Process Smells

Flawed or risky design processes:

- "Design by committee"
  - Everyone on the committee puts in their favorite features into the "soup." What is the result?
  - Moral: The design must be owned and managed (and fended for) by someone.
- Not having several design options to choose from
- Not iterating over designs
- Other examples?

Symptoms of Bad Designs: Product Smells

Source: "Agile Software Development: Principles, Patterns, and Practices", by Robert Martin

Design product smells:

- Rigidity – changes force other changes
- Fragility – changes cause system to break in unexpected ways
- Immobility – can't extract and reuse subcomponents
- Viscosity – hard to do the right thing, easy to do the wrong thing
- Needless complexity – excessive infrastructure with no clear benefit
- Needless repetition – repeating structures, but no abstractions
- Opacity – code is hard to read and understand

Additional Principles for Building Software Systems

- Make the common case fast and the uncommon case correct.
  - But do not spend time on optimizing code early on.
  - "*It is easier to optimize correct code than to correct optimized code.*" -- Donald Knuth
- Among the three desirable aspects – *consistency*, *availability*, and *no network partitioning* – you can only have two.
- Fail early, gracefully, and transparently.
- Establish and maintain a clear audit trail.
  - It requires little investment upfront but is invaluable for debugging purposes.