Today’s Topics

• How can software projects manage change?

• What is prototyping?

• What is incremental development?
Coping With Change

• **Change is inevitable** in all large software projects:
  – Business changes lead to new (or changed) system requirements.
  – New technologies open up new possibilities.

• **This cost of change** is equal to the **cost of reworking completed work** (re-analyzing requirements, design, recoding) plus the **cost of implementing new functionality**.
Reducing The Cost Of Rework

1. Change Avoidance
   - The software development process includes activities to anticipate possible changes before significant rework is required.
   - Example: develop a Prototype system to show a key (uncertain?) features to customers.

2. Change Tolerance
   - The software development process can be designed to accommodate changes at lower cost.
   - Usually through Incremental Development.
   - Changes may be in a future increment (no rework), or may have to alter part of the existing system.
Throwaway Software Prototyping

- **Prototypes** are a test implementation of the system. Use them to try out different options.
- "Throw-Away" Code
  - **Not a basis** for the system.
  - Prototypes could ignore things like **code quality**, **error-handling**, or **testability**.
  - Built to **answer a specific question**, not to see if the whole system will work.
Software Prototyping

- A prototype can be used in:
  - Requirements engineering to help with requirements elicitation and validation.
  - Design processes to explore options. For example, a paper prototype of the UI.

Prototyping Process:

Define Objective → Prototype → Evaluate
Benefits of Prototyping

1. Improved **system usability**.
2. A **closer match** to users’ **real needs**.
3. Improved **design quality**.
4. Improved **maintainability**.
5. Reduced **development effort**.

Example UI prototyping tool: Balsamiq

Image credit: [https://blog.balsamiq.com/3-1/](https://blog.balsamiq.com/3-1/)
Prototype Development

• Prototypes leave out some functionality.
  – Focus on poorly understood areas of the product;
  – Error checking and recovery may be omitted;

• Focus on functional rather than non-functional requirements.

• Prototypes should be discarded after use. They are deliberately not a good basis for a production system:
  – Very hard to tune it to meet non-functional requirements.
  – Normally undocumented;
  – Degraded structure from rapid change (no refactoring)
  – Likely below software quality standards.
Incremental Delivery

- Development and delivery are broken down into **Increments**
  - Each increment delivers some required functionality.
- **User requirements** are prioritized, as they’re the highest impact once delivery begins.
  - Highest priority ones included in early increments.
- Once the development of an increment is started, the **requirements are frozen**.
  - Requirements for later increments continue to evolve.
Incremental Development and Delivery

• **Incremental Development**
  – **Develop** the system in *increments*.
  – Customer evaluates increment before proceeding to development of next increment.
  – Normal approach used in *Agile methods*.

• **Incremental Delivery**
  – **Deploy** an increment for use by *end-users*.
  – More realistic evaluation because of practical use.
  – Difficult to implement for replacement systems as increments have less functionality than old system.
Incremental Delivery

Start

Develop the Increment

Plan the Increment

Validate the Increment

Deploy the Increment

Final system delivered
Incremental Delivery Advantages

• New functionality delivered with each increment so system functionality is available earlier.
• Early increments act like a prototype to help elicit requirements for later increments.
• Lower risk of overall project failure.
• Highest priority requirements implemented first and receive the most testing.
Incremental Delivery Problems

• **Common Functionality**
  – Most systems require a set of *basic facilities* that are used by different parts of the system.
  – **Hard to identify common facilities** because requirements are not defined in detail until an increment is to be implemented.

• **Contracts**
  – *Specification developed iteratively* with the software.
  – *Complete system specification* can be needed as part of the system *development contract*. 
Recap – Learning To Cope

• **Processes** should cope with **change**.

• **Change Avoidance**
  – Throwaway prototyping helps avoid poor decisions on requirements and design.

• **Change Tolerance**
  – Iterative development and delivery allows changes without disrupting whole system.