Engineering 6806 ECE Design Project

Incremental Development

Dennis Peters & Siu O'Young

Fall 2007

A "Traditional" Development Process

1. Requirements Analysis
2. High level design
   - Divide solution into functional blocks.
   - Define interfaces between the blocks.
3. Split the group into sub-teams to work on each block. Each group:
   1. Design the details for the block (a.k.a., low level design).
   2. Implement the block.
   3. Test the block.
4. Integration (put it all together).
5. Test.

When does the Traditional Process work well?

- Large teams — Blocks can be reasonably worked on in parallel.
- Few unknowns
  - Requirements are stable.
  - Technology is well known.
  - The team has solved similar problems before.
- Implementation and design are clearly distinct tasks.
- Implementation is a significant portion of development.
- Blocks can be effectively tested independently.

Few, if any, of these apply to this project, so how should we proceed?

Incremental Development

- The development period is divided into a sequence of **short** (2–4 week) *increments*.
- Each increment follows the full life-cycle:
  - requirements analysis,
  - design,
  - implementation
  - testing.
- The endpoint of each increment is a **functioning system**.
- You can objectively determine if you've met the goals of the increment by the **behaviour** of the system.
- Usually all behaviour of previous increments is also exhibited in later increments — there is progress towards the final goal (i.e., all behaviours).

How to plan for and execute incremental development

- Identify a set of distinct system behaviours (e.g., driving straight ahead, stopping to avoid obstacles, turning, reporting distance travelled, . . .).
- Assign each system behaviour to a particular increment.
- Candidates for early increments:
  - Highest priority (from the customer's point of view).
  - Highest risk (i.e., least well understood).
- Focus on what is needed for the current increment only (if it's not needed in this increment, *then don't do it*).
- Constantly track progress
  - If it looks like you won't meet a target date, drop behaviour rather than moving the date.

When and Why to use Incremental Development

- Requirements are not well understood.
  - Developers get a better understanding by solving parts of the problem.
  - Users can give feedback on early increments.
- Technological uncertainty.
  - Early increments used to test if/how well technology is working to solve the problem.
- Schedule uncertainty.
  - Lack of experience makes it difficult to know how long it will take to do some parts.
  - Constant reflection makes adjusting the schedule easy.

When and Why (cont'd)

- Design/interface uncertainty.
  - Constant integration ensures that interfaces are understood.
- Important to deliver something quickly.
  - Time to market can be critical in some industries.
  - Gives management/customers confidence that progress is happening.
- Small team.
  - Parallel development of components isn't feasible.
  - Directs energy, avoids "thrashing."

Common mistakes

- An actual progress report from a previous student: "Increment 1 is 80% complete, increment 2 is 50% complete and increment 3 is 20% complete."
  - This student clearly missed the point.
  - Each increment should be finished before the next is started.
- Failure to refactor.
  - At each increment consider the design and how it can evolve into the next increment.
  - If it isn't right for the next increment throw it out **now** before it becomes an albatross around your neck.

Mistakes (cont'd)

- Focusing on components rather than behaviour.
  - Increments are defined by the behaviour they deliver, not the development that goes into them.
  - Keep your eyes on the bottom line (behaviour), not the components.
- Over design.
  - There is a strong tendency to want to make an "elegant design" (e.g., more flexible, configurable . . .).
  - The best designs are the simple designs (KISS = Keep It Simple, Stupid).
- Failure to make a plan and communicate it.
  - Write down what behaviour is in each increment.
  - Make sure everybody has the plan.
  - Keep the plan current.

Documenting the Plan

- An *increment* is defined by a set of *behaviour* that will be exhibited on a given *date*.
  - Set the end date.
  - Describe the observable system behaviour that will be demonstrated (i.e., a test case that can be used to see that the increment is complete).
  - Determine what tasks are to be completed to achieve this behaviour.
- A *task* is a block of work that can be assigned to one or more people to complete.
  - Set start and end dates.
  - Assign it to one or more people.
  - Describe the work that is to be done.
  - Define an end point — how can we *objectively* determine if the task is complete?

Some Advice

- A task end point need not be in terms of observable behaviour at the system level.
- Tasks can include stuff that isn't part of the running system (e.g., documentation, testing etc.)
- For both tasks and increments try to set *objective* and *precise* definitions.

Vague	The judge communications software is complete.
Precise	The judge communications software implements the protocol as described in the provided Judging Software Interface document and has been tested with the provided judge software through a complete rally course of at least four waypoints.