Software Engineering

CS305, Autumn 2020

Week 9

Class Progress…

- Two weeks ago
  - Design patterns, Design principles, Rational Unified Process (RUP)
- Last class
  - Overview of RUP

Class Progress…

- This week:
  - RUP phases, Software Construction, Testing

- Some Topics for Next Assignment (due two weeks from now):
  - Facebook API
  - Ajax, HTML5
  - Google Web Toolkit
  - Google Docs API
  - Webservices: REST API
  - Junit
  - Maven
  - Selenium
  - Android SDK
  - Dart

RUP Contd..

Inception Phase

- **What:** from idea to vision of the end product
  - Identify scope
  - Worth doing? What are the success criteria? Risks? Resources needed?
- **Output:**
  1. Vision document
  2. Simplified initial use-case (who are the actors?)
  3. Draft architecture (what could be the architecture?)
  4. Project Plan and Risks document (How much will it cost? What is the plan?)
  5. Prototype *(optional)*

Inception Phase – Milestone #1

- **When should we stop and consequences:**
  - Stakeholder agrees on scope, cost/schedule estimates
  - Fidelity of the use cases
  - Credibility of cost/schedule estimates, priorities, risks, development process
  - Depth and breadth of prototype if produced

  Project may be cancelled or considerably re-thought if it fails to pass this milestone

Elaboration Phase

- **What:** four main activities
  - Analyze problem domain (for better understanding)
  - Establish architectural foundation
  - Eliminate highest risk elements (identify most critical use cases)
  - Refine project plan and estimates
- **Output:**
  - Almost complete use-case model (all actors and use cases identified and most use cases described)
  - Supplementary requirements (functional and non-functional)
    - All those requirements not directly associated with use cases
  - Software Architecture

Elaboration Phase

- Lower-level design model, test cases, *executable prototype*
- Preliminary user manual
- Revised project plan and risk assessment doc

- **When should we stop and consequences (milestone #2):**
  - Are vision and architecture stable?
  - Are major risks addressed and resolved in the prototype?
  - Thoroughness of the plan:
    - Sufficiently accurate / detailed?
    - Stakeholders agree that the vision can be achieved with the current plan?
    - Actual vs. Planned resource expenditure acceptable?

Project may be cancelled or considerably re-thought if it fails to pass this milestone

Construction Phase

- **What:** transition from intellectual property (IP) development to Product
  - All features developed
  - All features thoroughly tested
- **Output:**
  - All use cases realized, with traceability information – which part of design and code realize a particular use case? Which test cases derived from which use case?
  - Software Product integrated on required platforms
  - Complete system test results
  - User manual

Construction Phase

  - Beta release: design, code, test cases etc.
- **When should we stop and consequences (milestone #3):**
  - Product stable enough to be deployed?
  - Are stakeholders ready for the transition into the user community?
  - Are actual vs. planned resource expenditures still acceptable?

Transition to post-Beta release may be postponed if it fails to pass this milestone

Transition Phase

- **What:** mostly about deployment and maintenance
  - Users might report bugs, suggest improvements
  - Training customer service and providing help-line assistance
  - Beginning of a new cycle
- **Output:**
  - Complete project
  - Project in use
  - Lessons learnt
    - What should we do in the next cycle?
    - What went well? What did not go well?
    - Plan for the next release

Transition Phase

- **When should we stop and consequences (milestone #4):**
  - Users satisfied?
  - Are actual vs. planned resource expenditures still acceptable?

consequences?

Software Construction

Software Design  Code

and much more..

Software Construction

- Also known as Coding or Implementation Phase
- Overlaps with Design (previous) and Testing (next) phases
- Is a large part of SDLC activity
  - Not just coding. *Involves substantial judgement and creativity*
  - Involves Reviews, Programming, Refactoring, and Unit Testing
- **Output:** source code
  - *Should not happen that the source code is the only accurate description of software*

Reviews / Inspections

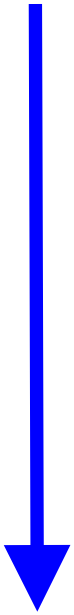
- Introduced by Fagan in 1976
- **Scope:** applicable to code, design documents, documentation, bug fixes, test plans
- **Goal:**
  - Fault detection (not correction)
    - Detect bugs and defects
  - Check conformance to standards
  - *Not to educate staff*
- **Output:**
  - Inspection defects list (with priority and severity of defects mentioned), outcome.

Reviews, Inspections, Walkthroughs

- **Review:** examined by individuals other than the one who produced it
- **Walkthrough:** author describes the artifact and requests for comments from participants
- **Inspections:** more disciplined practice for detecting defects. More like an audit.

Review Methods and Effectiveness

- Review Methods:
  - Peer group review
  - Informal presentation
  - Formal presentation
  - Walk through – More formal. But not as formal and comprehensive as an inspection. Well-Structured.
  - Inspection – formal audit.



Most effective

Reviews: Roles

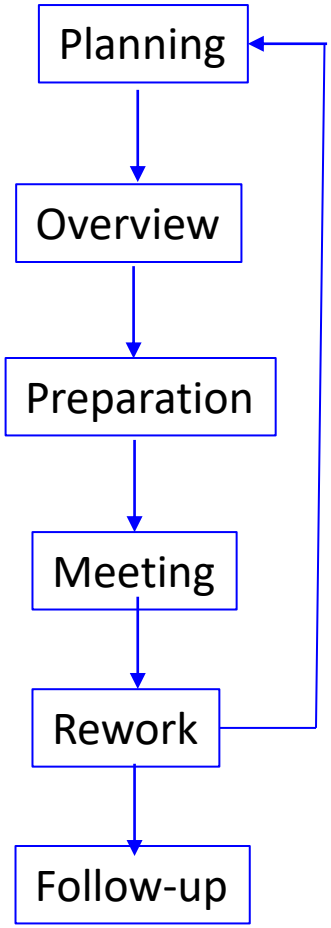
- *Author*: answers questions, clarifies
- *Reader*: reads and paraphrases as needed
- *Recorder*: records defects and issues an *inspection defect list*
- *Inspectors*: analyze and detect defects in the review artifact

Reviews: Roles (contd..)

- *Moderator*:
  - Selects participants
  - Evaluates preparedness of inspection team prior to commencement (aborts inspection if team is not prepared)
  - Verifies that entry criteria is met (actual completion of work to be inspected)
  - Keeps meetings on track and arbitrates differences
  - Is responsible for completion of follow-up

Inspection Process

1. Planning – prepare materials
2. Overview – Moderator selects and educates participants. ~Time < 5 days.
3. Preparation – participants study the material. ~Time = 2 hours
4. Meeting - Moderator evaluates preparedness, Reader performs a line-by-line walkthrough of the document, Recorder gives written report to moderator within one day of meeting, *Moderator reviews defect list, determines severity and priority, and decides outcome.* ~Time = max(2 hours or 100LOC per hour)
5. Rework – Author confirms and removes defects, team checks if all issues are resolved.
6. Follow-up – moderator verifies and validates rework, publishes results, updates process



Reviews: possible outcomes

- *Accept*
  - Some bugs, left to the author's discretion
- *Conditional accept*
  - A few major bugs, to be fixed by the author and verified by the moderator
- *Re-inspect*
  - Many major bugs, to be fixed by the author; the artifact must be re-inspected
- *Reject product outright*

Conducting the Review - Guidelines

- Raise issues don’t resolve them
- Review the product and not the producer
- Keep your tone mild by asking questions instead of making accusations
- Stick to the review agenda

Guidelines (I)

- Inspections should not be used for personnel evaluations
- Managers should not attend unless they have participated technically in the production of the document being inspected
- Hold a separate overview meeting to educate staff
- Spread out inspections over time

Guidelines (II)

- Do not allow additional participants (observers)
- Avoid use of “you” and discussion that might raise defensiveness
- Author should not be moderator, recorder, or reader
- Each type of inspection should have its own checklist

Guidelines (III)

- Avoid problem solving during the inspection meeting
- Cost of inspection should be between 10-20% of development effort
- Avoid discussions of style (unless it is in the checklist)

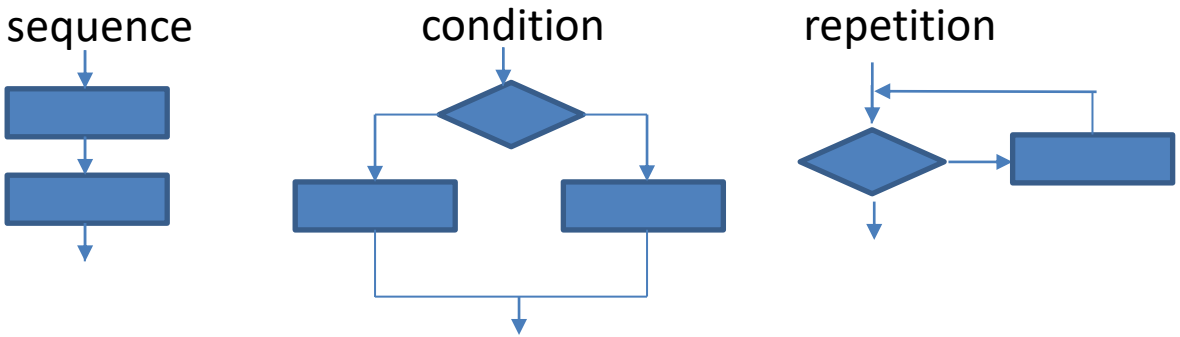
Coding

Coding

- Could involve:
  - Writing source code / programming in a chosen language
  - Automatic generation of source code using a design representation of the component to be constructed
  - Automatic generation of executable code using a *fourth-generation* language – program generating language

Programming Paradigms

- Unstructured Programming
  - Writing a sequence of commands or statements that access 'Global' data. E.g. Assembly lang. programming.
- Structured Programming (sometimes used interchangeably with procedural programming)
  - Dijkstra's advice on using simple logical constructs of:



  - Focus on writing 'modular' programs. Have single-entry and single-exit for a procedure / function (control construct). E.g. C, Assembly lang. programming

Programming Paradigms

- Object Oriented Programming
  - Modeling real-world objects. Data is the centerpiece. Combine data and functions, allow code reuse, incremental dev. maintainability, modularity. *(more in Week3 lectures).* E.g. C++, Java
- Functional Programming
  - Focus on what to do and not how to do. Don't create state that is changeable. E.g. Lisp, Racket
- Concurrent Programming
  - Focus on concurrent execution of a sequence of statements.
  - Parallel programming is a type.
  - E.g. Threads programming (Java threads), Open MP, MPI, CUDA-C.

Human understanding is facilitated by linear sequence of logical statements

Coding Principles

- Ensure that the problem is well-understood before coding (i.e. design is clear, programming language is clear)
- Follow Dijkstra's advice and create modular code that is highly cohesive and loosely coupled
- Select data structures that meet the design objectives
- Create readable code (have indentation, blank lines, and comments)
- Select meaningful names for variables, functions, and follow coding standards and best practices
  - `tmp, temp, data` are "symptoms of programmer laziness".
  - (for GCC) https://gcc.gnu.org/wiki/CppConventions
- Get code reviewed by peers

Code Review – class exercise

- Review the following Fortran code

```
 1 DOUBLE PRECISION FUNCTION SIN(X, E)
 2 C     THIS DECLARATION COMPUTES SIN(X)TO ACCURACY E
 3         DOUBLE PRECISION E,TERM,SUM
 4         REAL X
 5         TERM=X
 6         DO 20 I=3,100,2
 7         TERM=TERM*X**2/(I*(I-1))
 8         IF(TERM.LT.E)GO TO 30
 9         SUM=SUM+(-1**(I/2))*TERM
10      20 CONTINUE
11      30 SIN=SUM
12         RETURN
13         END
```

Code Review – class exercise

- Review the following Fortran code

```
1  DOUBLE PRECISION FUNCTION SIN(X, E)
2  C     THIS DECLARATION COMPUTES SIN(X)TO ACCURACY E
3         DOUBLE PRECISION E,TERM,SUM
4         REAL X
5         TERM=X
6         DO 20 I=3,100,2
7         TERM=TERM*X**2/(I*(I-1))
8         IF(TERM.LT.E)GO TO 30
9         SUM=SUM+(-1**(I/2))*TERM
10     20 CONTINUE
11     30 SIN=SUM
12        RETURN
13        END
```

- `C` is comment to end of line
- The `CONTINUE` statement is often used as a place to hang a statement label, usually it is the end of a `DO` loop. If the `CONTINUE` statement is used as the terminal statement of a `DO` loop, the next statement executed depends on the `DO` loop exit condition.
- `.LT.` is less than
- `**` is exponentiation (has higher priority than *)

- `DO` *`label var = expr1, expr2, expr3`*
  `statements`
  `Label CONTINUE`

`var` is the loop variable (often called the *loop index*) which must be integer. *`expr1`* specifies the initial value of *`var`*, *`expr2`* is the terminating bound, and *`expr3`* is the increment (step).

Code Inspection Checklist (excerpt)

1. Data (DA)

- Is each variable correctly typed?
- Is each variable initialized before use?
- Is the initialization appropriate for the type?
- Can global variables be made local?
- Are buffer overflows checked?
- Is dynamically allocated memory freed?

2. Interface (IF)

- Are appropriate values returned from functions?
- Do function calls have correct parameter types/values?
- Are return values tested?

3. Functionality (FN)

- Do loops terminate?
- Do all loops iterate the correct number of times (no off-by-one errors)?
- Is behavior correct if a loop is never entered?
- Is there dead (unreachable) code?
- Do all switch statements have a default case?
- Do all switch arms have break statements? If not, is the ``fall through'' correct?

4. Input/Output (IO)

- Are files opened before use?
- Are files closed after use?
- Are error conditions checked?

5. Other (OT)

- Any defect discovered that does not fall into one of the above categories.

Slide courtesy: Alex Orso, CS3300

Further Reading

- Code Reviews:

http://web.mit.edu/6.005/www/fa16/classes/04-code-review/

Misc: "The Mess We're In" – Joe Armstrong
https://youtu.be/lKXe3HUG2l4

Pay special attention to the slide on "7 deadly sins" at around 8:00