CITS4401 Software Requirements and Design
Week 11 – Non-OO design

Lecturer: Arran Stewart

Some Other Design Methods

- So far we have focused on object-oriented design
- However object oriented design may NOT be suitable for all types of systems
- All design methods involve a hierarchy of decompositions which partition the design into subsystems or components
- This lecture outlines some non-object oriented methods for doing this decomposition

Data-Oriented Design (DOD)

- A school of thought: “identification of the inherent data structure can be used to derive the structure of a program”
- Logical Construction of Programs (LCP)
    - Developed by Warnier (’74)
    - Draw upon relationship between data structure and procedure structure

Data-Oriented Design (DOD)

- Data Oriented Design (DOD) may be successfully applied in applications that have well-defined, hierarchical structure of information
- LCP is same as DOD
- (See Pressman, 3rd Ed, pp. 429-432)

Data-Oriented Design (DOD)

As SW design methods have evolved, one school of thought holds that “The identification of the inherent data structure can be used to derive the structure of a program”.

Logical construction of programs (LCP) by Warnier is a text (avail UWA Science Library 006.32 1974-34). The author proposed to draw upon the underlying data structure and the relationship between data structure and procedure structure to construct programs. This is exactly the concept of DOD (Data-Oriented Design). Data structure-oriented design transforms a representation of data structure into a representation of software, e.g. recursive binary tree data structure => tree parsing via recursion.

Data structure is considered to be the core element in program construction. It is a core unit offered under Computer Science program in universities throughout the world. Warnier also suggested the use of flowchart, which has been widely used since.

Event-Oriented Design

- Each subsystem consists of components that handle similar type of events
- Subsystems are event-handlers
- Examples: editors; rule-based systems in AI; most real-time systems are event driven
- Disadvantage:
    - Subsystems don't know if or when events will be handled

Data-oriented vs Event-oriented design

Data-oriented design

- start with external data structures and continue by adding more detailed data structures

Event-oriented design

- what events are possible for this system?
- what response is required for each event?
- how does each event change the system state?

“Structured” design

- Try to decompose each subsystem into modules
- Two main strategies:
  - Object-oriented decomposition – subsystem decomposed into a set of communicating objects
  - Function-oriented pipelining – subsystem decomposed into functional modules

Structured design

modular / functional design

- each subsystem captures one of the functions of the system

outside-in / top-down design

- start with black boxes and their inputs and outputs
- then divide each box into internal input-output boxes

Formal Methods and Refinement

- A final (extreme) alternative to the design process is the use of formal methods.
- Software specifications are meticulously transformed into mathematical statements.
- Then a process of refinement is used to derive provably working code from the mathematical specification.

Formal Methods cont.

- Formal methods are used for safety critical applications.
- Z (“Zed”) is a specification language that can map specifications in first-order logic into executable pseudo-code.
- The requirements stage of development is very expensive, but the design, implementation and testing can be a lot cheaper than other methodologies.

Agile Methods

- In the 80s and early 90s, a wide spread view that the best way to achieve better software was through careful project planning, formalized quality assurance. . . - Suitable for large critical projects shared by large teams (mostly located at different locations) - For medium to small sized projects, the overhead is too large
- Agile methods rely on an iterative approach to software specifications, development and delivery. They are designed for business applications where the system requirements usually change rapidly.

Principles of Agile Methods

All agile methods share the following set of principles:

Customer involvement

- Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and evaluate the iterations of the system.

Incremental delivery

- The software is developed in increments with the customer specifying the requirements to be included in each increment.

People, not process

- The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

Principles of Agile Methods (cont.)

Embrace change

- Expect the system requirements to change, so design the system to accommodate these changes

Maintain simplicity

- Focus on simplicity in both the software and the process. Wherever possible, actively work to eliminate complexity from the system.

Advantages of Agile Methods

- Better customer satisfaction by the rapid and continuous delivery of software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- The close interaction between customer and software developer allows continuous attention to technical excellence and good design.
- Software development is able to adapt to changing circumstances. Even late changes in requirements are welcomed.

Disadvantages of Agile Methods

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

Best-known Agile Methods

- Extreme Programming
  - Proposed by Beck (1999, 2000)
  - Commonly abbreviated as XP
- Scrum
  - Proposed by Schwaber and Beedle (2001)
- Crystal Clear
  - Proposed by Cockburn (2001)
- Adaptive Software Development
  - Proposed by Highsmith (2000)
- Feature Driven Development
  - Proposed by Palmer and Felsing (2002)
- Test Driven Development (TDD)
  - Sometimes referred to as test-driven design

Extreme Programming (XP)

- emphasizes on customer satisfaction
- improves software project on 5 essential ways: communication, simplicity, feedback, respect, and courage
- advocates frequent “releases” in short development cycles
- Other elements of XP include:
  - Pair programming
  - Extensive code review
  - Unit testing
  - Avoid programming of features until they are actually needed

Scrum

- Scrum defines a flexible, holistic product development
- It encourages team members to self-organize, to be
- It adopts an empirical approach to requirements

There are 3 core roles in the Scrum framework:

- Product owner (representing the stakeholders)
- Project team
- Scrum master (not a traditional team lead or project manager, but acts as a buffer between the team and any distracting influences)
- Other elements in the framework: sprint (a basic unit of development in Scrum), daily scrum, sprint review and sprint retrospective

Scrum

A sprint (or iteration) is the basic unit of development in scrum.

Daily scrum: A daily scrum in the computing room. This centralized location helps the team start on time. Each day during a sprint, the team holds a daily scrum (or stand-up) with specific guidelines

At the end of a sprint, the team holds two events: the sprint review and the sprint retrospective.

Scrum

At the sprint review, the team:

- Reviews the work that was completed and the planned work that was not completed
- Presents the completed work to the stakeholders (a.k.a. the demo)

At the sprint retrospective, the team: - Reflects on the past sprint - Identifies and agrees on continuous process improvement actions

Crystal

- The Crystal methodology focuses on people, interaction, community, skills, talents, and communications. The software process is consider important, but secondary.
- Cockburn's philosophy translate into a recognition that each team has a different set of talents and skills and therefore each team should use a process uniquely tailored to it.

Crystal Clear

- is a member of the Crystal family of methodology
- is considered an example of an agile
- can be applied to small team (up to 8 developers) co-located working on systems that are not life critical.
- has the following properties:
  - Frequent delivery of usable code to client/users
  - Reflective improvement
  - Face-to-face close communication preferably by being co-located
  - Preferred for development of systems that are not life critical

Adaptive Software Development

- Adaptive Software Development (ASD) is an advance form of Rapid Application Development (RAD). It embodies the principle that developers should continuously adapt the software process to the work at hand.
- ASD replaces the traditional waterfall cycle with a repeating series of speculate, collaborate, and learn cycles.

Adaptive Software Development

- Speculation – during speculation, the project is initiated and adaptive cycle planning is conducted, the set of release cycles is defined.
- Collaboration – refers to the efforts for balancing the work based on the predictive parts of the environment and the uncertain surrounding mix of changes.
- Learning – during the learning cycle, knowledge is gathered by making small mistakes based on false assumptions and correcting those mistakes. This leads to greater experience and eventually mastery in the problem domain.

Feature Driven Development

- FDD blends a number of industry-recognized best practices together.
- Its main purpose is to deliver tangible, working software repeatedly in a timely manner.
- FDD is a model-driven short-iteration process that consists of five basic activities:

FDD

- Develop overall model – high-level walkthrough of the scope of the system; creation of detailed domain model.
- Build feature list - The knowledge gathered during the initial modelling above is used to identify a list of features (a feature is a small client-valued function for the system).
- Plan by the feature – produce the development plan from the feature list; assign ownership of feature sets to programmers
- Design by feature – produce a design package for each feature; a chief programmer develops a selected set of features within 2 weeks.
- Build by feature - After a unit test and a successful code inspection, the completed feature is promoted to the main build.

Test Driven Development

Test driven development is an Agile method

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan
- TDD maps the requirements directly into testing code. The source code is then written specifically to pass these tests (and only to pass these tests)
    - This process is repeated incrementally until the product passes all the tests, and thus meets the requirements.

Also referred to as “Test Driven Design”

Test Driven Development

“TDD completely turns traditional development around. When you first go to implement a new feature, the first question that you ask is whether the existing design is the best design possible that enables you to implement that functionality. If so, you proceed via a Test First Development (TFD) approach. If not, you refactor it locally to change the portion of the design affected by the new feature, enabling you to add that feature as easily as possible.” Scott Ambler

page24

References

- Sommerville, Software Engineering, 7th ed., Addison-Wesley 2004
  - Section 17.1 “Agile Methods”
  - Section 17.2 “Extreme Programming”
- Pressman, 3rd Ed,
  - pp. 429-432