Large Scale Systems Design
G52LSS

Lecture 3 – Rapid and Agile Development

- Rapid Application Development
- Prototyping
- CASE Tools
- Agile Development
- Extreme Programming

Learning outcomes: describe main features of methods for RAD and Agile development; understand pros and cons of prototyping approaches; understand purpose of CASE tools in SDLC; appreciate the features, pros and cons of XP.

Rapid Application Development

RAD is an object-oriented approach to systems development. Involves a development methodology and software tools (e.g. visual programming languages). Typically implemented in three phases:

1. Requirements Planning
2. Sketch with Users of Design System
3. Build the System
4. Implementation

Prototyping

Prototyping can be defined as an interactive process for systems development in which users and analysts are in close collaboration for converting requirements to a working system that is continuously revised.

- Prototyping can be used as:
  - Alternative to traditional SDLC
  - Information gathering technique
- Prototyping does not mean quick and unplanned development!
- Prototyping is a complex technique
- Prototyping helps to set priorities and adapt the planning

Users play a key role in prototyping

Prototyping has two main purposes

Guidelines for working with prototypes

- Work in manageable modules
- Modify in successive iterations
- Estimate the costs of building the prototype
- Build the prototype rapidly
- Stress the user interface

Types of prototypes

- Patched-up prototype
- Non-operational prototype
- First-of-a-series prototype
- Selected features prototype
- Advantages of prototypes
  - Potential for changes to the system early in the development
  - Opportunity to stop developing a non-working system
  - Possibility of developing a system that truly addresses user’s requirements and expectations

- Disadvantages of prototypes
  - Prototyping is difficult to manage
  - Risk that users and analysts adopt an incomplete system as complete

Exercise 3.1 Consulting opportunity 9.2 from (Kendall & Kendall, 2000).
World’s Trend is building a web site in which to sell clearance merchandise usually sold through the web and through its catalogue operation. As a newly hired web consultant, Lincoln Cerf finds himself in a very cold, wintry city, fighting its way through several inches of snow to meet with some of the systems team members, Mary Maye, at World’s Trend headquarters.

Mary welcomes Lincoln, saying “at least the weather doesn’t seem to affect our web sales! They’ve brisk no matter what.” Lincoln groans appreciatively at her weak attempt of humour, smiles, and says, “I gather from your email last week that you are trying to determine the type of information that needs to be displayed on your clearance web site.”

Mary replies, “Yes, I am trying to get it organised in the best possible way. Our customers are all so busy. I know photos of all our merchandise can take a long time to appear on the page if a customer is accessing the web via a slower modem from home.” Mary continues by saying, “Lincoln I’m not even that concerned about how to design our clearance site at this time.

As a third member of the World’s Trend web site development group, respond in a brief written report to Lincoln and Mary about whether you should use a prototype to elicit recommendations from potential customers about the proposed web site. What type of prototype is appropriate? Consider each form of prototype and explain why each type would apply or would not apply to this problem.

I am worried though, about how much information we need to include on a page. For example, when items are on clearance, not all colours and sizes are available. Which do you think is better, to include some basic information and let the customer click a button to ask for more information, or to be as complete as possible on one page? If I use the linking method, then I could fit more items in the screen...but it might be too orderly. Customers like the look and feel of a sale in which merchandise is kind of jumbled together.”

Lincoln continues his line of thought, saying “Yeah I wonder how customers want the information organised. Have you actually watched them using the web? I mean, do they look for shoes when they buy a suit? If so, should shoes appear on the suit page or be linked in some way?

Mary comments, “Those are my questions, too. Then I wonder if we should just try this approach for men’s clothes first, before we implement it for women’s clothing. What if men’s and women’s approaches to shopping in the web are different?

CASE Tools
Computer Assisted Software Engineering (CASE) tools:
- Help to automate activities in the SDLC
- Aim to enforce an engineering-type approach to the development of software systems
- Range from simple diagramming tools to very sophisticated programs to document and automate most steps in the SDLC

Benefits of using CASE tools:
- Improve quality
- Help increase productivity
- Improve communication
- Encourage integrated approach
- Improve management
- Help for better systems maintenance

Other benefits of prototyping:
- Faster creation/modification of documentation
- Documents can easily be distributed and reviewed
- Maintain a record of the SDLC in an effective way
- Reduction in the amount of paperwork
- Members of the development team can interact more easily throughout the project
- Communication with the user is improved because changes can be reported, implemented and revised more quickly
- Persuades the use of a SDLC methodology
- Interrelation and continuity between stages in the SDLC are easier to identify and verify
- Quality assurance for consistency and completeness
- Integration with other productivity tools
- The impact of changes can be assessed for the whole system
- An integral assessment helps to develop better maintenance plans
- Crucial for RAD, XP and similar methodologies
Potential drawbacks of CASE tools:
- No standards
- No substitute for human expertise
- Risk of driving development
- Long-term investment
- Long-term benefits

Types of CASE tools
- Scope
  - CASE tools
  - CASE toolkits
  - CASE workbench
- Focus
  - Upper CASE
  - Lower CASE
  - Intermediate CASE

Upper CASE tools
- Help to create and modify the system design
- Information about the analysis and design is stored in the CASE repository
- Analysis reports show incomplete parts and errors in the system design. For example, balance between process and data models
- Support modelling of how the systems fits in the organisation

Lower CASE tools
- Generate source code (25-99%) and reduce need for systems programming
- Once mastered, promote the re-use of existing documentation and components
- Time for maintenance is reduced because test and debug are eliminated
- Easy for migrating systems across different platforms
- Help to modify third-party software at a low cost

Examples of CASE tools
Diagramming – for representing processes, data and control structures graphically
CASE repository – holds information required to create, modify and evolve the system
Form and report generators – automate generation of forms and reports to aid prototyping
Code generators – automate generation of source code from diagrams and forms

Examples of CASE tools (cont.)
Project management – aid in the planning, tracking, controlling and reporting of project management
Document generator – create standard reports based upon the contents of the CASE repository
CASE analysis tools – help to identify problems of inconsistency, redundancy, and omissions (more likely in analysis and design)

Agile Development
Real-world software development is likely to be conducted in volatile environments, as organisations adapt to changes in technology, markets and social conditions.

A number of Agile Development techniques have been proposed for the development of complex and dynamic software systems in today's world:
- Extreme Programming (XP)
- Crystal
- Scrum
- Rational unified process (RUP), and others...
Augustine et al. (2005) propose a number of practices to be followed in Agile Project Development:
- Organic teams of seven to nine members
- Guiding vision
- Simple rules
- Free and open access to information
- Light touch management style
- Adaptive leadership

---

**Extreme Programming**

- XP by Kent Beck and Ward Cunningham in late 1990’s
- Software development approach that pushes good software development practices to the extreme
- Define an overall system **plan quickly**
- Develop and **release software quickly**
- Promotes to **continuously revise the software** to incorporate additional features
- Extreme programming: **Do it quickly! Do it Well!**
- XP is based on defined **values, principles, resources, activities and core practices**

---

**The Phases of XP**

*Figure 1.1P The phases of the XP development process shows that frequent iterations are essential to successful system development.*

---

**(Good) Lessons learned from XP**

- Short releases allow systems to evolve
- Pair programming enhances overall quality
- Onsite customers are mutually beneficial to the business and the XP team
- The 40-hour week improves effectiveness
- Balanced resources and activities support project goals
- Extreme programming values are crucial to success

---

**Criticism of XP**

XP is not regarded as the panacea by everyone, some criticisms:
- XP is an undisciplined approach
- Pair programming does not always work
- Testing everything is non-practical
- Simple design hinders scalability and robustness
- XP works only if all guidelines and practices are followed
- It can deteriorate to low disciplined practices (e.g. minimal documentation) without ensuring high disciplined practices (e.g. unit testing, pair programming, collective ownership, constant refactoring)
- XP is aimed at customers who do not know what they want. But systems analysts are meant to help customers to identify requirements!
Criticism of XP (cont.)
- With XP it is very difficult to look ahead and plan future developments
- The cost of continuous change (particularly programming time) is not taken seriously into account
- XP appeals to young, inexperienced programmers that want to avoid documentation
- XP ‘hopes’ that simplicity will be what the customer wants
- Feedback from customer to programmers is not practical and it is likely to unnecessarily extend the development process
- Real courage is needed to just throwaway code that is out of control or does not work by the end of the day. XP is a throwaway approach!
- XP rules are self-referential!

Exercise 3.2 Investigate how other methodologies like SPIRAL, RUP and SCRUM work for SDLC.

Criticism of XP (cont.)
- Why XP rules are self-referential:
  - No detailed written requirements – XP is aimed at “high risk projects with dynamic requirements”, vague and sporadic (dynamic) requirements
  - No big up-front design – little or no time is spent designing the system before coding begins
  - Constant refactoring – constant tweaking and improving of your code creates an unnecessary overhead and it could potentially introduce lots of bugs
  - Unit tests – useful in everyday ending (not just in XP), leave a critical area uncovered: design correctness, unit tests catch certain types of code-level bug, but they do not catch “wrongness” of a design
  - Pair programming – somewhat overrated, programmers would not know what to code without a specification
  - On-site customer representation – risky because customer is bound to be pretty junior, the customer representative “becomes” the formalised requirements spec

Additional Reading
Augustine et al., Agile Project Management: Steering from the Edges, Communications of the ACM, Vol. 48, No. 12, pp. 86-89, 2005
Chapter 6 of (Kendall and Kendall, 2005)
Appendix 2 of (Hoffer et al., 2005)
See the following web sites for more on XP:
http://www.extremeprogramming.org/
http://www.xprogramming.com/
http://www.jre.com/techno/xpfaq.html
http://en.wikipedia.org/wiki/Extreme_programming
See the following web site for a criticism of XP:
http://www.softwarereality.com/ExtremeProgramming.jsp