Software Development Methodologies

Lecturer: Raman Ramsin

Lecture 8

Agile Methodologies: DSDM
Agile Methodologies: Brief History


- The once-common perception that agile methodologies are nothing but controlled code-&-fix approaches, with little or no sign of a clear-cut process, is only true of a small – albeit influential – minority.

- Essentially based on practices of program design, coding and testing that are believed to enhance software development flexibility and productivity.

- Most agile methodologies incorporate explicit processes, although striving to keep them as lightweight as possible.
Agile Methodologies: Evolution Map

[Abrahamsson et al. 2003]
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
Agile Methodologies: Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

- Business people and developers must work together daily throughout the project.

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Agile Methodologies: Principles (Contd.)

- Working software is the primary measure of progress.

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- Continuous attention to technical excellence and good design enhances agility.

- Simplicity—the art of maximizing the amount of work not done—is essential.

- The best architectures, requirements, and designs emerge from self-organizing teams.

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.
Dynamic Systems Development Method (DSDM)

- First introduced in 1995 by a consortium of UK companies.

- Motivated by an ever-increasing need for a standard, generally-accepted RAD methodology.

- Produced as an iterative-incremental generic framework based on evolutionary prototyping and principles that are nowadays attributed to agile development.

- Starting with 16 UK companies, the consortium now has more than 1000 members, including industry giants such as IBM, Microsoft and Siemens.

- The framework proposed by DSDM is now considered the de facto standard for RAD.
DSDM: Process

1. **Pre-project**: providing the necessary resources for starting the project, along with preliminary planning

2. **Project-proper**, during which the five main phases of the DSDM are applied; the first two sequentially at the start of the project, and the remaining three as interwoven cycles:
   1. **Sequential Phases**: studying the business domain and performing a preliminary analysis of the system:
      1. Feasibility Study
      2. Business Study
   2. **Iterative Phases (The Development Cycle)**: iterative and incremental analysis, design, coding and deployment of the system through evolutionary prototyping:
      1. Functional Model Iteration
      2. Design-and-Build Iteration
      3. Implementation

3. **Post-project**: system maintenance through further iterations of the main phases
DSDM: Process – Sequential Phases

1. **Feasibility Study**: analogous to the classic feasibility analysis, albeit with a special focus on analyzing the suitability of DSDM for the project, and coming up with an outline plan for the subsequent phases.

2. **Business Study**: with the focus on identifying system-relevant processes and information entities in the business domain, defining and prioritizing the high-level requirements of the system, developing the system architecture, and producing a development plan.
DSDM: Process – Iterative Phases

1. **Functional Model Iteration**: with the focus on selecting requirements according to their priority, and performing detailed analysis and modeling of the selected requirements through prototyping.

2. **Design-and-Build Iteration**: with the focus on evolving the prototypes into final deliverable increments of the system.

3. **Implementation**: with the focus on deploying the deliverable increments into the operational environment, and reviewing and validating the system built so far.
DSDM: Process

[Diagram showing the DSDM process]

[Text]

[Reference: DSDM Consortium 2003]
DSDM: A Configurable Process Framework

DSDM is referred to as a *configurable process framework*, since the developers can tailor the process to fit the project in hand:

- The three iterative phases form an outer interwoven cycle; the selection of the number of iterations in each cycle, and the way the iterations should interact, is completely dependent on the project and up to the development team to decide.

- The introduction of multiple development sub-teams working in parallel enables the phases to overlap, adding another configurable dimension to the process.
DSDM: Phases – *Feasibility Study*

1. Acquire high-level knowledge as to the nature of the project, its scope, and the risks and constraints involved.

2. Check whether DSDM is the suitable approach for the project in hand. This is done by applying a list of project and organizational criteria (called the *Suitability Filter*) to the project.

3. Perform the traditional activities of feasibility analysis, paying special attention to technical, schedule, and managerial feasibilities.

4. Develop rough estimates and an overall Outline Plan for the project.
DSDM: Feasibility Study - Suitability Filter

1. The system to be developed should be interactive, with the functionality amply visible at the user interface level (screens, reports and controls).
2. The system should have a clearly defined user group, so that well-informed representatives (called Ambassador Users) can be identified and involved as active participants in the project.
3. The system should not be computationally complex (more business-oriented rather than scientific).
4. The requirements should not be too complex to elicit, delineate, prioritize, or implement individually.
5. There should be no constraint or criticality issue compelling the developers to fully specify the requirements before any coding can commence.
6. If the system is large, it should lend itself to partitioning.
7. The sponsor/senior-management should understand and accept the principles and practices of DSDM.
DSDM: Phases – Business Study

1. Identify the processes and information entities in the business domain that are relevant to the system, as well as the types of users interacting with or affected by it.

2. Define and prioritize the high-level functional and non-functional requirements of the system.

3. The results of the first two tasks are packaged as the Business Area Definition document.

4. Develop the System Architecture Definition, which highlights the architecture of the software solution, and specifies the development and operational platforms.

5. Produce the Prototyping Plan, outlining the order of activities during the iterative phases of the development.
DSDM: *Business Study - Prioritization Rules*

- The requirements are prioritized according to what DSDM calls the 
  **MoSCoW** Rules, which is, in effect, categorizing each of the requirements as one of the following:
  - **Must-Haves**: essential requirements on which the project’s success relies.
  - **Should-Haves**: important requirements, but not essential to the project’s success.
  - **Could-Haves**: requirements that can be excluded from the system functionality without having any serious effect on the project.
  - **Won’t-Haves**: requirements that will not be part of the system functionality in the current project.

- The project must guarantee the implementation of the must-haves and should strive hard to deliver the should-haves. The could-haves will only be realized if time and resources allow their implementation.
A risk analysis is conducted in order to assess the risks involved in developing the requirements. The analysis will be refined during the iterations (based on the prototypes), ultimately resulting in the Development Risk Analysis Report.

Requirements are selected according to their development risk and functional prototypes are iteratively built in order to demonstrate and refine the relevant functionality. The prototypes form the main part of the Functional Model, and will ultimately evolve into the final system.

Non-functional requirements are refined, listed and added to the Functional Model.

If necessary, static models (class diagrams) are used for modeling the structural aspects of the domain area being analyzed. These too are added to the Functional Model.
DSDM: Phases – Design and Build Iteration

1. Prototypes are iteratively refined and gradually evolved into a working software subsystem, ready to be deployed as an increment into the operational environment, and integrated into the system built so far.

2. Testing is performed on a continuous basis, with test cases and relevant results and decisions carefully logged.

3. Intermediate prototypes (called Design Prototypes) are kept on record as design documentation.
DSDM: Phases – Implementation

1. Users and support personnel are trained, and manuals are prepared.
2. The increment is introduced into the operational environment.
3. A comprehensive validation review is performed on the system, results of which are compiled in the *Increment Review Document*. There are four possible outcomes:
   - All requirements planned to be realized have been implemented to the users’ satisfaction, in which case the project is declared as finished.
   - A major area of functionality was discovered during development that had to be abandoned because of time-box constraints, but should be developed; in this case a return to the business study phase is required.
   - An area of functionality had to be left out because of time-box constraints, but should be developed; in this case a return to the functional-model-iteration phase is required.
   - A non-functional requirement had to be ignored because of time-box constraints, yet should be realized; in this case a return to the design-and-build-iteration phase is required.
DSDM: Strengths and Weaknesses

**Strengths**

- Iterative-incremental process
- Based on functional and structural modeling performed on the problem domain and the system
- Early specification of the physical architecture
- Flexible and configurable process (through defining the main development cycle as consisting of interwoven Analyze-Design-Implement cycles)
- Especially suitable for projects with highly volatile requirements, since it is easily adaptable
DSDM: Strengths and Weaknesses

**Strengths (Contd.)**

- Carefully worked-out process
- Seamless development through using prototypes
- Incorporating a *Suitability Filter* to make sure that the project can be carried out with DSDM
- Based on careful planning
- Test-based development
- Active user involvement
- Reversibility of changes
DSDM: Strengths and Weaknesses

- **Strengths** (Contd.)

  - Early and frequent releases
  - Smooth transition from stage to stage
  - Traceability to requirements achieved through constant testing and via the prototype produced
  - Based on prioritization of requirements by categorizing them into specific types
  - Design-based development
DSDM: Strengths and Weaknesses

**Weaknesses**

- Not scalable
- Limited applicability scope: the project should lend itself to RAD through evolutionary prototyping.
- Stringent constraints on time and resources
- Severe model-phobia: text reports are abundant but visual models are avoided unless absolutely essential. The prototype is considered the main model.
- Lack of formalism
Reference