PART 5: Software Development Methods

- Agile development processes
- Model-based development
- Professional issues in software development.

Chapters 1, 3 and 16 of textbook are relevant to this part.
Agile development methods: Scrum, XP

- Agile software development introduced c. 2000 to correct drawbacks of conventional plan-based/heavyweight processes.
- Conventional development relies on gathering and formalising all requirements before starting coding; agile emphasises incremental work on requirements, coding, integration in short cycles. Responsive to changing requirements.
- In conventional development, team isolated from stakeholders: delays in obtaining information/feedback. Agile emphasises close collaboration between team + customer.
- Agile now widely adopted in industry; 2 main approaches: eXtreme Programming (XP) and Scrum.
Agile Development

- Traditional software development processes are plan-based: focus on prescriptive activities + fixed sequence of stages (eg., Analysis, Design, Coding, Testing in Waterfall process).

- Agile processes in contrast aim to have lightweight process: primary goal is to deliver system to customer that meets their needs, in shortest possible time.
Agile development concepts

- Traditional development approaches have become too slow to handle rapid change in markets and technology: by time code is delivered, it is out-of-date.
- Agile development addresses problem by delivering parts of system as working code as soon as possible.
- Short delivery cycles, so developers can adapt system to deliver what customer requires.
Agile development principles

- Responding to change is more important than following a plan.
- Producing working software is more important than comprehensive documentation.
- Individuals + interactions emphasised over processes and tools.
- Customer collaboration emphasised over contract negotiation.

Development cycles are iterative, build small parts of systems, with continuous testing + integration.
Agile development principles

- “Self-selecting teams” – best architectures, requirements, designs emerge from such teams.
- Agile methodologies suitable for smaller organisations and projects.
- Extreme programming (XP) suitable for single projects developed and maintained by single person/small team.
<table>
<thead>
<tr>
<th>Agile development</th>
<th>Plan-based development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small/medium-scale</td>
<td>Large scale (10+ people)</td>
</tr>
<tr>
<td>In-house project, co-located team</td>
<td>Distributed/outsourced team</td>
</tr>
<tr>
<td>Experienced/self-directed developers</td>
<td>Varied experience levels</td>
</tr>
<tr>
<td>Requirements/environment volatile</td>
<td>Requirements fixed in advance</td>
</tr>
<tr>
<td>Close interaction with customer</td>
<td>Distant customer/stakeholders</td>
</tr>
<tr>
<td>Rapid value, high-responsiveness required</td>
<td>High reliability/correctness required</td>
</tr>
</tbody>
</table>
Agile development techniques

- **Sprints**: development work which implements specific user requirements, in short time frame to produce new release. “Deliver working software frequently”. Scrum methodology.

- **Refactoring**: Regularly restructure code to improve it, remove redundancy, etc. (ref: Fowler, “Refactoring: improving the design of existing software”).
Scrum process
**Agile development: Sprints**

- Sprints are regular re-occurring iterations in which project work is completed. Produce deliverables that contribute to overall project.

- Each iteration of system involves set of use cases/work items (‘user stories’ in Scrum) to be implemented. Use cases classified by business value to customer (high, medium, low), and by risk/effort. High-priority high-risk uc’s should be dealt with first.

- Project *velocity*: amount of developer-time available per iteration.

- Taking these into account, can define *release plan*: which uc’s will be delivered by which iteration, by which developers.
Defining a release plan

- Release consists of set of iterations (eg., sprints) and produces a deliverable to client

- If use case \( uc_1 \) depends on \( uc_2 \) via \( \ll extend \), \( \ll include \) or inheritance arrow from \( uc_1 \) to \( uc_2 \), then \( uc_1 \) must be in same or later iteration to \( uc_2 \)

- Iterations also depend on availability of developers with required skills for development of \( uc \)'s in iteration.

- Some use cases may be prioritised over others.

In following example, plan has 3 sprints. Total duration = 28 days, effort = 34 person days (pd).
Release plan example
Defining a release plan

Order tasks in decreasing order of duration. Order staff in increasing order of cost.

Consider tasks/use cases which do not depend on any others first, consider higher priority/higher cost tasks before lower priority/cost ones. Assign each task to lowest cost available staff with necessary skills.

Assign tasks, whose prerequisite tasks have already been assigned, until no further developers can be assigned. This ends one iteration. A new iteration is started with all staff available again.

Duration of an iteration = sum of durations of most time-consuming chain of dependent tasks in iteration.

Duration of release plan = sum of iteration durations.
**Example exam question**

Define release plan for a system consisting of four use cases:

<table>
<thead>
<tr>
<th>Use case</th>
<th>Includes</th>
<th>Effort required</th>
<th>Skills required</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc1</td>
<td>uc2, uc3</td>
<td>10pd</td>
<td>A, B, C</td>
</tr>
<tr>
<td>uc2</td>
<td>uc4</td>
<td>8pd</td>
<td>B, C</td>
</tr>
<tr>
<td>uc3</td>
<td></td>
<td>6pd</td>
<td>B</td>
</tr>
<tr>
<td>uc4</td>
<td></td>
<td>5pd</td>
<td>A, C</td>
</tr>
</tbody>
</table>

Three developers are available: Developer 1 has skills A, B, C, but is more expensive than developer 2 with skills A, B, and developer 3 with skills B, C. For each use case, assign the cheapest developer who is able to develop it, assuming only 1 developer per use case.
Solution is shown in following table.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Use cases</th>
<th>Developers</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>uc4; uc3</td>
<td>developer 1: uc4</td>
<td>6 days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>developer 2 or 3: uc3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uc2</td>
<td>developer 3</td>
<td>8 days</td>
</tr>
<tr>
<td>3</td>
<td>uc1</td>
<td>developer 1</td>
<td>10 days</td>
</tr>
</tbody>
</table>

Total duration = 24 days, effort = 29 pd.
Agile development: Refactoring

- Small changes to rationalise/generalise or improve structure of existing system
- Eg: if discover common attributes in all subclasses of a class: move common attributes up to superclass.
- Refactoring should not change existing functionality: run tests to check this
- Do regular refactoring whenever you identify the need.

These are just refactoring transformations on systems.
Pull-up attribute refactoring
Agile methods: eXtreme Programming (XP)

- XP aims to combine best practices in development projects to focus on agility + code production.

- Based on short development cycles, uses: pair programming; code reviews; frequent integration and testing; close collaboration with customers; daily review meetings.

- Emphasis on coding as key development activity, for communicating ideas between developers and demonstrating ideas to customers.

- Unit testing to check correct implementation of individual features, acceptance testing to check implementation meets customer expectations. Integration testing checks that separately developed features operate correctly together.
Agile methods: XP

- 5 Values: communication, simplicity, feedback, courage, respect.

- 3 Principles: Feedback – via customer interaction, fine-grain testing; Assuming simplicity – code that’s just good enough, small changes; Embracing change.

- 12 Practices: pair programming; planning game; test-driven development; whole team; refactoring; small releases; system metaphor; simple design; continuous integration; collective code ownership; sustainable pace; coding standards.

Appropriate for small teams (up to 12 people).
Agile methods: eXtreme Programming (XP)

Has five phases:

- **Exploration**: determine feasibility, understand requirements, develop exploratory prototypes
- **Planning**: agree date and uc’s for 1st release
- **Iterations**: implement/test uc’s in series of iterations
- **Productionizing**: prepare documentation, training, etc; deploy system
- **Maintenance**: fix and enhance deployed system.

XP recommends that iterations are one to two weeks long.
**Agile methods: Scrum**

Has four phases:

- **Planning**: establish the vision, set expectations, develop exploratory prototypes
- **Staging**: prioritise and plan for 1st iteration, develop exploratory prototypes
- **Development**: implement uc’s in series of sprints, refine iteration plan. Daily Scrum meetings to check progress.
- **Release**: prepare documentation, training, etc; deploy system.

Scrum recommends that iterations (sprints) are 1 week to 1 month long.
**Agile methods: Scrum**

Key events:

- *Sprint Planning*: performed by Scrum team before sprint, team agrees uc’s to be worked on in sprint

- *Daily Scrum*: organises activities of team, review sprint progress + deal with issues. Time limited (eg., 15 mins). Key questions for developers: (i) what did I achieve yesterday? (ii) what will I achieve today? (iii) is there anything blocking me from achieving my work?

- *Sprint Review*: at end of sprint

- *Sprint Retrospective*: after sprint review, before next sprint planning. Analyse achievements of sprint, ideas for improvement of process.

For student projects, daily Scrum can be weekly, etc.
**Agile methods: Scrum**

- **User story**: a generalised task (i.e., use case). Can consist of set of subtasks.

- **Product Backlog**: ordered (in terms of priority) list of stories relevant to project.

- **Sprint Backlog**: ordered list of stories to be completed in a sprint.

Team uses *Scrum Board* showing tasks to do, in progress and completed. *Burndown Chart* shows graph of remaining work against time.
<table>
<thead>
<tr>
<th>Product Backlog</th>
<th>Iteration Backlog</th>
<th>In progress</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>task 9</td>
<td>task 6</td>
<td>task 2</td>
<td>task 1</td>
</tr>
<tr>
<td>task 10</td>
<td>task 7</td>
<td>task 4</td>
<td>task 3</td>
</tr>
<tr>
<td>task 12</td>
<td>task 8</td>
<td>task 5</td>
<td></td>
</tr>
<tr>
<td>task 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>task 14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>task 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scrum board example
Agile methods: Scrum

Scrum defines 3 key roles for software development team members:

- **Product owner**: customer representative in team, role is responsible for liaising between technical staff and stakeholders. Product owner identifies required work items, identifies their priority, and adds to product backlog.

- **Development team**: workers who perform technical work. Team should have all needed skills + be self-organising. Typically between 3 to 9 members in this role.

- **Scrum master**: facilitates Scrum process and events, + self-organisation of the team. Not project manager role, does not have personnel management responsibilities.
Scrum process
Benefits/disadvantages of Agile

9th State of Agile Survey (versionone.com, 2015):

- Main benefits of agile: (i) management of changing priorities; (ii) team productivity; (iii) project visibility.
- Majority of respondents found agile projects mainly successful.
- Scrum main method used, with daily standups, short iterations, prioritised backlogs, iteration planning most common techniques.

Disadvantages of agile:

- Focussed on manual coding – resource intensive.
- Focussed on functional requirements; does not emphasise reuse.
Barriers to adoption of Agile development

- Inability to change organisational culture
- Lack of experience with agile methods
- Company philosophy/values conflict with agile
- Established rigid/Waterfall development approach
- Lack of management support/ management concerns about lack of up-front planning
- External demand to follow conventional staged process
- Lack of access to users/customers.
Other agile development techniques

- *Test driven development (TDD)*: write unit tests first, then successively extend and refine code until it passes all tests.

- *Pair programming*: two developers work at same terminal, observer has role to improve programmers code.

TDD appropriate if no clear understanding of how to solve problem. Pair programming can improve quality, and reduce time, whilst increasing cost.
Pair programming
Model-based Development (MBD)

- Uses UML or other modelling language to develop systems based on models, instead of code.

- MDD (Model-driven development), MDA (Model-driven architecture) make models primary artifact.

- Benefits: models simpler to review + modify than code; code generation can raise productivity substantially; model repositories + libraries can be established for rapid production of systems in a common ‘product line’ family. Reduces need for outsourcing.

- Has been widely adopted in industry, with generally positive results.

- But training + adoption costs, and limited tool support.
Example MBD approach: MDA

- *Model-driven Architecture (MDA)*, an OMG standard (www.omg.org/mda) for MBD using UML.

- MDA based on defining Platform Independent Models (PIMs) + Platform-specific Models (PSMs) of a system.

- Developers should express logical business data + rules in platform-independent manner using UML notations such as class diagrams, then, for required implementation platform(s), map PIMs to PSMs for specific platforms.

- From PSMs, code can be generated automatically.
MDA process
**MBD process using UML-RSDS**

Alternative to MDA.

- Platform-independent specifications defined using UML class diagrams and use cases. Analysed for internal quality + correctness.

- Platform-independent designs synthesised (these use pseudocode notation).

- From designs, executable code in programming language (currently, Java, C, C# or C++) is automatically synthesised.

More restricted in scope than MDA, but more automated.
UML-RSDS software production process
Can Agile and MBD be combined?

- MBD provides modelling, verification + reuse support lacking in Agile
- Agile MBD approach uses models as primary artifact, not code
- But MBD tends to be ‘heavyweight’ process. Multiple models in UML hinder rapid specification change – changing one model may require changes to others.
- Several Agile MBD approaches: MDD-SLAP at Motorola; Simulink process at Volvo Cars; xUML/fUML/Alf from OMG.
**Professional issues in Software Development**

As society becomes more dependent on software, essential to ensure software quality is as high as possible. Software professionals should have:

- Awareness of relevant standards + best practice for software development
- Awareness of legal, social + ethical issues (eg., data protection regulations)
- Awareness of safety and security issues (can your software cause hazards/risks for users?)

In UK, British Computer Society (BCS) sets standards for software practitioners (www.bcs.org). This degree is BCS-accredited.
Summary

• Have introduced concepts of Agile development and Model-based development
• Given examples of their use in industry, and rationales for/against their use.
• Discussed professional issues in software development.
Useful references

- Agile manifesto: www.agilemanifesto.org