9.2. Heavyweight vs. lightweight software development methodologies

As far as the heavyweight approach is concerned, the most common is the **waterfall methodology**. Its modifications include **spiral model** and **v-model**.

The **waterfall model** includes a sequential series of steps which divide **software development life cycle (SDLC)** into the following distinct phases: planning and **requirements gathering**, **system design**, **implementation**, testing, **deployment** and finally software **maintenance**.

The waterfall approach is rather clear and one of its advantages is that the series of fixed **milestones** make it easy to **track progress** of the project. Another advantage of this **linear software development process** is that potential **problems**, which would have been found during development phase, can be **investigated** and **bottomed out** during the **design phase**, before implementation of the solution. The whole process is also well documented so in the end there are no misunderstandings. This type of methodology is usually used in large projects in which the technology, requirements and product definition remain unchanged. Such **inflexibility** and **unresponsiveness** to change seem to become less popular in today's business environment in which companies are constantly changing their software requirements **to keep up the pace** of market development. They do not want to **stay behind their competitors**. Other disadvantages of waterfall approach include **susceptibility to bottlenecks** and **delays** and possible late product delivery as only the completion of final project phase provides a **deliverable**.

The **Rational Unified Process (RUP)** methodology is also frequently used in the software industry. It uses **iterative** and **incremental approach** and is strongly tied to using **object-oriented modelling**. In RUP, a project is divided into a number of iterations and for each iteration the project goes through the following phases, which makes it similar to the spiral model:

- **Inception**: At this stage the development team conducts the analysis of the problem to be solved, assesses **feasibility** of the project and specifies **high-level requirements**. A project vision is created and **business case** is developed.
- **Elaboration**: At this point the project team designs a **baseline architecture** of the system, further **elaborates** on the requirements and conducts **risk analysis**.
- **Construction**: By the end of this phase a **beta-release working system** is made available.
- **Transition**: This is the time to **validate** the product and obtain the acceptance of **intended users** and **stakeholders**.

As far as the lightweight software development methodologies are concerned, the basis for them is the Agile Manifesto which presents key values to the philosophy behind them:

- Individuals and interactions which help to **get rid of** any misunderstandings between the team members, giving rooms for ideas and immediate **feedback**, are more important than processes and tools.
- Having a **working software** given earlier to the user, even if it provides minimal value, is preferred over development of the entire software **in detail** with **comprehensive documentation**. Such documentation may turn out to be a **throwaway task** because of the change of user requirements. It must be pointed out that documentation should not be **overlooked** or **disregarded** but it should exist in its basic form.
- Customer collaboration leading to full understanding of customer needs is more important than contract negotiation which limits the evolution of software project due to **agreed-upon feature set**.
- Responding to change, which should be **embraced** and prepared for as software **evolves**, is more important than following the plan as planned features may be **reprioritized**.

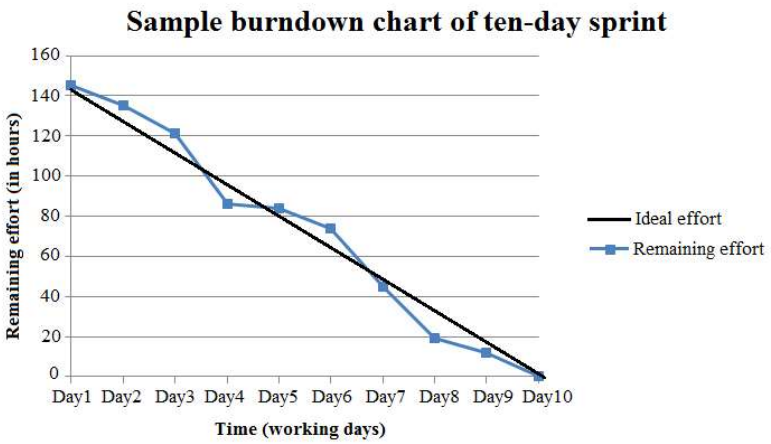
9.3. Agile software development methodologies and frameworks

As it has been mentioned above, agile software methodologies replace **sequential software development approach** and **design upfront** with incremental and iterative one. They allow changes of requirements which are delivered by **self-organizing**, **cross-functional teams** including designers, developers and testers, working on iteration of the product over fixed time periods called **timeboxes**. A working software is delivered at the end of each iteration. The most commonly used agile methodologies or software development frameworks are the following:

- **Scrum**: It is a process **framework** with timeboxed units referred to as **sprints**. Sprints are usually 1 – 4 weeks long. After the **Product Owner** defines a set of requirements in **Product Backlog**, the **Scrum Team** (comprising of the Product Owner, the **Development Team** and the **Scrum Master**) agrees on a set of items to be **addressed** in a sprint and put them in a **Sprint Backlog** during the **Sprint Planning**. The items in Product Backlog are reviewed and revised during **Backlog Refinement**, also referred to as **Backlog Grooming**. They are expressed in the form of **user stories** which are a part of one or more **epics**. The relative effort to implement each user story is estimated in **story points** using for example **Planning Poker**. After that the Development Team identifies tasks which are needed to complete each user story, there is a time for R&D work of developers, followed by demos of the solution and retrospectives. During the sprint each day there are 15-minute time-boxed meetings for the Development Team called **Daily Scrums** to synchronize activities and create a plan for the next 24 hours. Each sprint ends up with a demonstration of potentially **shippable working product** demonstrated to stakeholders which takes place during the **Sprint Review**. After that and prior to the next Sprint Planning, a **Sprint Retrospective** takes place. It is an opportunity for the Scrum Team to inspect the results of its work and talk about the improvements to be **enacted** during

the next Sprint. A measurement of the amount of work the Development Team can complete during a single Sprint (**velocity**) can be graphically presented using a **burn-down chart**. The **forecasted velocity** can be calculated on the basis of **Focus Factor** (velocity/**capacity**). A simple burn-down chart of ten-day sprint is shown in figure 9.1 below.

Figure 9.1.
A simple burn-down chart



- **DSDM Agile Project Framework**: This is a more formal and well-known agile project management framework which enables the organisations to deliver working solutions on time and on budget. According to the framework, there are for example the following roles of people involved in a project:
  - **Business Sponsor**: It is a person responsible for the **Business Case** and project budget who **resolves business issues** and makes financial decisions.
  - **Business Visionary**: It is a person responsible for clear vision of the project who interprets the needs of Business Sponsor and communicates them to the team.
  - **Technical Coordinator**: It is a person responsible for delivery of compatible output, meeting the agreed technical quality standards.
  - **Solution Developer**: It is a person who interprets business requirements and translates them to **deployable solution** which meets the needs of **solution recipients**.
  - **DSDM Coach**: It is an independent person certified in DSDM Agile Project Framework whose role is to help less experienced team use this approach properly and effectively.

  DSDM stands for **dynamic systems development method**. DSDM Agile Project Framework integrates a **project management lifecycle** and a **product development lifecycle** into a single framework. The project process includes seven phases: **Pre-Project, Feasibility, Foundations, Evolutionary Development, Deployment, Post-Project**. A crucial element of this method is the **Prioritized Requirements List (PRL)** which includes high-level requirements established during Feasibility and Foundations phases, prioritized using the **MoSCoW technique**. The MoSCoW technique involves prioritizing the requirements according to the following rules: M (**must have requirements**

B. Match the word from the left with the one from the right to form full expression and translate it into Polish.

1) gather	**a)** story	.....
2) user	**b)** integration	.....
3) baseline	**c)** progress	.....
4) task	**d)** factor	.....
5) continuous	**e)** requirements	*zbierać wymagania*
6) track	**f)** user	.....
7) user	**g)** board	.....
8) focus	**h)** upfront	.....
9) intended	**i)** architecture	.....
10) design	**j)** experience	.....

C. Fill in the prepositions in the following sentences. The first one has been done for you.

1. Presently software development practices are subject *to* constant evolution.

2. Software development approaches can be divided ..... heavyweight and lightweight methodologies.

3. In waterfall software development approach potential problems can be investigated and bottomed ..... during the design phase, before implementation of the solution. The process is also well documented so ..... the end there are no misunderstandings.

4. RUP methodology is strongly tied ..... using object-oriented modelling.

5. The Scrum Team which comprises ..... the Product Owner, the Development Team and the Scrum Master agrees ..... a set of items to be addressed ..... a Sprint.

6. According to Agile Manifesto, individuals and interactions which help to get rid ..... any misunderstandings between the team members are more important than processes and tools.

7. The basis ..... lightweight software development methodologies is the Agile Manifesto which presents key values ..... the philosophy behind them.

8. In Agile a working software is delivered ..... the end of each iteration.

9. In FDD when each team successfully finishes the implementation of the feature, it is validated ..... the client and integrated ..... the main product.

10. According to Agile Manifesto, providing a working software ..... minimal value is preferred ..... development of the entire software ..... detail.

to (x2)	over	of (x3)	by	on	out	with	for	in (x3)	at	into