# Athlone Institute of Technology

## Faculty of Engineering & Informatics

## Semester 2 Examinations 2018

## Summer Session


**Bachelor of Engineering (Hons) in Software Engineering**

**Year 4**

**Agile Methodologies 4.2**

**External Examiner(s):** **Dr Norah Power & Mr Padraig Brennan**

**Internal Examiner(s):** **Mr Michael P. Russell**

**Instructions to Candidates:** ***(****make sure you have received the* ***correct*** *exam paper****)***

**Answer ALL Questions.**

**Time allowed: 2 Hours**
**No. of pages (including cover sheet): 3**

Q.1. A client requires that you develop a Java method to convert US dollars to Euros based on the current exchange rate. The method requires that both the amount of dollars to be converted and the current exchange rate be specified.

(i) Develop a user story to capture the client's requirements. Document any assumptions made.

As a User

I want to convert US dollars to Euros based on the current exchange rate

So I can get the currency in Euros

Acceptance Criteria :

Amount of Dollars should be valid

Current exchange rate should be valid

Error thrown for incorrect inputs

Amount in Euros (converted) = Amount of Dollars * Exchange rate

Correct amount in euros should be displayed for valid inputs

(6 marks)

(ii) Design a suitable product skeleton in Java to implement the user story.

```
public class ConvertBetweenCurrencies {

// Partial Product Skeleton Code

double convertCurrency (double exchangeRate, double amountUSD ) {

	// This method converts US dollar values to Euro values based on the
	// current exchange rate between U.S. dollars and and the Euro.
	// On successful conversion, the Euro amount is returned.

	// Additional Requirement - Define your own exception handler to
	// handle any invalid values.

	throw new RuntimeException("Product code not yet written");

}

}
```

---

```
public class ConvertBetweenCurrenciesExceptionHandler extends Exception {

String message;

public ConvertBetweenCurrenciesExceptionHandler(String errMessage){
	message = errMessage;
}

public String getMessage() {
	return message;
}


}
```

---

Note: Create this class automatically within Eclipse

```
import junit.framework.TestCase;

public class ConvertBetweenCurrenciesTest extends TestCase {

}
```

---

(4 marks)

(iii) Design a suitable Test Design Template to document designed Unit Tests.

(2 marks)

1. Test number
2. Test objective
3. Test type
4. Inputs
5. Exp Outputs
6. Test Procedures

(iv) Employ Boundary Value Analysis to design relevant Unit Tests.

//EQUIVALENCE_____

// Test no. 1

// Objective: Verify exchange rate and USD are positive

// Input(s): exchange rate = 1.2, USD = 23

// Expected output: 27.6

// Test no. 2

// Objective: Verify exchange rate and USD are negative

// Input(s): exchange rate = -1.2, USD = -23

// Expected output: Invalid exchange rate

//BOUNDARY VALUE_____

// Test no. 3

// Objective: Verify exchange rate and USD are greater than 0

// Input(s): exchange rate = 0.1, USD = 0.1

// Expected output: 0.01

// Test no. 4

// Objective: Verify exchange rate and USD are 0

// Input(s): exchange rate = 0, USD = 0

// Expected output: Invalid exchange rate

// Test no. 5

// Objective: Verify exchange rate and USD are max

// Input(s): exchange rate = Double.MAX_VALUE, USD = Double.MAX_VALUE

// Expected output: Too large value

// Test no. 6

// Objective: Verify exchange rate and USD are min

// Input(s): exchange rate = Double.MIN_VALUE, USD = Double.MIN_VALUE

// Expected output: Invalid exchange rate

(v) Ensure that 100% Boundary Value Analysis coverage is achieved.

(4 marks)

```
public void testconvertCurrency002()
{
ConvertBetweenCurrencies test1 = new ConvertBetweenCurrencies();
try
{
Double result = test1.convertCurrency(-1.2, -23);
fail("Exception expected");
}
catch (ConvertBetweenCurrenciesExceptionHandler e) {
assertEquals("Invalid exchange rate", e.getMessage());
}
        }
```

```
public void testconvertCurrency001()
{
ConvertBetweenCurrencies test1 = new ConvertBetweenCurrencies();
try
{
Double result = test1.convertCurrency(1.2, 23);
assertEquals(27.6, result);
}
catch (ConvertBetweenCurrenciesExceptionHandler e) {
fail("Exception not expected");
}
        }
```

**[20 marks]**

Q.2. (a) In planning for a sprint the following inputs need to be considered:

Describe each input and the role it fulfils within the Scrum/Sprint Planning process.

- story point estimation,

  - *Story points* are a unit of measure for expressing an *estimate* of the overall effort that will be required to fully implement a product backlog item
  - A **story point** is a high-level **estimation** of complexity involved in the user **stories**. **Story points** along with sprint velocity provide a guideline about the **stories** to be completed in the coming sprints.

- team velocity,

  - At the end of each iteration, the team adds up effort estimates associated with user stories that were completed during that iteration. This total is called velocity. Within Agile, as teams spend time together and work through a few sprints, they start to get a sense for their “velocity”: This is the amount of work that the team can usually deliver within the time frame of a sprint or iteration and can be used as a predictor for future iterations

  Knowing velocity, the team can compute (or revise) an estimate of how long the project will take to complete, based on the estimates associated with remaining user stories and assuming that velocity over the remaining iterations will remain approximately the same. This is generally an accurate prediction, even though rarely a precise one.

- definition of done,

  - The team agrees on, and displays prominently somewhere in the team room, a list of criteria which must be met before a product increment "often a user story" is considered "done". Failure to meet these criteria at the end of a sprint normally implies that the work should not be counted toward that sprint's velocity.

Every new team should dedicate time to discussing and agreeing on their definition of done, which may or may not include elements such as code that is checked in and integrated and has automated tests. This is a meaningful conversation that the development teams, product owner, and stakeholders need to have to ensure a common understanding of completion.

When the definition of done is clear and well understood, then we ensure that the conclusion of every sprint will meet everyone’s expectations

- incorporation of technical debt, and  management of bugs/defects.

With Agile, teams are moving very quickly, and sometimes to achieve the sprint goal within the time frame allotted, the team, either knowingly or unknowingly, creates technical debt.

Technical Debt includes those internal things that you choose not to do now, but which will impede future development if left undone. This includes deferred refactoring.

• There are two basic kinds of debt—unintended and intentional

Unintended technical debt is simply the consequences of the team's learning curve.
• Perhaps they designed something poorly or did not test thoroughly enough, or the business was not 100% certain about the business requirements.
• No matter what the root cause, this type of debt is unintentional—no one meant to write bad code or be sloppy—and it happens as part of the continuous learning process that is essential to Agile.

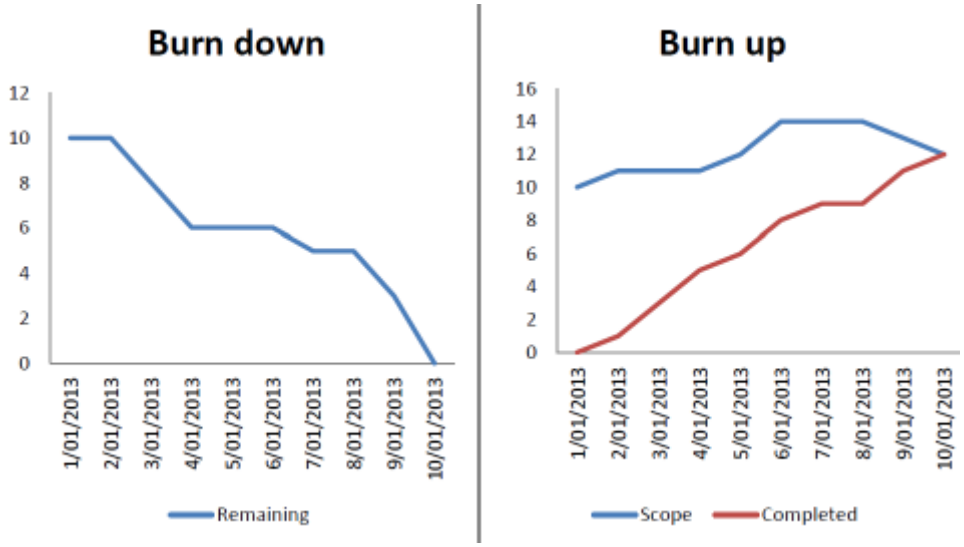
Intentional technical debt is incurred as trade-offs are made in the development process.
• We may choose to incur technical debt for short-term, tactical reasons or long-term, strategic ones.
• For example, creation of an un-scalable feature for the initial release, knowing that at some late point the feature will need to be re-done to its scalability requirements.

Bugs, also referred to as defects, are different from technical debt in that they are usually errors in the writing or implementation of the code that are impeding the performance or usability of the application.
• In many companies, bugs are handled outside of the feature release (sprint) process because of their immediacy.
• Within organizations where bug fixes are resource intensive and can wait for the development cycle, bugs should also be included in the backlog and prioritized accordingly.

(10 marks)

(b) What is the difference between a burn-up chart and a burn-down chart? How do you know from a burn-down chart if you are behind schedule?

Burn down and burn up charts are two types of charts that project managers use to track and communicate the progress of their projects. A burn down chart shows how much work is remaining to be done in the project, whereas a burn up shows how much work has been completed, and the total amount of work. These charts are particularly widely used in Agile and scrum software project management.

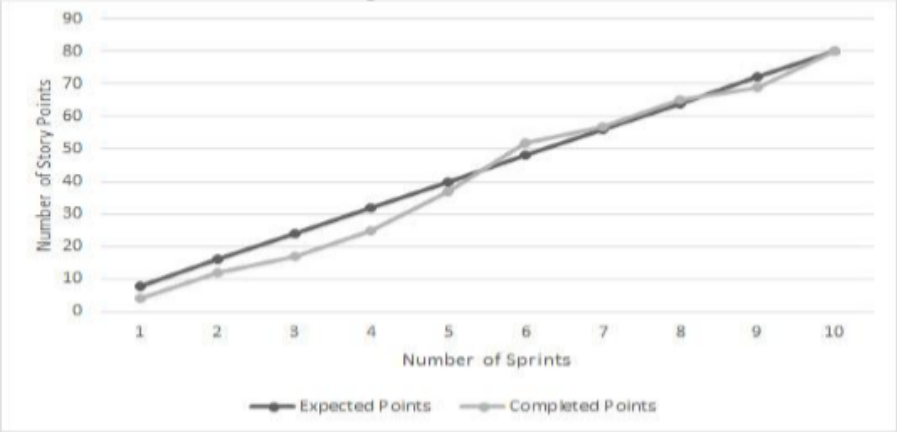



Burn-up charts are a way to depict progress toward a product's release goal.

• Typically, time is on the X axis, either in sprints, months, or quarters, and the Y axis represents the release at completion.

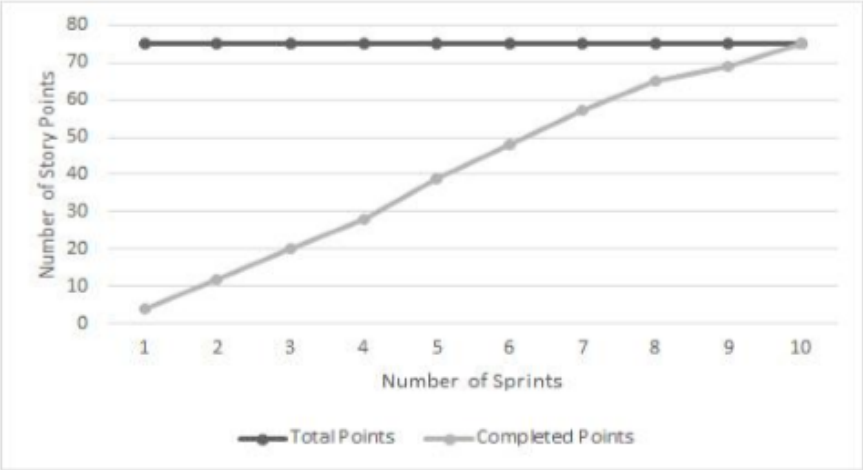
• The Y axis can be either story points or feature descriptions.

## Example burn-up chart by expected points




- The black line is the cadence that we expect to see relative to the number of story points delivered in each sprint.
- The grey line represents actual story points delivered.

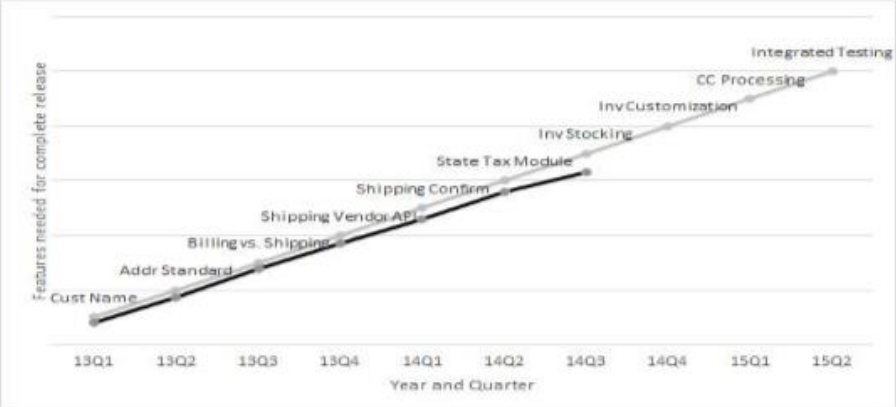
# Example burn-up chart by total points




- Here we see the black line as the number of story points that we need to achieve to meet our release goals.
- Our grey "actual" line tracks upward, and we meet the goal after Sprint 10.

(5 marks)

# Burn-up chart by feature




- This graph shows the feature milestones that we need to complete each quarter to realize the vision for this product.
- The actual progress is then tracked against these features, as completed.

Although the first two graphs do reveal the progress that the team is making, the charts may not convey the necessary information to stakeholders.

• It is interesting to note the team's velocity and how quickly they are delivering story points, but that does not necessarily mean that the product is producing the right deliverables.

• The third graph shows the actual features that are delivered to the marketplace. Executives often prefer this view because they can easily see the feature delivery progress.

Burn-down charts serve a different purpose: These are the daily status checks for the team relative to where they expected to be at a particular point in time.

• This is a critical tracking element within a sprint or iteration to ensure that any necessary course correction is identified and addressed as early as possible.

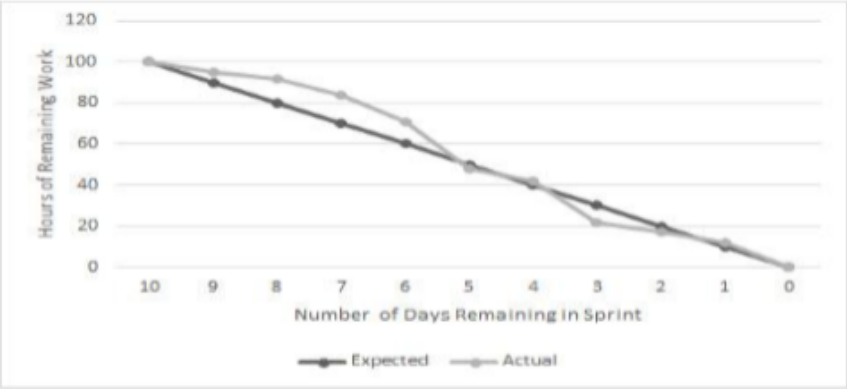
Before Sprint planning, most activities are at the user story level, and they are estimated in story points.

• Once Sprint planning is completed, stories are broken down into tasks, and tasks are estimated in hours.

• Now that the developer is known, and he or she has small increments of work, it is easier to estimate the actual amount of time a task will take.

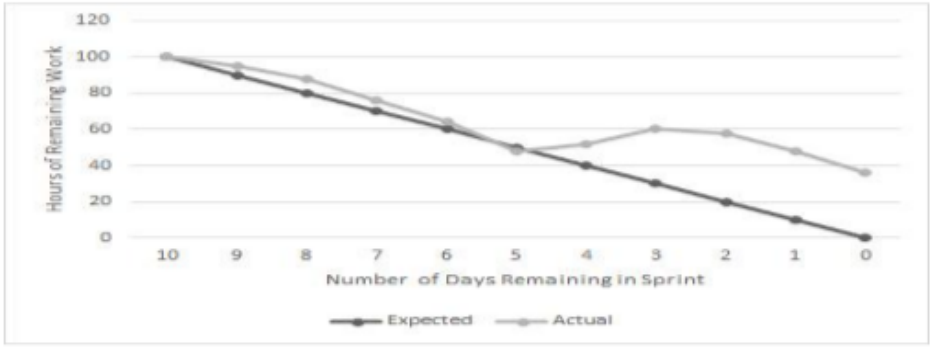
• Therefore, the team can see the total number of hours allocated to the sprint and track their actual progress against the target.

# Burn-down Charts




The black "expected" line represents the ideal sprint progress if the same level of work was completed every day. You can tell from the graph that the sprint got off of a rocky start because our grey "actual" line was above the "expected" line for several days. Whenever this happens, the team must determine if they are able to finish the expected amount of work within the sprint. In this case, the team recovered nicely, and with three days remaining in the sprint, they were actually ahead of schedule. They finished the last day with all of the work completed.

# Burn-down Charts




As you can see from this graph, something went horribly wrong on day 5. Perhaps several team members were out sick with the flu; or the team uncovered a significant problem with the database; or a production issue occurred and the entire development team was pulled off the sprint to work on the live issue; or a story ended up being significantly more difficult than originally thought.

In any case, this burn-down graph clearly shows that the sprint is in jeopardy, and we can see this almost immediately.

• The team has several options to try to remedy the situation.

• For example, they can reassign tasks to the most competent developers to speed things up, or they can work long hours (with lots of caffeine) to make up the difference (noting that this is not a sustainable way to manage work).

• They can also negotiate with the product owner about the situation and the best resolution.

(c) What is the purpose of the stand-up meeting? What three key questions are answered by team members during a daily stand-up meeting? Who typically gets assigned action items during a daily stand-up meeting?

(5 marks)

Development teams use the daily stand-up meeting to do a quick status check and prepare the team for the day.

• These meetings usually take less than 15 minutes and provide an opportunity to understand the team's progress in the iteration.

• Each team member answers three key questions during the daily stand-up meeting:

- **What did I do yesterday?** This is an opportunity for a developer to share what tasks were closed yesterday or if something was more difficult than expected or will require more time.
- **What am I planning to do today?** This information helps the team know the developer's area of focus for the day and also if he or she will be unavailable for some portion of time because of meetings or personal considerations. By understanding what a teammate is working on, the other team members can align their work, if necessary.
- **Is there anything blocking my progress?** This could be anything that is preventing advancement, from a technical issue to the lack of a test environment to a clarification question with the product owner. The key is to share challenges to keep things moving forward in the iteration. The output of this discussion serves as the action items for the Scrum master.