We are going to add Date arithmetic functionality to UniTime Project and this would typically involve these steps:

1. Determine the requirements: Review the project requirements and determine which date arithmetic operations are required and how they will be used.
2. Choose a date library: Decide which date library to use. Java provides several libraries for working with dates and times, such as java.util.Date, java.util.Calendar, and the newer java.time package. Choose the library that best meets the project requirements.
3. Design the API: Define a set of methods that will perform the required date arithmetic operations. Consider the input and output types of the methods, as well as any error handling that may be necessary.
4. Implement the methods: Implement the methods using the chosen date library. Be sure to test the methods thoroughly to ensure that they work correctly in all cases.
5. Integrate with the project: Integrate the new date arithmetic functionality with the project, making any necessary changes to existing code or adding new code as needed. Test the integration to ensure that it works correctly and does not introduce any new bugs.
6. Document the API: Document the new date arithmetic API to make it easy for other developers to use and understand. Provide examples and usage scenarios to help developers get started quickly.
7. Maintain the code: Continuously maintain and update the date arithmetic code to ensure that it remains up-to-date and meets the changing needs of the project.

Functions To Be Added:

addDays(Date date, *int* days): This method takes a Date object and an integer representing the number of days to add to the date. It creates a new Calendar object, sets its time to the specified date, adds the specified number of days, and returns the resulting date as a Date object

addWeeks(Date date, *int* weeks): This method takes a Date object and an integer representing the number of weeks to add to the date. It creates a new Calendar object, sets its time to the specified date, adds the specified number of weeks, and returns the resulting date as a Date object.

addMonths(Date date, *int* weeks): This method takes a Date object and an integer representing the number of months to add to the date. It creates a new Calendar object, sets its time to the specified date, adds the specified number of months, and returns the resulting date as a Date object.

durationInDays(Date start, Date end): This method takes two Date objects representing a start date and an end date. It calculates the difference between the two dates in milliseconds, divides the result by the number of milliseconds in a day, and returns the resulting number of days as a long.

durationInWeeks(Date start, Date end): This method takes two Date objects representing a start date and an end date. It calculates the difference between the two dates in milliseconds, divides the result by the number of milliseconds in a week, and returns the resulting number of weeks as an int.

durationInMonths(Date start, Date end): This method takes two Date objects representing a start date and an end date. It creates two Calendar objects, sets their times to the start and end dates, and calculates the difference between the months of the two dates, taking into account any difference in years. It returns the total number of months between the two dates as an int.