Formative Assessment Guidelines

This matrix is to give you in-sight to the guidelines your teachers use to give you feedback related to the learning outcomes (LOs). Note: your teacher can deviate from this matrix based on your situation.

	LO1 Justify stakeholder feedback	LO2 Communication	LO3 Analysis and design	LO4 OO Concepts	LO5 Algorithms	LO6 Databases	LO7 Code quality
Undefined	No explanation is given about made decisions. Feedback received from the teacher(s) and peers has not been incorporated.	You are rarely present during lectures, practical, demos and meetings. You rarely meet the expected deadlines from the teachers with the expected deliverables. During collaboration with peers, no feedback is given by you.	No documentation is delivered (e.g. no project plan, UML Class Diagram, Ideation Document, URS, etc.).	Delivered implementation(s) use single class (or none). No application is delivered using ASP.NET Core Razor Pages.	Delivered application(s) contain only simple CRUD functionalities	No database(s) delivered	Git is not used during development. Nothing is delivered related to testing (e.g. unit tests, test plan or test report).
Orienting	Incomplete explanation is given about made decisions but it is superficial and contains mistakes (e.g. terms used without relating them to the deliverables). Mostly feedback related to easy-fixes have been incorporated	Most of the time you are present during lectures, practical, demos and meetings, but non-responsive/passive. Most of the time you meet the expected deadlines from the teachers with the expected deliverables. During collaboration with peers, feedback given by you to peers are shallow and not really usable to receiver to improve/keep positive points.	Expected documentations are delivered, but the content is incomplete and contains mistakes. For example: - Project plan is missing expected chapter(s). - UML Class Diagram(s) contain format mistakes and simple OO principles are applied (i.e. associations, inheritance) with mistakes; - URS is still unfinished and/or written from the wrong perspective (e.g. not sea level UCs, missing UC components, etc.).	Delivered implementation(s) use multiple classes but they are not really connected/related to each other (e.g. no really OO, relying on ID for associated classes, etc.). Razor Page application(s) are delivered with only the views containing the static content.	Delivered application(s) contain simple CRUD functionalities with simple input validation (e.g. non empty, handling wrong types, etc.).	Expected database(s) are delivered but contain big mistakes such as missing tables, not correctly normalized, missing keys, missing relationships, etc. Application(s) contain some queries to perform CRUD actions, but are still incomplete.	Git is used during development for both individual as collaboration projects, but the commits are minimal. Delivered application(s) contain unit tests but do not cover all the units in the logic layer and/or incomplete. A test plan is created for the 'synthesis assignment' & project, but the test cases are incomplete /missing details to perform the tests.
Beginning	Explanation given about made decisions are explicit but still contain mistakes and the student does not always realize it during feedback. Most of the feedback has been incorporated, but similar mistakes are repeated later on	You are present during lectures, practical, demos and meetings, but often unable to answer question when queried. You meet the expected deadlines from the teachers with the expected deliverables. During collaboration with peers, feedback given by you to peers are shallow and not really usable to the receiver to improve/keep positive points.	Expected documentations are delivered, but the content is incomplete/lacking. For example: - Project plan contains expected chapters but information included is missing details to allow all stakeholders to understand the context of the project, what will be delivered and how it can successfully be achieved. - UML Class Diagram(s) only have simple OO principles applied (i.e. associations, inheritance) correctly; - URS contains all the expected FRs and/or UCs, but there are still (minor) mistakes (e.g. missing exceptions/extensions, constraints, etc.) and inconsistency between the FRs/UCs/application (e.g. login as FR but no UC, UC about updating expects unexpected data when considering UC about adding, etc.).	Delivered implementation(s) match most of the UML Class Diagram design, if any, and applies simple OO principles (i.e. correct usage of associations, encapsulation, inheritance, polymorphism). Razor Page application(s) are delivered but do not always make use of the framework's build-in functionalities covered during WAD (e.g. layout, input validation, model binding, etc.).	Delivered application(s) contain CRUD functionalities with proper input validation (e.g. non empty, uniqueness, handling wrong types, ranges, simple regular expressions, etc.).	Expected database(s) are delivered, but sill contains minor mistakes or do not support the code design (e.g. missing intersection tables, tables to store inheritance, etc.). Application(s) contain all queries for CRUD actions, but are sub-optimal (e.g. using code to join data instead of using queries for joins, retrieving primary key with a select max(id), etc.). Querying the database via an application is sub-optimal, e.g. multiple queries should be combined into one.	Git is used during development for both individual and collaboration projects, but during collaboration no branching is used and it is hard to trace back who did what. Delivered application(s) contain unit tests to test units in the logic layer but only cover the happy-flow. A test plan is created for the 'synthesis assignment' & project, but the test cases only cover the happy-flow.
Proficient	Explanation given about made decisions are explicit but during feedback sessions minor mistakes are discovered which the student is able to correct after minimal hints from the teacher All of the feedback has been incorporated and similar mistakes seldom happen anymore	You are present during lectures, practical, demos and meetings and, when required, is able to answer questions. You meet the expected deadlines from the stakeholders (teachers and peers) with the expected deliverables. During collaboration with peers, feedback given by you to peers are usable to the receiver after some more discussions. In addition, features of GitLab are used to support a professional way of collaboration (e.g. issue tracking, issue board, etc.).	Expected documentations are delivered with appropriate content. For example: - Project plan contains expected chapters and allows all stakeholders to understand the context of the project, what will be delivered and how it can successfully be achieved. - UML Class Diagram(s) have correct application of only intermediate OO principles (i.e. associations, encapsulation, inheritance, layered design and SRP); - URS contains all the expected FRs and/or UCs and does not completely match with the most recent version of the application. Minor inconsistencies are present but overall does not impact how the UCs/implementation behaves.	Delivered implementation(s) match most of the class diagram design, if any, and apply intermediate OO principles (i.e. correct usage of associations, encapsulation, inheritance, polymorphism, layered design and SRP). Razor Page application(s) are delivered and make use of the framework's build-in functionalities (e.g. layout, input validation, model binding, etc.) covered during WAD.	Delivered application(s) contain simple CRUD functionalities with proper input validation (e.g. non empty, uniqueness, handling wrong types, ranges, regular expressions, etc.). In addition, the application(s) contain functionalities implementing business rules (e.g. free shipping based on x, y, z, limiting input based on related entities, etc.) and/or algorithms to solve non-trivial functional requirements	Expected database(s) are delivered to support the application(s), but sill contain minor mistakes. Application(s) contain all queries for CRUD actions, but the more complex queries are still sub-optimal. Querying the database via an application is kept at a minimum (e.g. smart/combined queries should be used instead of loops sending multiple queries).	Git is used during development for both individual and collaboration projects, but a basic branching strategy is used (e.g. a branch for each developer), and, during collaboration, it is possible to trace back who did what. Unit tests are present to test units in the logic layer covering the happy-flow and most important alternative-flows are also tested and covered. AAA structure is applied to keep unit test readable. When required, Dependency Injection is applied to decouple logic and data access layer to allow proper testing. A test plan and test report are delivered for the 'synthesis assignment' & project and the test cases cover the happy-flow and the most important alternative-flows.
Advanced	Explanation given about made decisions are explicit and during feedback sessions any minor mistakes the student realizes are corrected without any hints from the teacher Most of the feedback have been incorporated and a pro-active attitude is shown related to double checking for similar mistakes, proposing possible fixes/workarounds, etc.	You have a pro-active participation during lectures, practical, demos and meetings. You meet the expected deadlines from the stakeholders (teachers and peers) with the expected deliverables. During collaboration with peers, feedback given by you to peers are constructive to the receiver. In addition to GitLab, other tools are properly used to support a professional way of collaboration.	Expected documentations are delivered with appropriate and up-to-date content. For example: - Project plan contains expected chapters are kept up-to-date (i.e. a living document) and allows all stakeholders to understand the context of the project, what will be delivered and how it can successfully be achieved. - UML Class Diagram(s) have correct application of the covered OO principles (i.e. associations, encapsulation, inheritance, layered design and SOLID) and matches the implementation; - URS contains all the expected FRs and UCs that are up-to-date (i.e. treated as a living document).	Delivered implementation(s) match the class diagram design , if any, and apply covered OO principles (i.e. correct usage of associations, encapsulation, inheritance, SOLID) correctly Razor Page application(s) are delivered and make use of the framework's build-in functionalities (e.g. layout, input validation, model binding, etc.) covered during WAD.	Delivered application(s) contain simple CRUD functionalities with proper input validation (e.g. non empty, uniqueness, handling wrong types, ranges, regular expressions, etc.). In addition, the application(s) contain functionalities implementing business rules (e.g. free shipping based on x, y, z, limiting input based on related entities, etc.) and algorithms to solve non-trivial functional requirements From an implementation perspective, proper data structures are used/included to support non-trivial functionalities	Expected database(s) are delivered and support the application design properly. Application(s) contain optimized queries for CRUD actions. Querying the database via the application is kept at a minimum (e.g. smart/combined queries instead of loops sending simple queries). SQL functions (e.g. MIN, MAX, SUM, EXTRACT, etc.) are used when applicable.	Git is used during development for both individual as collaboration project; branching strategies are used, commits are easily trace-back to each developers task, releases are tagged appropriately, etc. Unit tests are present to test units in the application covering happy and alternative-flows. AAA structure and SOLID principles are applied to keep unit test maintainable. When required, Dependency Injection is applied to decouple logic and data access layers to allow proper testing. A test plan and test report are delivered for the 'synthesis assignment' & project and cover both happy- and alternative-flows.