Q1.1*(1 point) : Dans le diagramme de cas d'utilisation d'un réseau social type FaceBook, on a modélisé que le use case « Poster Message » inclut (<>) le cas d'utilisation « se loger ». Critiquer cette modélisation.*
A priori, non, on ne doit pas se « re »-loger à chaque fois qu'on poste, on cherche ici une relation de précédence, qui ne s'exprime pas directement sur ces diagrammes. On préfére une modélisation avec deux acteurs, l'un étant authentifié et pouvant « poster », l'autre ne pouvant que « se loger » dans un premier temps.

Q1.2*(1.5 point) : Dans une approche en boite noire basée sur les tests, expliquez les termes : oracle, verdict.*
Oracle : interprète la sortie fournie par le système et détermine le verdict. Dans les cas simples il s'agit simplement de comparer la sortie obtenue à une valeur de contrôle, dans les cas plus complexes (e.g. indéterminisme…) l'oracle peut être plus complexe (e.g. comparer la sortie avec celle d'un outil de référence, autoriser des réordonnancements…).
Verdict : La conclusion de l'Oracle, pour un test donné sur un système donné. Le verdict est le plus souvent : PASS/FAIL, mais on peut rencontrer parfois d'autres valeurs comme TIMEOUT, INCONCLUSIVE…

Q1.3 *(1 point) : Pourquoi les classes métier ne contiennent-elles pas d'opérations ? Pourquoi ne précise-t-on pas la navigabilité des associations ?*
Les classes métier ne *sont PAS* des classes au sens OO habituel, c'est une modélisation du domaine métier. Ces « classes métier » ne sont pas des artefacts techniques liés à la solution logicielle, mais bien un artefact qui spécifie en analyse les données et leurs liens. L'attribution de méthodes aux classes est une tâche de Conception. De même, la navigabilité des associations est une problématique de conception, pas d'analyse.

Q1.4 *(1,5 point): Quels sont les trois éléments qu'un langage de description d'architecture doit définir séparément ?*
1. Les connecteurs (e.g. interfaces UML)
2. 2. Les composants, offrent/requièrent des connecteurs (e.g. composants UML)
3. 3. La topologie d'instanciation, la description d'une configuration des composants instanciés (e.g. dia de structure interne UML)

Q1.1 : Quel est la nature d'un bus logiciel dans une plateforme orientée composants ? Quels problèmes en particulier sont traités par un bus logiciel ?
Offre une Api de communication entre les composants : empaqueter les requêtes, les véhiculer à la cible (système de nommage), les désérialiser, retransmettre la réponse
Permet de s'affranchir de l'hétérogénéité : plateforme d'éxecution + langages réalisant les composants

Q1.2 : Comment peut-on s'assurer que le jeu de tests de validation est complet ? On ne peut pas être « complet ».
On ne peut que se satisfaire de métriques, e.g. couverture des SN, ALT et EXC des fiches détaillées avec au moins 1 jeu de données.

Q1.3 : On considère le diagramme de cas d'utilisation d'un système de vente en ligne. Comment modéliser que le cas d'utilisation « mettre en vente » précède nécessairement le cas d'utilisation « acheter produit » ?
On ne peut pas le faire directement sur le diagramme ; ni extends ni include ne répondent à cette question de séquencement temporel. On peut s'en sortir avec différents acteurs (e.g. connexion + acteur Utilisateur Connecté)
Ou avec des préconditions/postconditions :
? mettre en vente => post : il y a des produits.
? Et « acheter produit » précondtion « il y a des produits »

Q1.4 : Pourquoi ne peut-on se contenter du code pour décrire une application ? Pourquoi le code reste-t-il cependant nécessaire ?
Diversité des intevenants, niveau d'abstraction unique (faible) : le code n'est pas l'artefact qui est adapté au métier de tous les intervenants
Importance pour discuter entre intervenants « code », e.g. algorithmes, couches technique. Ca reste un artefact important du développement pour les développeurs + construire l'executable.

Q1.1 : Dans un diagramme de cas d'utilisation décrivant un site web bancaire, on a modélisé des <> depuis les cas d'utilisation critiques (consulter compte, effectuer virement…) vers un cas d'utilisation « ouvrir session connectée ». Critiquez cette approche.

Include ne traduit pas la précédence, c'est incorrect si le comportement c'est de se loger une fois, puis de faire plusieurs opérations sans redonner les identifiants. Il vaut mieux modéliser avec des acteurs différents représentant les catégories de droits des utilisateurs sur le système.

Q1.2 : Comment modéliser une boucle (e.g. plusieurs questions dans un QCM) dans la description d'une fiche détaillée de cas d'utilisation ?
On doit modéliser avec une alternative le comportement qui boucle. Le mieux est de placer une étape dans le SN qui teste la fin de boucle.
Sur l'exemple : SN6 : le système affiche la question… …réponse étudiant
SN9 : le système vérifie que c'est la dernière question du QCM
SN10 : la système affiche le score …
ALT : A1 : autres questions
A1.1 en SN9, il reste des questions dans le QCM
A1.2 : retour en SN6 avec la question suivante

Q1.3 : On considère le composant CComp qui offre une interface IOffert portant l'opération +foo() et qui requiert une interface IRequis portant une operation +bar(). Représentez sur un diagramme de classe une conception détaillée possible du package «comp » réalisant ce composant.

Q1.4 : Comment élaborer un jeu de test complet, qui garantisse qu'un système est correct?
Impossible sauf cas finis très petits, par essence on ne prouve pas par l'exemple, le test peut révéler des fautes mais ne peut pas garantir la correction.

Q1.1 : Pourquoi n'utiliser que des types simples dans les signatures des opérations d'interface des composants ?
Portabilité, + ça n'induit que des dépendances fonctionnelles. Echanger des classes viole le principe d'encapsulation.

Q1.2 : Qu'est-ce qu'une métrique de qualité logicielle ? Citez deux exemples.
Métrique : Un outil objectif qui fournit une mesure quantitative quand on l'applique à un sujet donné, ici on veut mesurer de la « qualité » et le sujet c'est donc un logiciel (ou un modèle, ou un procédé… la qualité logicielle c'est vaste)
La valeur absolue doit s'interpréter dans le contexte du projet (certains projets sont plus complexe ou gros intrinsèquement), souvent on surveille plus son évolution au fil du temps que sa valeur absolue, e.g. les variations rapides peuvent alerter sur des problèmes réels.
Exemples : nombre de lignes de code total, nombre de classes, profondeur des arbres d'héritage, nombre de cycles de dépendance, nombre d'opérations par classe, lignes de code par classe, lignes de code par méthode (moyen et max), diverses métriques de couplage interne des classes et inter classes existent…

Q1.3 : Expliquez à quel moment(s) dans le projet il faut réaliser les étapes de l'Analyse (branche construction ET branche validation) du cycle en V présenté dans l'UE.
Début projet
Analyse + rédaction des tests de validation
Présentation TV au client (retour éventuel en analyse)
Conception et réalisation (TUNNEL)
Recette (basée sur TV)

Q1.4 : Quels sont les alternatives possibles pour intégrer des éléments algorithmiques à un modèle UML ?
Alt 1 : modélisation à l'aide de Behavior UML, e.g. diagrammes d'activité
Alt 2 : notes de code, pas strictement UML.

Q1.1 : Expliquez la nature d'une dépendance fonctionnelle et d'une dépendance structurelle. Laquelle préférer en général ?
Fonctionnel = ens de méthodes, e.g. interface Structurel = connaitre une classe concrète On préfère fonctionnel.

Q1.2 : Quelles métriques peut-on proposer pour mesurer la couverture des tests de validation ?
Couvrir tous les use case, tous les Alt et Exc des fiches détaillées, utiliser plusieurs jeux de données pour chaque test…

Q1.3 : Quel intérêt présentent les tables de hash pour la réalisation en orienté-objet d'un composant qui présente une interface basée sur des identifiants ?
Pour indexer les objets stockés via leur identifiant. On a du O(1) pour retrouver les objets.

Q1.4 : A quel moment et dans quel but construit-on des diagrammes de séquence représentant le système opposé aux acteurs ?
En analyse pour découvrir les opérations (responsabilités) du système et les données que lui fournissent les acteurs. Ils servent également à l'élaboration des Tests de Validation.

Q1.1 : Pourquoi parle-t-on de rupture du cycle de vie du logiciel entre l'analyse et la conception ?
On passe du QUOI (Problème) au Comment (Solution).

Q1.2 : Le test permet-il de prouver qu'un système fonctionne correctement ? Justifiez votre réponse.
Non, car non exhaustif, donc on ne peut que se satisfaire d'une couverture. Exemples domaine String, séquences infinies d'appels => impossible de prouver.

Q1.3 : Pourquoi limite-t-on les signatures des opérations des interfaces des composants à des types simples (String, Bool, Float…) ou d'autres interfaces ?
Pour éviter tout dépendance à une implémentation spécifique. Cela permet de franchir les barrières entre langage hétérogènes (dans divers composants) sans problèmes. Les interfaces sont ok, car prise en charge par le modèle de composant.

Q1.4 : A quel moment et dans quel but construit-on des diagrammes de séquence représentant des instances de composants dans le cycle en V de l'ue ?
En conception générale ou architecturale pour découvrir puis figer les interfaces des composants et les grandes lignes de l'interaction + valider le découpage proposé.

Q1.1 : Pourquoi utiliser exclusivement des interfaces pour définir les interactions entre composants ? Quelles sont les principales qualités d'une conception orientée composant ?
On se limite à des dépendances fonctionnelles. Substituabilité, flexibilité, hétérogénéité des impléms (langages), possibilité de répartition, maintenance facilitée, élaboration « diviser pour régner », faible couplage…

Q1.2 : Quelle granularité de description est adoptée en analyse ? Et en conception architecturale ? En quoi peut-on voir les diagrammes de séquence en conception comme des raffinements des diagrammes de séquence d'analyse ?
Système vs acteurs
(Acteurs), composants
La ligne de vie « système » devient plusieurs lignes de vie (les composants constituant l'appli).

Q1.3 : Expliquez les notions « système à tester », « entrée », « sortie », « oracle » dans une démarche de test.
SUT : une boite noire, contenant le module/fonctionnalité à tester, instrumentée de façon à contrôler • Ses entrées : données de test
- Ses sorties : i.e. les résultats fournis par le SUT pour une entrée donnée,
- Oracle : interprète la sortie pour dire s'ils correspondent aux résultats attendus.

Q1.4 : Expliquez la différence entre un modèle et un diagramme. Donnez un exemple de lien de cohérence entre diagrammes qu'on peut établir grâce au modèle.
Modèle = ensemble d'objets
Diagramme = représentation graphique d'une partie (vue) de ces objets Opérations invoquées sur diagramme de séquence déclarées dans les classes du diagramme de classe.

Q1.5 : Expliquez à quel moment dans le cycle en V les étapes de la branche contrôle/test sont définies et exécutées.
Définition des tests : au cours de la descente, juste après l'étape de construction correspondante (Analyse -> Test de Validation …)
Exécution des tests : quand le produit remonte (i.e à la fin).

Q1.1 : Que représente le diagramme des classes d'analyse ? Pourquoi ne porte-t-on pas d'opérations sur ces classes ?

Il représente les données manipulées par l'application, issues du domaine métier de l'application. Il explique précisément la nature des données et leurs liens (références entre données, cardinalités). On peut penser à un méta-modèle, ou à un schéma de BDD type entité-association.

On est en analyse, donc on ne cherche pas encore à affecter des responsabilités. De plus ces classes métier n'ont pas vocation à servir à la réalisation de quoi que ce soit, elles décrivent simplement les données.

Q1.2 : Comment représenter qu'on doit d'abord « emprunter un livre » avant de pouvoir « restituer un livre » sur un diagramme de cas d'utilisation ?

On ne peut pas, ne doit pas, bref, question piège. On peut cependant spécifier cela avec des pré-conditions parfois, ici cela parait difficile.

Q1.3 : Quelle information apporte le diagramme de structure interne par rapport au diagramme de composants ?

Une topologie d'assemblage des composants, nombre d'instanciations, configuration particulière.

Q1.4 : Expliquez la différence entre dépendance structurelle et dépendance fonctionnelle entre parties d'une application. Laquelle préférer en général ?

Structurelle : on dérive, contient… plus généralement utilise une donnée (typiquement une classe) appartenant à l'autre module.

Fonctionnelle : on se borne à invoquer des services que fournit l'autre module.

Q1.5 : Décrivez (nature, étape…) les échanges d'informations avec le client dans le cycle en V.

Il y en a très peu :

- le CdC, du client vers l'équipe en amont de l'analyse
- le document de tests de validation, de l'équipe vers le client + approbation(contractuelle) du client
- la recette