EVALUATION GUIDELINES

Marko Schütz-Schmuck

These guidelines should be used to evaluate each team's submission. The whole project is still in process, so we do not need to see any type of _completeness_. However, we do want to see some progress (not necessarily the same amount) on each of those aspects. The guidelines and examples are more detailed than what would easily fit into the format of rubrics, so the guidelines are stated separately and the rubrics then are generic.

1 informative part

1.1 team

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines concerning team composition and description.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Team members and partners are not 100% the same: every team member is a partner, but there may be partners who are not team members (outside experts, client, …). If you think there are no other partners than the team members then it might be a good idea to justify this. On the other hand, if you think there are possibly other partners who you haven't contacted yet, then it's a good idea to document this as well. What role and responsibilities would they have?

1.2 current situation, needs, ideas

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines to describe the current situation, the needs, and the ideas.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

◊ 1.2.1 current situation

- Try to use this section to prepare for the next section of needs. For example in a project concerned with used car owner/repair history you could emphasize the point that there is a large used car market and that for used cars information about the car's history is more important than for new cars.
- If you have to make reference to data like percentages or estimates, its good if you include where you found it or a source.

◊ 1.2.2 need

- Think more about the needs of the people "out there in the domain" (actors/stakeholders) instead of the specific solution you want to create. The need is not for the platform. The platform is part of the ideas for satisfying the needs. For example, families that want to play games together need to have a place where they can find such games. This does not mention any specifics of the solution you might already have in mind and that's a good thing.
- The purpose of the section is that the developers have clear understanding of the needs independently of the system-to-be. It also concerns other parts of the project: is there a need for domain description, requirements prescription, software architecture, software component design, implementation, test plan, …? Ensure that the needs you present are not like the ideas you have for solving them. For example, in a project improving the situation with people having too many calendars "the tool should direct all the tasks and events into one single place" is not a need. There is a need for people to be able to find all their calendar events in one place. One of the ideas is to provide the tool that accomplishes that.
- As another example, in a project improving the situation when trying to schedule table-top role playing sessions: the need is not for an app to schedule. Support or simplification of the scheduling is what players would want. Whether this makes use of an app or magic would be of no concern to the players.
- You will develop domain description, requirements prescription, … But the fact that you will develop these is not a need in itself,

the need can be derived from the fact that there currently are no domain description, requirements prescription, … or at least you do not know of any, but for the project you want to carry out you need these. A related _idea_ is then to develop these yourself. A different idea to satisfy the need might be to outsource the development of e.g. the domain description to someone else. This will also affect scope, span, synopsis, design brief, and many other topics in the informative section. Of course, this is just a generic template and there will be specifics for your project.

◊ 1.2.3 ideas

- Do not go too much into specifics here (e.g. implementation) details. Show the idea in a concise manner. Bring out the idea clearly.
- You can include some features of the system-to-be, but refrain from going to much into the requirements of the system-to-be.

1.3 scope, span, and synopsis

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines to describe the scope, the span, and the synopsis.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

◊ 1.3.1 scope and span

- Scope should broadly go over all aspects of your project, such as domain, requirements, implementation, etc.
- Span is related to scope: in scope you discuss the more general, more high-level view. In span you go more after the specifics. For example, you could have a section in the descriptive part on "methodology" and describe the agile methodology and how you are using it there.
- The span is usually a more detailed view than the scope, but similar.
- Scope is the broad area in which your project is operating. For example, tourism, water-sports, beach activities, community forum, social recommendation system, … But elaborate it some more. For example, (to be further detailed) group scheduling, entertainment,

gaming, event planning, … Span is the more specific concern of the project. For example, groups of typically 3-6 players/master (and rarely up to 10), …
- The 'scope' provides a higher-level overview of the project. The 'span' refers to the detailed perspective you're providing as part of the scope. Is your scope realistic? For example in a project concerned with offering families a go-to place to find casual games: Can you actually develop several such games or are you only curating them?
- Scope and span: scope describes the broader area in which your project takes place (university student spacial and temporal orientation and support…) span describes this in more specifically (UPRM students, GPS, classrooms, assignments, …).
- In general implementation decisions do not belong in the scope and span. You could mention them in the design brief or you could not mention them at all in the informative part, but characterize the relevant goals that you have and then explain in the descriptive part (and maybe to some extent in the analytic part) your reasoning for choosing the implementation technology.
- Scope and span should focus on the project not so much on the result. For example, having a stable internet connection does not seem to belong here.
- Scope includes all those things you will do on the project. Include domain engineering, requirements engineering, software architecture, …

◊ 1.3.2 synopsis

- In the synopsis you can talk about how you'll conduct the project.
- The synopsis can and should also refer to the domain description, the requirements prescription, the software architecture, … Remember the synopsis is a concise overview about the project you have planned.

1.4 other activities than just developing source code

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on informing about activities other than implementing an application.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows the guidelines exactly.

- Your project will also concern itself with domain and requirements engineering, software architecture, component design, implementation, testing, deployment, … Since these are all things you will do, what are your ideas about these points, what specific needs are there that are satisfied by doing these things, and what about the current situation brings about these needs. For example, part of the ideas is to implement a system. Then that implementation needs requirements in order to specify what functionality needs to be implemented. The need is for the developers to have a clear understanding of the functionality of the system-to-be. The current situation is such that the developers do not have such a requirements collection available. Similarly, for domain description. This will also affect scope, span, synopsis, design brief, and many other topics in the informative section.

1.5 derived goals

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on describing secondary goals.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Your primary project goal is described by the other sections: current situation, needs, ideas, … Here you talk about secondary desirable outcomes. These should be substantially different from your primary goals and not just reiterations of those primary goals. For example, a project concerned with beach quality monitoring might have as its primary goal to give information about the cleanliness of the water and the beach to beach-goers. A secondary goal could be to increase tourism to some lesser known beaches or to raise awareness among the public about the need to keep the beaches clean.

2 descriptive part

2.1 domain description

- Functions stand for actions, operations, things that can be done to or with the entities. For example, an appointment can be "made", or

"canceled", or "attended". What is required when we would make an appointment? For who the appointment is made, for how long. What is the result of making an appointment? Well, an appointment. So, we might capture this first approach to understanding in "makeAppointment(forWho:Patient, when:Period) : Appointment". But then we might consider how to think about conflicting appointments (slots already taken). From what I just wrote there is no way to check since the function would not have access to a calendar. So, we might consider passing in a calendar, but that would bring in all of the concepts related to the calendar. Instead we might drill it down to only the bare minimum in concepts: we need to know the available periods. Then "makeAppointment(forWho:Patient, when:Period, free:List Period) : Appointment". Still we need to think about dealing with the possibility that an appointment can not be allocated if no slot long enough is available. "makeAppointment(forWho:Patient, when:Period, free:List Period) : Option Appointment" where "Option a" indicates either "Success" with a value of type 'a' or "Failure".

- When thinking about domain functions I recommend looking at them as first-class citizens. i.e. not as functions that belong to a single entity.
- Beware that the structures you define are suitable in general. For example, when describing a game as a domain entity: does every game use rounds? So should rounds be something you can observe on every game?
- Ensure that the functions, events, or behaviors are really from the domain. Think about the operations in the domain very thoroughly: for example, when you "add an event" what all do you need? What are you adding? An event. Where are you adding it to? A schedule? What are you getting as a result? A modified schedule?
- Look over the domain functions, domain events, domain behaviors and see whether they are really from the domain: i.e. are they purely introduced by the need of describing the domain independently of the application.
- We have the domain description which only explains the phenomena and concepts from the domain. "Calendar", "appointment", … would be concepts to go there in a project concerned with e.g. doctor's appointments. The domain is independent of any system-to-be. Then we have the requirements that capture what a system-to-be would have to "do". One way to capture the requirements is as user stories. Typically, user stories would make use of some concepts from the domain, but might speak of additional concepts that come e.g. from the need to implement this as a system. For example. "session".
- The domain description (rough sketch, terminology, narrative, …) is concerned only with those concepts and phenomena that can be observed in the domain independently of the system-to-be. So, be careful referring to a system in any of the descriptions of the domain. When you describe (or prescribe) the requirements you start talking about behaviors that the system must have.
- Domain description, etc. are not added to your web application. These are activities or outcomes of activities that are needed to have a well-founded project.

◊ 2.1.1 domain rough sketch

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on presenting a domain rough sketch.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Rough sketches are meant for collecting unprocessed information (or ideas) for example from brainstorming sessions, interviews, literature review,record meetings, Q&A… Your domain description rough sketch looks a bit too processed. What you write might better be placed in the domain narrative since it tells the story of the domain.
- What you explain about the small talk with Adriana and the visit to the art studio with the talks to the artists there can be documented in the analytical part where you describe the activities you are undertaking to obtain insight into the domain, the requirements, etc. Arguably, this might also belong into the descriptive part: if we consider the knowledge-building activities you are designing as outcomes of your project in their own right as opposed to only the insights deriving from them, then we would describe these activities in the descriptive part.
- You are getting there… The rough sketch would include the specifics that you were told. For example, (I'm making this up, but something similar might have been said) Adriana: "I work with very large photo prints on which I then use crayons. My prints start out with a size of at least 3' x 3'. Before the pandemic, in 2019, I sold a total of 30 such works. In 2020 I was down to 7 and all of these were bought by people who had bought other works in 2019 or 2018. Sometimes people have offered that I do a specific picture for them. I mean, they want me to start with a photo that they provide, but I have never accepted such work…" You get the point: it's typically very specific, some of it is related to the project's topic, some of it is not related, some of it might be related, but we do not yet see how. The point is then to analyze all the individual snippets and see what the relevant abstractions or concepts in them are. For example, there is the actual piece of art and there could be any number of "renderings": photos of the art, written description, audible description, … Or we could derive that there are artworks made to order and those that are made upon the artist's initiative and that this is a concept we might decide to call "impulse".

- The 'rough domain sketch' would include the specifics that you were told.
- The domain rough sketch refers to an application: the domain is that part of the observable universe that is of interest to your project, but independent of a system-to-be. Rough sketches are for collecting unprocessed information for example from brainstorming sessions, interviews, literature review, … Your domain description rough sketch mentions the application you intend to create: that's not

part of the domain.

- Rough Sketch: You're talking about your service (the application, web page, system-to-be), not the domain. The domain is the environment and stakeholders your service will be on and for. Try making as little mention of your project as possible so you can properly define the domain later on.
- The rough sketch focuses almost completely on the features that will be included in the application that will be developed as well as descriptions of how they should work. The domain is independent of any system-to-be and consists of the concepts and phenomena "out there" in the observable universe that are of interest to your project. Your rough sketch is not an unfiltered snapshot of the initial discussions of the domain that led to the formulation of the current situation. I suggest that this part be completely redone.
- Your rough sketch is not rough it's quite polished and could just as well be the narrative. Rough sketches are meant for collecting ideas in brainstorming, record meetings, Q&A, clippings from the literature, … Then you analyze the rough sketch (documented in concept analysis) and you write the domain narrative using the new concepts and the details from the rough sketch.
- What you have in the rough sketch of the domain is already quite processed.
- Rough sketches are unprocessed, often containing a lot of specifics. The concept analysis goes over the rough sketch and abstracts out meaningful concepts that help structure the specifics in the rough sketch. Based o nthis you then write the narrative.
- Rough sketches are for collecting unprocessed information for example from brainstorming sessions, interviews, literature review, … Your domain description rough sketch looks a bit too processed, what you write might better be placed in the domain narrative.
- Rough Sketch - seems like a broad version of the terminology, which isn't bad, but it would go in to the narrative section and could go a bit more into how the domain functions and its environment. Rough sketches are for collecting unprocessed information for example from brainstorming sessions, interviews, literature review, … Your domain description rough sketch looks a bit too processed, what you write might better be placed in the domain narrative. The concept analysis goes over the rough sketch and abstracts out meaningful concepts that help structure the specifics in the rough sketch. Based on this you then write the narrative. The reason to make this process explicit is so we can retrace it should our knowledge or perspective change.

◊ 2.1.2 terminology

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on the content of a terminology.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Terminology - What are the actions that can be performed with these entities? Events? Behaviors?
- Profile is not a domain concept, nor is "showing beaches". Emphasize the instantaneousness of the events. "… has just been …".
- For the functions it will help to explain the choice of parameter types. For example, "rate : Rating >< Beach >< City >< Activities rating -> Rating". Is this the creating function? In that case Rating does not yet exist and can therefore not be passed in. Is it the update function? OK, in that case the Rating parameter is needed. What about the Beach, City, Activities rating? Is a beach identified by just a name or do we need name and city and activity to rate? Or is it supposed to mean that we can rate beaches, we can rate cities, or we can rate activities? Formalism by itself is not very useful. Formalism should complement natural language explanations.
- Some of the terms in the terminology are not from the domain. For example, "account" is not a domain entity. You can widen the terminology to include any term, not just domain terms, and then annotate the terms with the most abstract phase that introduces the term. "Account" would then be annotated with requirements.
- A 'domain terminology' would define relevant terms from the domain. Very likely you will also have terms that are not from the domain, but that you also want to define. Also, use the terminology to indicate what kind of phenomenon or concept the term represents. Most of what you write here is better placed in the domain narrative.
- Terminology: Seems good for now, try to not include circular definitions. I would change "tourist attractions" for "attractions" for example. You dont make mention of the tourist in its definition either, so it doesnt seem necessary in my opinion.
- In the domain terminology you want to clarify terms from the perspective of domain. Some of the terms are actually domain concepts, but you are not relating them to the other domain terms in the terminology.
- There is a significant overlap between what is under the "Terminology" and what is under the "entities". There is no reason to have separate sections for domain terminology, domain entities, domain functions, domain events, and domain behaviors. Instead, you can merge all these points together into the terminology and annotate the entries in the terminology with the kind of concepts they are. Again, "user", "web app" would not be part of the domain. You can have a common terminology that includes terms from all phases of development. You can annotate the entries also with the most abstract phase that introduces the term.
- There is no reason to have the domain entities, domain functions, domain events, and domain behaviors separate to the domain terminology. I suggest putting all your terms into a single terminology and then to annotate them with the kind of concept or phenomenon they are. While you are at it you might also include terms that are not from the domain, but come about in the development project from other concerns than that of the domain. Of course, then you also want to annotate all the terms with the most abstract phase that introduced them. For example, "patient" is

clearly introduced by domain description, whereas "user" is introduced by requirements (several users must be able to use the system simultaneously).

- In the domain terminology you want to clarify terms from the _domain_, but "developer", "navigation bar", … are not from the domain. Terms from the domain could be "beach", "activity", "rating", … If you want to include other terms, not from the domain, you can collect a general terminology and indicate for each term the phase that first introduces the term. For example, "navigation bar" would be tagged to come from implementation/user interface. Some of the terms are actually domain concepts, but the way you define them is not at the domain level. For example, "comment". That a user uploads a comment is an application or interface concern. The domain concept "comment" is e.g. "an annotation (usually as written text) on a beach, an activity, … that helps a reader better understand the specifics and that complements the rating scale". No reference to system, upload, user is made.
- Some definitions involve terms that are unrelated to the domain. "Shares: broadcasting a platform content to their connections with other users in the platform, other social media, groups, or individuals." Platform, user, are terms that have to do with the application, but not with the domain. "Sharing" has a meaning purely in the domain, independent of any implementation on any system-to-be. What is that meaning? "User" is not a domain entity: you only have users because you want to make an application. On the domain level you only have cooks, learners, … Look over the domain functions, domain events, domain behaviors and see whether they are really from the domain: i.e. are they purely introduced by the need of describing the domain independently of the application.

◊ 2.1.3 domain terminology in relation to domain rough sketch

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on creating a terminology from a rough sketch.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- The domain terminology is not part of the rough sketch: to develop the terminology you have to analyze the rough sketch and come up with good definitions for the terms used e.g. in stakeholder interviews. So, the definitions are results of the processing and are therefore not rough.

◊ 2.1.4 narrative

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on the content of the domain narrative.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- The domain narrative tells the story of the domain. The domain is that part of the universe of interest to your project but independent of the system-to-be. So, "When a student enrolls into a course, the student has this course for the running semester unless the student drops the course." could be part of the domain narrative, but anything talking about the "application" is not part of domain narrative.
- The domain is that part of the universe that is of interest to your project, but independent of any system-to-be. Therefore, "home page", "navigation bar" etc. have no place here. You can describe these where you start talking about the actual implementation. Also, in domain narrative you want to talk about the _domain_ not about using the application.
- Narrative: Again, you're talking about the app that you will create, not the domain. The narrative is supposed to tell you how the domain currently works without your project and is a more elaborate version of your sketch.
- The first paragraph of the domain narrative talks about the necessity to understand but not about the domain itself. It's valuable, but should be placed somewhere else.
- The domain narrative is similarly not concerned with the platform, application, or system-to-be. You start with the creation of an event in the application that will notify all users that the creator of said event selects. Then the third paragraph talks about how the team plans to implement this. Implementation is also not part of the domain.
- You cannot "add an artist to the domain". The domain is the part of the observable universe that is relevant to your project but independent of the system-to-be. In your case you can observe that there are artists. You observe this in the world, but you do not add an artist to the domain. You can add an artist to the system-to-be, but that is not a domain function since it assumes the existence of the system-to-be and so is not independent of that system-to-be. Something that can be done in the domain to the artist: observe/obtain their name, or observe their art. Similarly, we can assume that for some given work of art there is a way to obtain the artist who created it. That would be a function in the

domain. We could call the function "obtainCreator : ArtWork -> Artist", but then we might decide that there may be works of art that have several creators and instead we would write "obtainCreator : ArtWork -> Set Artist". If we would want to exclude the possibility that a work of art has no creator, we could even write "obtainCreator : ArtWork -> NonEmptySet Artist". The entities you have are all from the domain, but some functionality and events that you have identified are not from the domain. Which related functionality or event exits out in the domain independent of a system-to-be?

- Similarly for domain entities: "register to the application" is not within the domain. Keeping track of sessions and allowing users to continue from past sessions is an application concern: only because you want to simulate part of the domain in the system-to-be do you need sessions, registration in the application, login, … Go over all domain entities with this in mind.
- Narrative - The domain is that part of the observed universe that is related to the project but independent of the system-to-be (which cannot yet be observed). In the context of car histories we have the phenomena and concepts of cars, owners, transfer of ownership, maintenance, repair, … These are all independent of the system-to-be. The domain description is then a documentation of that domain in a way that is suitable for further development (requirements, architecture, implementation, …).

◊ 2.1.5 events, actions, and behaviors

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on how to describe events, actions, and behaviors.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- There is a difference between the moment from which a person has just become a member (event), the act of becoming a member (action), and the process consisting of various smaller actions that together make up the "becoming a member" (behavior).

◊ 2.1.6 function signatures

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on forming insightful function signatures.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- For the function signatures think about what it takes to carry out the action, what information will be need and what will be produced/changed. For example, registering a member: Where is the new member registered? In the gym? In the gym's records? In a membership book? Depending on how you see this the signature would have to be changed. Examples registerNewMember : Gym >< Client -> Gym says that registering a client as a new member in a gym produces a (likely changed) gym, registerNewMember : MembershipBook >< Client -> MembershipBook reflects that the change is more local, it does not pull in as many dependencies since the concept membership book would be one that is accessible from gym, but other dependencies from gym would be left out. registerNewMember : MembershipBook >< Client -> MembershipBook >< Membership would report the newly created membership as part of the result of the operation. If the operation is one that can fail, you may want to provide for that possibility: registerNewMember : MembershipBook >< Client -> MembershipBook >< (Option Membership) where Option would be a datatype wrapping a possible result or providing an indicator for failure.
- Exercising would be a good candidate for a behavior, where a client performs an exercise, then changes to a different station, … possibly rests in between, …

2.2 requirements

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines concerning user stories.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Find the right granularity for the user stories. On the one hand you do not want them to be too broad. As in "as a student I want to get my enrollment done in order to have time to do more interesting things". In this case try zooming in and breaking them apart into smaller user stories. On the other hand you don't want them to be

too detailed. As in "as a student I want to click a square button in order to submit my entries in the current screen". In this case take a step back and see what the higher-up task is to which this contributes. Look for meaning to the student's enrollment tasks themselves: does this mean anything in the context of the student enrolling?

◊ 2.2.1 domain requirements

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on domain requirements.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Domain requirements are derived from observations of the domain and the decision or the insight that a phenomenon or concept needs to be "simulated" or "operated" inside the machine. These phenomena and concepts will have certain properties in the domain and since we want the system-to-be to faithfully operate that part of the domain, these properties become requirements on the system-to-be. For example, if you observe in the domain that an artwork always has at least one artist as its creator and you decide that the concepts of artwork, artist, and creating will be simulated by your system-to-be, then you can derive a domain requirement like "the system must associate to every artwork the non-empty set of artists who are the creators of that piece".
- Conversely, the requirement "The platform must be able to store the user's profile" is not a domain requirement if we assume that "user's profile" is not a domain concept. On the other hand, if you observe in the domain that comments are made by people and that the person making a comment can be identified (assuming this were so) and you decide that the concepts of comment, person, and identifying will be simulated by your system-to-be, then you can derive a domain requirement like "the system must provide a way to identify the person making a comment".
- If the concept of an "account" is not in the domain then "creating an account" is not driven by the domain and corresponding requirements are not domain requirements. WHat kind of requirements are those? We could call them application requirements since they are introduced by our wish to create an application or system-to-be.
- Requirements need to be stated in a way that they actually require something of the system-to-be. Therefore you see language such as "the system must …" or "the system shall …".
- In the requirements "must allow" is not as good as "must provide means to": one can always claim that "must allow" is satisfied: "It doesn't forbid it, so this requirement is satisfied." You might want to say "must provide means to".
- "User" is not normally a domain entity: you normally only have users

because you want to make an application. On the domain level you have for example patients and doctors as entities. Instead of ‘user’ it is better to then just use ‘patient’ or "doctor" as the entities.
- Refer to domain properties from your domain requirements. Which domain properties did the requirement come from? For example a requirement like "The system will not allow users to enter personal or group classes that are full." could reference the domain property that states "group classes have a maximum capacity of people".

◊ 2.2.2 interface requirements

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on interface requirements.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- Interface requirements are about shared phenomena and concepts: phenomena and concepts that can be observed in the domain and that are relevant to the simulation of part of the domain in the system-to-be and how they are shared between the domain and the internal representation inside the system. For example, in a project revolving around cooking, recipes, ingredients, and learning different dishes how often I have prepared a given dish is a concept from the domain and let's assume you have decided that it is relevant to the requirements of your application and so must be represented inside the system-to-be. How is value in the system-to-be initialized? From which observation in the domain? How is it updated when a dish is actually prepared in the outside world? Is there a connection to the cook's kitchen, to their stove, or does the system-to-be provide a screen in which the cook clicks a "Prepared" button to increase the number? These are all interface considerations and your choices are captured as interface requirements.
- Example, there are artworks that exist outside the system in the domain. We have decided that these need to be represented inside the system. By what means does the system go from not having any artworks within it to having representations of those artworks? We need to state requirements that characterize the initialization of the artworks in the system. We also need to characterize means to update the representation inside the system should the characteristics of an artworks in the domain change.
- There are many more interface requirements: can information about outside phenomena and concepts that was incorrectly or incompletely entered be updated? Can such information be edited? Can it be changed? In which ways? Which information should not be changed once it was entered?
- For example, a student drops a course that they had enrolled into. This is an occurrence we can observe in the domain. In order

for the system-to-be to reflect this information correctly, it needs to be entered/edited in the system. How? When?
- Some requirements that appear to be interface requirements can actually be decomposed into several parts. Some parts are concerned with the interface aspects and others e.g. with the domain aspects. That the system provides mean to the user to create an assignment is not purely an interface requirement. There is an interface requirement part to this, but there are also non-interface related requirements e.g. that assignments (internal representation) need to be created on behalf of the user.
- Many of your interface requirements can be derived by looking at the domain requirements, seeing that they require some concepts from the domain to be represented inside the system-to-be and then asking yourself: "hoe is that internal data representation first obtained? How is it kept up-to-date when the phenomenon in the domain changes?"

◊ 2.2.3 machine requirements

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on machine requirements.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- We want measurable requirements. For example, "little or no" is not measurable. What is could be the specific measurable requirement instead of "the system-to-be must have little or no outages"? Could the system crash? Which part? Could the system become slow? How slow is bearable? What should happen when more than X simultaneous users are there? Would the entire system be allowed to stop?
- Machine requirements will need some more detail as your project progresses. Possibly you need to add more specific circumstances? What does e.g. heavily loaded even mean? Some terms here might need extra clarification…
- It's OK to postpone clarification of some of these issues by saying e.g. "criteria for stability remain to be researched and defined", "acceptable degradation of services remains …"
- Indicate that the machine requirements are still under development.
- The machine requirements need to be measurable and attainable. For example, what does it mean that the platform allows 100 users? 100 users in total registered? 100 users simultaneously accessing the system? Doing what? Reading a comment, running a complex query? What happens when we get beyond 100, say 101 users?
- If you require very specific details of e.g. the software product running on the database server then you need to be able to justify why you think these are needed.

2.3 implementation

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on software architecture, software design, the use of screen designs, and source code.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows the guidelines exactly.

- Software architecture captures the big picture: what are to overall components and how are they related?
- Software design captures more detail: what is used to realize each of the individual components?
- When you use diagrams make sure that the diagrams _complement_ the textual explanation.
- You can show screen designs here also. Again, if you do this then make sure that the screens are _complementing_ e.g. usage scenarios and how these are covered/supported by the screens you preview.

◊ 2.3.1 selected fragments of the implementation

- Selected fragments of the implementation are only used when they complement other presentations. Consider this: you are explaining some concepts and operations in natural language text and that explanation alone becomes awkward, but if you show some source code fragments (or diagrams, or …) along with the natural language explanation it becomes clear. That's where you use the fragments of implementation. When adding code fragments please utilize snippets of the actual code instead of writing it on word or using screen shots. By themselves diagrams and fragments of source code are not helpful. They are only useful if there is something that you want to explain/describe, the explanation is too complicated/awkward in natural language only, and the explanation becomes much easier to understand if one includes some pieces of the implementation. So, only when these implementation details complement your documentation should you include them. Never for their own sake.

3 analytic part

3.1 concept analysis

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on domain concept analysis as well as on requirements concept analysis.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows exactly the guidelines.

- When you want to do domain concept analysis you need to take the domain rough sketch (obtained from interviews, common knowledge, literature review, …) and analyze what the common concepts are.
- For example, you hold one beach-goer's statement "I come to the beach daily to do 15 minutes of butterfly" and another beach-goer's statement "when I visit the beach I like to swim for half an hour" side-by-side. Outcomes of analysis: both are talking about swimming, butterfly is a style of swimming, (some) activities are measured by the duration for which they are performed, swimming is an activity that can be done on (some) beaches, … The outcomes can then be used to formulate the narrative. You want to start with the rough sketch so that your analysis, how you got from the initial findings to your concepts, can be retraced whenever needed.
- As another example, you have interviewed prospective users. User A says "I have trouble keeping track of all the tasks I need to do", user B says "I like to mark assignments that I get from STEM classes with a special tag". When you put them side-by-side you can come up with several abstractions. Here "task" and "assignment" are likely synonyms for the same concept. Also, (some) tasks have origins e.g. STEM classes. When you have the abstractions you then use these and the specifics to formulate the (polished) narrative.
- So, base your concept analysis on some rough sketch where you use the rough sketch and hold parts of it side-by-side to justify some decision you are taking.
- "User" is not normally a domain concept.
- When analyzing the domain rough sketch remember to stay within the domain (no system-to-be). When analyzing requirements rough sketches (e.g. as user stories) you may find terms that belong to e.g. the application functions or the implementation and not to the domain.
- Rough sketches are meant for collecting ideas in brainstorming, meeting recordings, Q&A, clippings from the literature, …

3.2 validation and verification

performance indicator	unsatisfactory	satisfactory	proficient	exemplary
Follows the guidelines on verification and validation.	Does not follow the guidelines or missing.	Follows the guidelines a little.	Follows the guidelines mostly.	Follows the guidelines exactly.

- Validation is not for getting the users' approval. It is so you can present the users, clients, stakeholders with what you understood in a way that they can give you additional information or insights.
- Say you understand that a scale is an important tool in the kitchen and you give recipe instructions in terms of weight. So, you run some specific recipes by a given prospective user, say Berta. Now, Berta says that her kitchen scale is currently out of battery and that the batteries are hard to come by. Together you come up with the idea to add amounts measured by volume.
- Say you understand that a there is a category that an artwork falls into (photo, painting, …). So, you create some specific scenarios involving these and present them to a given prospective user, say Berta. Now, Berta says that she created an artwork that users painting on photos. Together you come up with the idea that there is not always a single category for the piece of art. So, you were validating the concept that there is a single category for the method and you found that the stakeholder did not agree.
- For example, you think you understand how tasks are assigned in a project concerned with supporting students in school with their assignments, so you generate specific scenarios that you think could happen in the domain. Such a scenario could be: "Student Bruno likes to work his school homework every day. The times when he feels most productive are from 4pm-6pm. He gets an assignment from his Physics class and he estimates that assignment will take 4 hours to complete. The assignment is due in 4 days. Before starting on the Physics assig…" Then such a scenario is used as a starting point to talk to stakeholders, prospective users, … They might come up with questions such as "What if Bruno does not do homework on Sundays?" or "What if Bruno wants to change an estimate?". The dialog with the stakeholders can help you see whether you can accommodate all the things that they feel are necessary.
- "A user can post a recipe successfully" is not entirely verifiable by unit tests. Unit tests can test only the units, e.g. objects, classes. So you could test whether a specific recipe object's "publish" method is successful in ensuring that various "user" objects (Berta, Claire, David, Hector) can afterwards "view" this recipe.
- You can test e.g. completeness of your terminology by going over the

domain description, requirements prescription, software architecture, … checking whether all terms you use are defined in the terminology.

- You can validate the terminology e.g. by describing scenarios using the terms and getting feedback from stakeholders whether they would express the scenarios any different.
- You want to document why you need to validate or verify something and how this will actually be done.
- Don't just list model-checking as a testing method if you have not really thought about it and have clear and specific ideas on how you will use it.
- Also, A/B testing is used with different versions in order to see which version is better in some respect (to be defined). Often this is used in the context of UI/UX. For example, we may have a web page to check out from an online shop. We may hypothesize that putting the button to check out in the top right would make it easier for visitors to our shop to find it and check out. We might think that the time from reaching the page after adding the last item until the check out button is pressed is an indicator for how easy it is for the visitors to use the check out function. So we prepare an A and a B version and we measure, say for a month, how long it takes until the button is clicked.