SE4003 Systems Software Engineering

Final Exam

Steve Mazza

Directions: Complete all six questions below as well as the bonus question if desired. The exam is worth 15 points total and is open book/notes. Cite references properly, and state any assumptions.

Question 1 (3 pts)

You are a member of the Systems Engineering IPT for a new Unmanned Aerial System (UAS). Preliminary solution-neutral activity models that describe the UAS desired functionality have been completed. Your colleagues on the IPT are working with the results of the first round of requirements elicitation with the customer, and a system requirements document is now being developed. You suggest that software engineers be included in the development of the document, but some of your colleagues suggest that it is still too premature to ask for their involvement. Their arguments are as follows:

- “We haven’t even decided yet which functions will be implemented by hardware, and which by software, so there is nothing for them to really do yet.”
- “The software engineers are detail-oriented; they are going to want to incorporate detailed specifications that may constrain the requirements document too much at this point.”

What counterpoints might be offered to justify a position for including software engineers this early in the process? Which position are you more inclined to support and why?

The two objections appear to be somewhat internally inconsistent. On the one hand the argument is being made that there is nothing for the software engineers to do yet. And on the other hand there is an argument that software engineers will apply an inappropriate level of detail, rigor, and process to the problem, implying that there is in fact something for them to contribute.

My position, of course, is that software engineers should be involved as early as absolutely possible. Especially if decisions have yet to be made regarding portions of the system allocated to hardware vs. software, software engineers can serve as valuable subject matter experts in helping to guide those decisions.

Regarding the application of an inappropriate level of detail early on, some thoughtful guidance could include the setting of meeting agenda, statement of meeting outcomes, and clarification on the goals and standards for communicating with the stakeholders.

To omit the software engineers from the requirements planning would set back the understanding of key members of the engineering team. Furthermore, you would lose the software engineering insight into the requirements definition.

Question 2.

Refer to Figure 1 in IEEE Standard 1471 (reprinted below).

1) (1 pt) Identify the notation (*language*) and type of *diagram* that were used to create this model.

This is a simple Class Diagram showing the relationships among this class structure. In this diagram, classes are represented without members.

2) (2 pts) Stepping through the diagram one element pair and relationship at a time, write a natural language description of each relationship. Include multiplicity in your written explanation, e.g. “One system fulfills many missions.” You may capture relationships in just one direction, i.e., you do not have to also write “Many missions are fulfilled by one system.” *Use proper English grammar, as if you were explaining the diagram to someone else.*

- The System fulfills Missions
- The Environment influences the System
- The System inhabits the Environment
- The System has an Architecture
- The System has many Stakeholdes
- The Architecture is described by an Architectural Description
- The Architectural Description provides a Rationale
- The Rationale participates in the Architectural Description
- The Stakeholder identifes the Architectural Descriptions
- The Stakeholder identifies Concerns
- The Concerns are important to multiple Stakeholders
- The Concern has multiple Stekeholders
- The Concern identifies Architectural Descriptions
- The Viewpoint is used to cover Concerns
- The Viewpoint is addressed to the Stakeholders
- The Architectural Description selects Viewpoints
- The Viewpoint conforms to a View
- The Library Viewpoint may have source Viewpoint
- The Model establishes methods for the Viewpoints
- The Model participates in the Views
- The View consists of Models
- The Architectural Description is organized by Views (this is an aggregate relationship)
- The Architectural Description aggregates Models (this is an aggregate relationship)

Question3. (2 pts).

Define software coupling and cohesion. How do these design considerations impact software quality?

The reference I cite provides the following two definitions; I like them for their succinctness.

> *“Given two lines of code, A and B, they are coupled when B*

> *must change behavior only because A changed."*
>
> *"They are cohesive when a change to A allows B to change so that both add new value."*

While I said that I liked the definitions for their succinctness, I'll confess that I can only really make any sense out of the first of the two. My understanding of cohesion in software refers to the maintenance of a logical internal consistency, an encapsulation of like functionality, within a class, module, library, or other section of code.

Good software will have high cohesion and loose coupling. This minimizes the interactions among the parts of code, facilitates refactoring, and makes it easier to identify errors. Minimizing interactions is important because changes have fewer unintended consequences throughout the general body of code. Re-factoring is always good and is a by-product of the development of a greater understanding of the problem space. Errors are more easily found when the mechanics implementing a given feature are encapsulated in a given body of code (i.e., highly cohesive).

Question 4. (2 pts).

Describe some challenges for reusing software (technical, programmatic or both). Which of these are more prevalent for DoD. Why?

Reusing software is challenging for several reasons. But before we address the challenges it's important to briefly discuss some of the different types of software reuse. In order of increasing abstraction, software can be reused at the source code level, as implemented classes (or modules), as libraries linked at compile-time, as services (e.g., accessed via RESTful calls), or as abstracted patterns.

Interestingly, the difficulties in implementing software reuse diminish as the level of abstraction increases. So, in the spirit of the question, I will restrict my answer to either source code or module (class) reuse. In this context, software reuse is impeded by several factors including logistics (e.g., language compatibility, physical access, lack of communication among developers), politics (e.g., programs may not be allowed to share information), culture (e.g., lack of trust someone else's work product, fear of exposure to scrutiny, fear of losing control over source code), and technical (code may not be designed for reuse).

For the DoD, most of these impediments apply. But in my experience the largest hindrance to code reuse is most of what is developed is not built with reuse in mind. Development is not sufficiently cohesive and decoupled such that it can be used across programs. It is important to note also that the fairly recent introduction of the forge.mil site would seem to facilitate sharing of intellectual property but despite the fact that my work is not classified I am still prohibited from sharing it in source code form even with Army and other DoD colleagues. And while the Defense Research Engineering Networks (DRENs) exist, my lab's development machines are not allowed to connect to it. All of this conspires to greatly inhibit code reuse.

Question 5.

Provide short answers to all parts below.

1) (1 pt) Give an example of a safety-critical software function in a system you are familiar with (either from work or everyday life).

The Bluetooth pairing function in my wife's car only works when the vehicle is in park. This restriction prevents the driver from trying to change the pairing, a function that (for whatever reason) requires the traversal of way too many menus while operating the vehicle.

2) (1 pt) Name a potential hazard pertaining to this function, and a potential mishap that can occur if the hazard is not addressed.

A hazard associated with this function is the very real possibility of incurring an accident due to the distraction of trying to navigate the way too many menus required to affect a change in the Bluetooth pairing. To be fair, another hazard might also be that you will incur an injury to your hand as you attempt to smash the unresponsive touch screen that barely passes for an interface to the system.

3) (1 pt) What can be done to reduce the likelihood of hazardous failure modes, throughout the development of the system in which the software is embedded?

The primary safeguard is to ensure that the system functionality is locked out unless the vehicle is in park. Given that this safety check might fail due to a faulty proximity switch (or other sensor), the speedometer could be queried to see that the vehicle is not in motion. An even better method would be to also add a check with an internal inertial sensor or a GPS and have the systems vote. This way, a failure in any one system would not incorrectly prevent (or permit, in a worst case scenario) pairing.

Question 6 (2 pts).

What are the shortfalls of using lines of code as an indication of software size and complexity? Describe other available options.

Source lines of code (SLOC) is a very rudimentary method of determining software size and complexity. In fact, SLOC is merely a measure of the verbosity of the programmers and will vary greatly not only between individual software engineers solving the same problem but also between the same problem solution written in different languages.

There exists no deterministic method of assessing software size and complexity but several well-researched options are available. These include

1. COCOMO - The Constructive Cost Model (COCOMO) uses regression and is based on historical data. It is easy to use and provides consistent predictions.
2. Halstead Complexity Measures - This is related to function point analysis but improves on it by taking into account the number of distinct operators, the number of distinct operands, and the totals for each. Several measures can be calculated including calculated program length, volume, difficulty, and effort.
3. Cyclomatic Complexity - This is also referred to as the McCabe Score and is a measure of the total number of linearly independent paths of execution in the source code.

Bonus Question.

Answer one or more parts below.

1) (2 pts) What are formal methods? Describe how formal specification, model checking and theorem proving can each be used in software development. What are the benefits and limitations?

> *Formal methods apply a mathematical or analytical rigor to the software engineering process. If implemented, formal specifications, model checking, and theorem proving can each be used effectively.*
>
> *Formal specifications provide a method of describing what is to be implemented and provide an unambiguous reference for developers. Examples include Z-notation and VDM-SL.*
>
> *Model checking is a form of automated proof and is based on the running program state. Using this state information, the model is verified to ensure that it meets the system specification.*
>
> *Theorem proving is, in my opinion, the most satisfying of the formal methods. It is also often seen as the most tedious and is arguably the least implemented. It relies on applying a set of logical axioms from which specific inferences are drawn. This most often look like loop invariants and pre- and post-conditions.*
>
> *Limitations of formal methods include resource constraints and technical ability. Implementing formal methods directly addresses and benefits both validation and verification.*

2) (1 pt) Which of the following propositional formulas represents the sentence *'It will rain today in the morning or in the afternoon; if the former, the golf tournament will be rescheduled'*, where:

- p means *'It will rain today in the morning'*
- q means *'It will rain today in the afternoon'*
- r means *'The golf tournament will be rescheduled'*

1. $\neg p \to q \lor r$
2. $p \lor q \to r$
3. $(p \to q) \land (p \lor r)$
4. $p \lor \neg q \to r$
5. $(p \lor q) \land (p \to r)$

> *The correct answer is #5.*