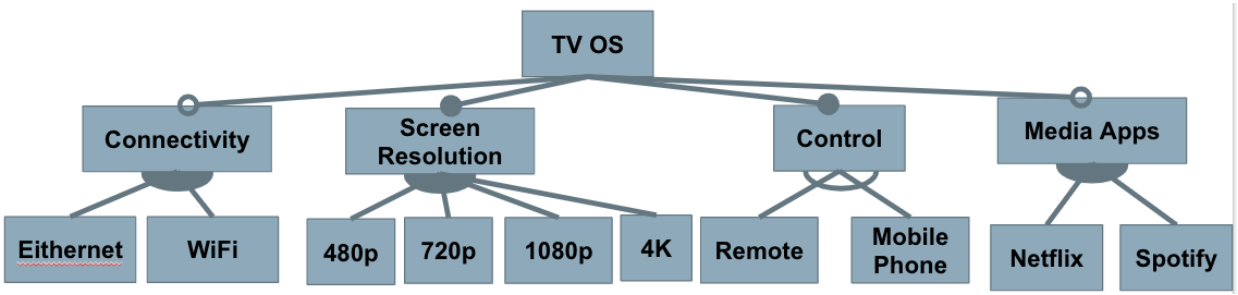
TDA 594/DIT 593 - Individual Written Assignment (Practice Version)

On all essay type questions, you will receive points based on the quality of the answer - not the quantity. You may either fill in this document or submit a separate document.

Question 1 - Domain and Application Engineering

Your company has developed a product line platform for smart TVs. Your current platform was developed according to the following feature diagram:



You receive the following requests from customers. For each request, decide if you will:

- **Extend the platform** to accommodate the request as one or more new features that can be reused in future products.
- **Add the new feature(s) to a single concrete application**, but not develop them for future reuse.
- **Decline the request** and continue to use your existing platform.

Write a short justification to explain your answer.

1. A game studio has requested you develop a new TV optimized to support 2K resolution (between 1080p and 4K) because that resolution offers a good balance between image sharpness and the speed the game can operate at.
2. Your current TVs come with a small selection of pre-installed apps (Netflix and Spotify currently). Customers have requested the addition of an app store where developers can publish their own apps and customers can download the apps that they want to use.
3. A restaurant chain has requested a special version of your TV that is locked to a single display for use in their stores for displaying menus, advertisements, and special videos. To extend your platform for their use, they ask for two features:
    a. A special app for that restaurant.
    b. A special remote containing only the power and volume buttons.

Sample Answer:

Note that many arguments can be “correct” here. In each case, you should consider the trade-offs involved. Developing for the platform is more difficult and time consuming than developing for a single product. It requires matching specialized interfaces and carefully testing feature interactions, for example, which may not be required for a single product (where the new functionality will only be used in one feature selection). If the feature would be useful in many products, it is good to develop it for the platform. If it will be used rarely, then it is more efficient to focus on one product. If few customers will care, then it is often better to decline the request entirely. Consider these factors (and others that come to mind) in justifying your answer.

1. **Extend the platform OR add to one product**: This would be relatively easy to support, so you shouldn’t ignore it entirely. However, there are arguments for either of the others.
    a. Platform: It would likely be easy to support, as you already support several other screen resolution options. Most 4K displays can handle this resolution without issues. Many gamers would appreciate the option, so there is a customer base.
    b. One Product: Only some customers will care (gamers who want to get the optimal performance), so may not be worth extra effort. There could be hardware incompatibility (not all displays can show 2K content). In the near future, games will perform well at 4K and higher resolutions, so the performance optimization only matters in the short-term.
2. **Ignore OR extend the platform:** This would require extensive effort to add, as the current platform is designed for only certain hand-selected apps. This would require the creation and management of a store, an interface for apps may need to be created (and followed by apps), apps would need to work on all TVs, etc. This could be enough work to ignore the request entirely. On the other hand, this could be of interest to a lot of customers and you could potentially earn revenue from the app developers as well. The investment may be worthwhile - IF the app store can appear on all future TVs and not just a single product.
3. **Single Product:** This is clearly a product for a single buyer. You could reasonably argue for ignoring this as well, but this could be a way to sell a lot of these TVs for relatively little work, so it’s likely worth producing a TV for this restaurant with the special app and the simplified remote. The remote could be potentially added to the platform and sold as a separate add-on or used in future models as well (e.g., to other restaurants or as a child-friendly remote).

Question 2 - Feature Modelling

You work at a company that is developing a **word processor**. You have decided to develop this as a software product line, so that you can easily provide different feature sets for different types of customers.

1. Analyze the domain and identify a set of features.
2. Model the domain with a feature diagram.

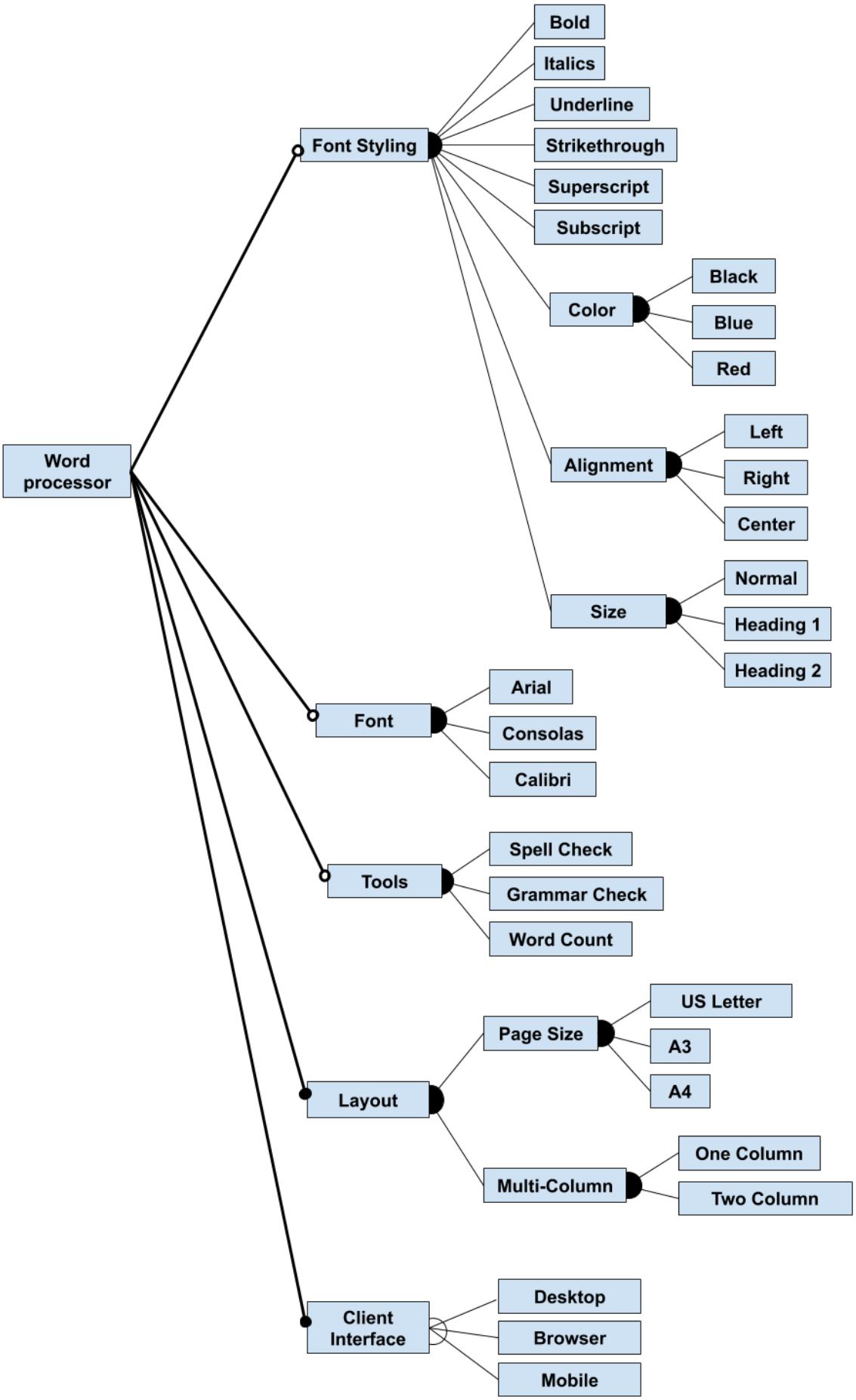
Some items to consider:

- Which features are likely to be requested by many customers?
- Which features are likely to be requested only by a few customers?
- Which features could distinguish your products from the products of your competitors in this market segment?
- Do not go crazy trying to identify all features. Try to capture an interesting set of important features (15 - 25, including all options for choices).
- Pay attention to feature dependencies and make sure you capture relevant cross-tree constraints and model structures (mandatory, optional, alternative, or).

Sample Answer:

We do not expect an exhaustive feature model, but you should capture an interesting set of features and variation points. Above, we recommended 15-25 features, but this is not an exact rule as many answers will differ. It is more important to demonstrate your knowledge. You are recommended to add some explanation and not just present the diagram. Features should be arranged in a reasonable hierarchy, with appropriate cross tree constraints.

(answer on next page)

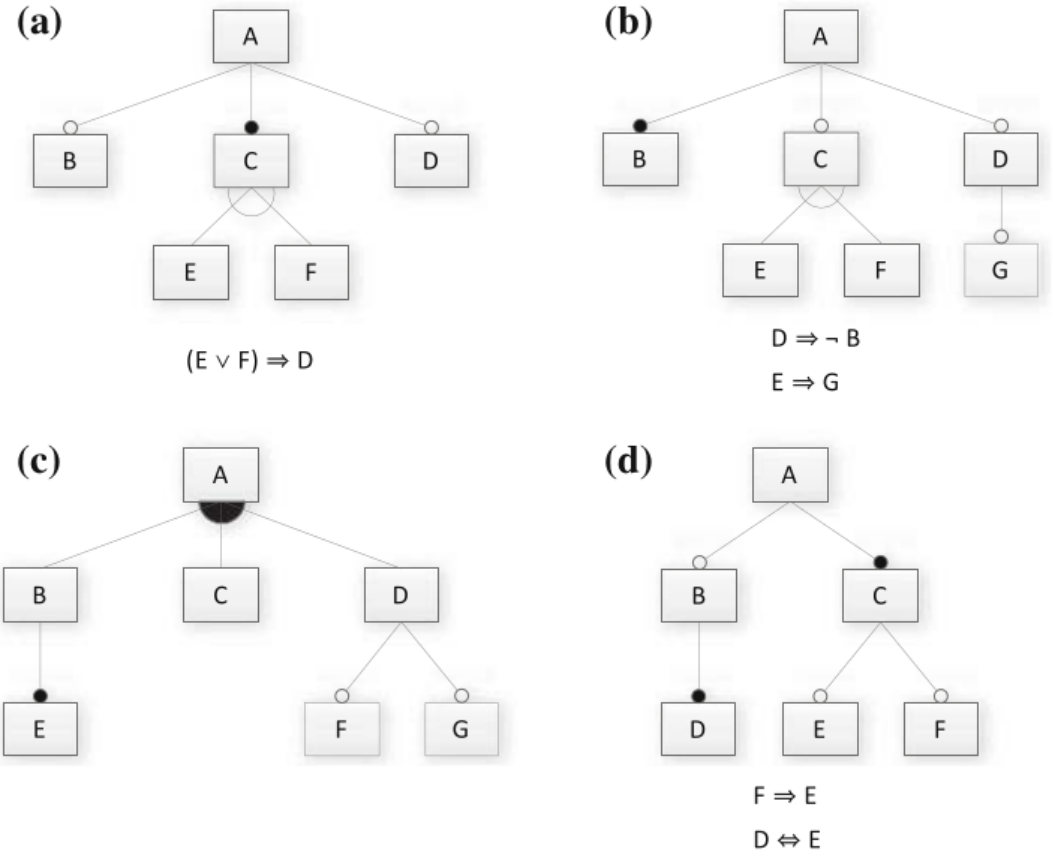

Word processor
Font Styling
Bold
Italics
Underline
Strikethrough
Superscript
Subscript
Color
Black
Blue
Red
Alignment
Left
Right
Center
Size
Normal
Heading 1
Heading 2
Font
Arial
Consolas
Calibri
Tools
Spell Check
Grammar Check
Word Count
Layout
Page Size
US Letter
A3
A4
Multi-Column
One Column
Two Column
Client Interface
Desktop
Browser
Mobile

Question 3 - Model Analysis

Recall the following transformations from feature diagram to logic (where p and f are two features, and p is the parent of f):

- mandatory(p, f) ≡ f ⇔ p
    - filled circle: if parent is chosen, child must be chosen as well
- optional(p, f) ≡ f ⇒ p
    - empty circle: if parent is chosen, child may optionally be chosen
- alternative(p, $\{f_1,...,f_n\}$) ≡ $((f_1 \vee \ldots \vee f_n) \Leftrightarrow p) \wedge_{(fi,fj)} \neg(f_i \wedge f_j)$
    - empty fan (XOR): if parent is chosen, exactly one child must be chosen
- or(p, $\{f_1,...,f_n\}$) ≡ $((f_1 \vee \ldots \vee f_n) \Leftrightarrow p)$
    - filled fan (OR): if parent is chosen, at least one child must be chosen

Consider the following Feature Models:



1. Translate the feature model into a propositional logic formula. Note that the logical expressions next to models A, B, and D are cross-tree constraints that must be incorporated as well.

2. Provide two valid and two invalid feature selections (if possible).
3. Determine whether the feature model is consistent (are there any valid configurations?). If it is not consistent, identify one reason why.

Sample Answer:

Model A:

- Formula: $A \wedge (B \Rightarrow A) \wedge (C \Leftrightarrow A) \wedge (D \Rightarrow A) \wedge ((C \Leftrightarrow (E \vee F)) \wedge \neg(E \wedge F)) \wedge ((E \vee F) \Rightarrow D))$
- Valid: A, B, C, D, F ; A, C, D, E
- Invalid: A, B, C, D, E, F ; A, B, C, E
- Is it consistent: Yes

Model B:

- Formula: $A \wedge (B \Leftrightarrow A) \wedge (C \Rightarrow A) \wedge (D \Rightarrow A) \wedge ((C \Leftrightarrow (E \vee F)) \wedge \neg(E \wedge F)) \wedge (G \Rightarrow D) \wedge (D \Rightarrow \neg B) \wedge (E \Rightarrow G)$
- Valid: A, B ; A, B, C, F
- Invalid: A, B, D, G ; A, B, C, E
- It is consistent: Yes, but D, E, and G are dead features (because B is mandatory).

Model C:

- Formula: $A \wedge ((B \vee C \vee D) \Leftrightarrow A) \wedge (E \Leftrightarrow B) \wedge (F \Rightarrow D) \wedge (G \Rightarrow D)$
- Valid: A, C ; A, B, C, D, E, F, G
- Invalid: A, B, C; A, C, E
- It is consistent: Yes (just remember that B and E need to come as a pair)

Model D:

- $A \wedge (B \Rightarrow A) \wedge (C \Leftrightarrow A) \wedge (D \Leftrightarrow B) \wedge (E \Rightarrow C) \wedge (F \Rightarrow C) \wedge (F \Rightarrow E) \wedge (D \Leftrightarrow E)$
- Valid: A, C ; A, B, C, D, E
- Invalid: A, B, C, D ; A, C, F
- It is consistent: Yes, but remember that if you have F, you need E, D, and B as well.

Question 4 - Implementation Concepts

1. Consider compile-time and load-time binding of variability decisions.
    a. Define each and note how they differ from each other.
    b. Explain potential advantages and disadvantages of each.
2. Discuss which binding times (compile, load, run-time) are suitable (or ideal, or necessary) for the following features:
    a. Multiple alternative localization features (language selection, metric versus imperial units, and so forth) for the graphical user interface of a satellite navigation system.
    b. Two alternative sorting features in a database system: a very fast and a power-efficient sorting algorithm.
    c. Two alternative features in an operating system: single-processor support and multi-processor support.
    d. Two alternative features for edge representations in a library of graph algorithms: directed and undirected edges.
    e. Multiple optional features for supported file formats in printer firmware.

Sample Answer:

1. Compile-time binding is when variability decisions are made when the code is compiled. Preprocessors are the most common form of this style of binding. Unselected features are removed from the compiled binary, resulting in a smaller, simpler compiled unit. This results in faster code, with less overhead and lighter requirements in terms of disc space and memory consumed. It may also result in more secure code. The negative side of this style is that the user loses any control of reconfiguration and can make no further customizations without requesting a new compiled binary. This is a common style in embedded development, where the additional efficiency is beneficial and reconfiguration is not needed.

   Load-time binding is when variability decisions are made when the program is executed, usually by providing flags through the command line or setting them in a configuration file. The decisions are read in and set as the program starts to run. All features remain in the compiled binary and are accessible. However, to change the configuration, the program must be re-executed with new settings. This form is often implemented using design patterns or parameters. This form of binding is less efficient than compile-time binding and can lead to slow or insecure code. However, it offers far more flexibility to the user, who is free to customize the program to their specific needs.

2. For each scenario, your actual argument is as important as your answer (if not more so):
    a. Run-time. The user may wish to change their preferences without a full reboot. Multiple users may share the same device, and you could even customize the options for each profile.

b. Load-time or run-time. We may want to switch based on current needs (performance when plugged-in, power-efficient when on battery). Run-time is likely ideal, as we could switch based on battery state. However, we also should consider the potential issues of switching algorithms during run-time. If there would be issues with elements of the system working with that data that could result from changing algorithms, we might want to wait until reloading the program.
c. Compile-time or load-time. At minimum, you will not change the hardware configuration without a reboot. You may want this as load-time, as you could conceivably change the CPU and then boot the computer again. However, this scenario, in the real world, is usually a compile-time decision. You would select the feature, then generate the right executable.
d. This could reasonably be compile or load-time. The other features in this system may depend on one graph type, so changing at run-time could cause issues (and render your current graph useless). At minimum, you would restart the program before changing. Given feature dependencies, a compile-time version could be very efficient (by removing all unselected features). Therefore, that may be the ideal version if you do not switch graph types regularly (you could even have two different lean executables - undirected and directed). Of course, a compile-time version would lose the flexibility of a load-time version.
e. Compile-time. The supported file types are unlikely to change often (if at all), as that may be determined by other hardware or software limitations. Additionally, as an embedded system, a printer will benefit from a simpler executable created by only including the supported features;.

Question 5 - Design Patterns

In class, we discussed the following design patterns - strategy, decorator, factory, facade, adapter, and template method. Choose one of these patterns and:

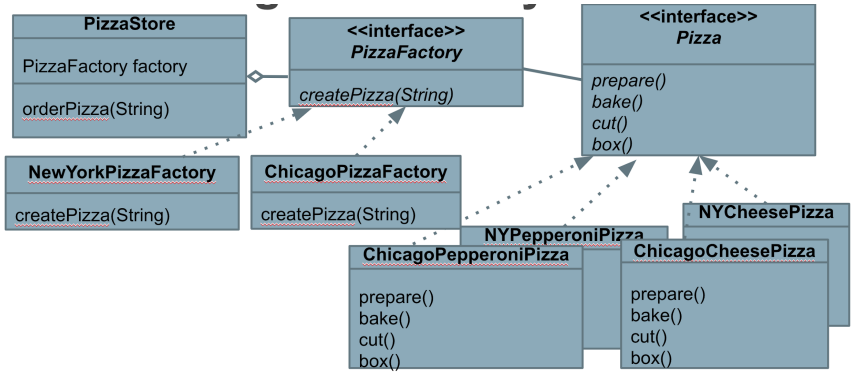
1. Describe - in your own words - the goal of the pattern and how it is applied to a system.
2. Describe how this pattern would help enable controlled variability in an effective and efficient manner.
3. Give an example (not one that we covered in class) of a concrete system that would benefit from the application of this pattern. Draw a partial class diagram to help explain this example.

Note: We are not concerned with the exact syntax of this class diagram - we just want to see the relevant portion of the class layout of the proposed system.

Sample Answer:

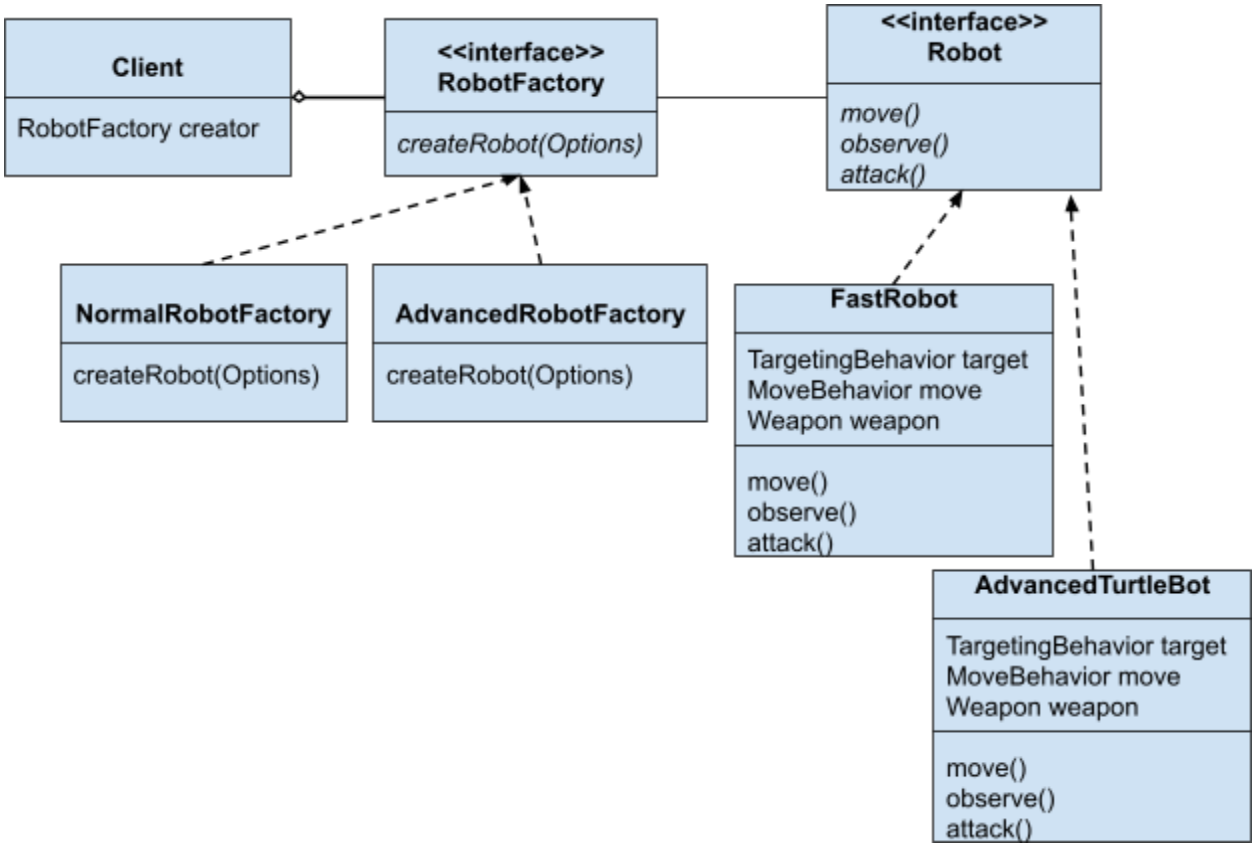
The Factory Pattern is used to perform object creation in a decoupled manner. A Factory object takes requests and returns the object (called a Product) that was requested. All Products of a particular type implement an interface representing that product, thus they all have the same data type and core set of methods. That means that client code can be developed around generic product types rather than specific products. The Factory contains all code needed to create and initialize Products. This means that new Products can be added and existing Products can be changed or removed at any time, without the client needing to change to accommodate the product changes.

Because of these features, the Factory Pattern enables the seamless initialization of related-but-varying objects without client code needing to know how the different concrete objects vary. These Products can be treated equivalently. Creation code exists in one location and only needs to evolve in that one location. This pattern enables loose coupling to concrete (varying) code and enables efficient evolution of the product portfolio.



In class, we focused on the example of a pizza shop that enables ordering of different types of pizza. These products vary, but the client code can treat them all equivalently (based on the operations defined by a Pizza interface). The Factory takes in a parameter (a string, representing the ID of the requested pizza type), and returns the correct Pizza object. Many other situations would map similarly to this exact situation.

For example, we could imagine a factory responsible for instantiating robots based on a set of options chosen, such as a movement pattern, targeting behavior, and weapon type. The Factory could take these options as parameters to a creation method, instantiate the correct Robot object, and return it to the client code. You could even have different factories to manage different subsets of robots.



Question 6 - Modularity and API Design

“Let’s Make a Deal” is a game where contestants are presented with three doors.

- One leads to a great prize, the others lead to nothing.
- Users select one door.
- Host opens one of the other doors.
- Users can then choose to open their door or the remaining unopened door.

You have been asked to implement Let’s Make a Deal as a web service. You must support:

- Creation of games.
- User selection of a door.
- The game will open one of the other doors.
- User opening of a door.
- Querying of the current state of the game and outcome (if complete) by user.
- Deletion of a game.

1. Create a REST API for this game. Determine the appropriate resources and verbs, and explain your API (what does applying a verb to that resource mean?).
2. Now, you want to extend your API into a generic, reusable API that could be used as the interface for additional games. Redesign your interface as a generic “game” interface, and explain why your new design could be reused for a different game.

Sample Answer:

Part 1:

Resource	**Verb**
/games	get – status of games server post – create new game
/games/{gid}	get – status of game (in_play, won, lost) delete – delete the game resource
/games/{gid}/doors	get – status of all doors
/games/{gid}/doors/{1..3}	get – door status {closed, selected, opened} put – update door status

Once a game is complete, only GET requests for that {gid} will be allowed.

Part 2:

Resource	**Verb**
/games	get – status of games server post – create new game
/games/{gid}	get – status of game **{results based on game}** delete – delete the game resource
/games/{gid}/items	**get – status of any in-game item** **post - create a new item (requires authentication)**
/games/{gid}/items/{iid}	**get – item status {results based on game}** **put – update item status** **delete - delete the item resource (requires authentication)**

The first two resources and their verbs could remain the same, although we could potentially allow a wider range of results for the game status. The remaining two can be made more generic.

Rather than reasoning over doors, we can reason over “items” - a notion that can include doors, laser guns, mushrooms, or any other object that a player can interact with in a game. We can add a POST action to even allow the creation and addition of new items easily, but we may want to require authentication to enable this action. For each item, we can still get a status - like with the door - but the results may be determined by the game behind the interface. We can still use PUT to interact with the item. We may also add a DELETE to allow removal of the item if it has been “used” by a player. Perhaps this also should require authentication to prevent misuse.

Question 7 - System Testing (Category Partition Method)

You have created a utility intended to find all instances of a **pattern** in a **file**.

`find(pattern,file)`

This pattern can contain spaces and quotes. For example:

`find(john,myFile)`
Finds all instances of john in the file

`find(“john smith”,myFile)`
Finds all instances of john smith in the file

`find(““john” smith”,myFile)`
Finds all instances of “john” smith in the file

Use the category-partition method to identify a pool of valid test specifications.

1. Identify the **choices** that you control when testing.
2. For each choice, identify a set of **representative values** that could lead to different outcomes of the function.
3. Impose constraints on the choices to reduce the pool of test cases.
    a. **error** constraints identify values that should result in an error no matter what other values they are paired with.
    b. **single** constraints identity values that should result in normal execution, but should be tried once because they have the potential for error or strange behavior.
    c. **if-constraints** identify values that should only be used if other choices are set to particular values (“if choice X = THIS, then choice Y = THAT)

Sample Answer:

There are two parameters to this utility: the **pattern** you are searching for and the **file** you input. Your **choices** when testing include elements you control about these two parameters that can affect the outcome when you execute the utility. Consider the different kind of outcomes you can get when you execute this service:

- All instances of the pattern that appear in an existing file that contains that pattern.
    - This is still true if there are spaces in the pattern
- Nothing, if the file is valid but the pattern is not present.
- An error if the file does not exist.
- An error if the pattern misuses quotes.
    - A space with no quoting.

- An unequal number of quotes.

Once you identify choices, identify the different abstract values that will lead to these different outcomes. Then, impose constraints to limit the number of test specifications you would form by combining those values.

- Pattern size:
  - **[error]** Empty
  - single character
  - many character
  - **[error]** longer than any line in the file
- Quoting:
  - pattern has no quotes
  - **[property quoted]** pattern has proper quotes
  - **[error]** pattern has improper quotes (only one “)
- Embedded spaces:
  - No spaces
  - **[if quoted]** One space
  - **[if quoted]** Several spaces

- File name:
  - Existing file name
  - no file with this name **[error]**
- Number of occurrence of pattern in file:
  - None
  - exactly one **[single]**
  - more than one
- Pattern occurrences on target line:
  - One
  - more than one **[single]**

Question 8 - System Testing (Combinatorial Interaction Testing)

You are designing system-level tests for a web browser with multiple configuration options. You have extracted the following features, with the following value classes for each:

Allow Content to Load	**Notify About Pop-Ups**	**Allow Cookies**	**Warn About Add-Ons**	**Warn About Attack Sites**	**Warn About Forgeries**
Allow	Yes	Allow	Yes	Yes	Yes
Restrict	No	Restrict	No	No	No
Block		Block			

The full set of possible test specifications contains 144 options.

Create a covering array of specifications that covers all **pairwise value combinations** in fewer test specifications.

(hint: start with two variables with the most values and add additional variables one at a time)

Sample Answer:

Allow Content	**Allow Cookies**	**Pop-Ups**	**Add-Ons**	**Attacks**	**Forgeries**
Allow	Allow	Yes	Yes	Yes	Yes
Allow	Restrict	No	No	Yes	No
Allow	Block	No	No	No	Yes
Restrict	Allow	Yes	No	No	No
Restrict	Restrict	Yes	-	-	Yes
Restrict	Block	No	Yes	Yes	No
Block	Allow	No	-	-	Yes
Block	Restrict	-	Yes	No	-
Block	Block	Yes	No	Yes	No

Question 9 - Automation

Metaheuristic search techniques can be divided into local and global search techniques.

1. Define what a “local” search and a “global” search is.
2. Contrast the two approaches. What are the strengths and weaknesses of each?
3. Choose one search algorithm and briefly explain how it works. State whether it is a global or local search, and explain why it belongs to that category.

Sample Answer:

1. Local search techniques formulate a solution, and attempt to improve that solution by making small changes (looking for a better solution in the “local neighborhood” - the possible solutions formed by making one small change). Global searches typically form more than one solution at a time, and freely change those solutions (moving to any spot in the search space).
2. Local searches are often very fast, easy to implement, and easy to understand conceptually. However, they depend strongly on the choice of initial guess. They can easily get stuck in local optima - where they find the best solution possible given the neighborhood, but not the best for the whole search space. This weakness can be partially overcome by allowing restarts. Global searches are harder to implement and are often slower, but have no problems with becoming stuck, as they try more than one solution at once. However, because they are slower, they may not find as good of a solution given the same time budget.
3. An example of a local search is Simulated Annealing. Initially, a solution is generated at random. Then, during each round of the search, a random neighboring solution is picked (created by making a small change to the current solution). If that neighbor is better, it becomes the new solution. If it is worse or identically good, at a certain probability, that solution will become the new solution anyways. This probability is based partially on how many rounds the search has progressed through. At earlier rounds, the search is more likely to accept a worse solution to avoid getting stuck in a local maxima. Over time, it will be more likely to reject worse solutions. This is a local search because it manipulates one solution at a time and focuses on the local neighborhood when making changes.

Question 10 - Research in Software Product Lines

Read the following research paper:

Wardah Mahmood, Daniel Strüber, Thorsten Berger, Ralf Lämmel, Mukelabai Mukelabai. *Seamless Variability Management With The Virtual Platform*. Available at https://arxiv.org/abs/2103.00437.

After reading this paper, explain (in your own words) the following:

1. What problem are the authors attempting to address?
2. Why is this problem important to address?
3. What did the authors do to address this problem?
4. What conclusions did they come to?
5. What is one thing you think could be done to extend this work in the future? Do not state one of the ideas for future work that the authors proposed themselves, but come up with your own idea.

Sample Answer:

1. Two common strategies for development of customizable software are clone & own and platforms (software product lines). In the former, a developer creates a new branch or clone of the code repository, makes changes, and maintains their separate branch. This is inexpensive, flexible, enables developer independence, and allows fast innovation. However, it does not scale as the number of variants increases and creates a large maintenance burden, as it is difficult to incorporate new changes to the original system. In platform development, a set of reusable assets are developed and composed into new concrete products based on a feature selection. This strategy is scalable, but requires substantial up-front investment. Many companies start with clone & own, and migrate to product line development. This transition is risky and expensive. The authors seek a way to make the transition easier and less risky, enabling developers to more easily transition to a product line development approach.
2. This problem is important to address because of the cost and effort required to develop a software product line. Many developers *cannot* start with a platform, and must transition from a single product to a product line. This transition is risky, and could result in poor products, cancelled products, failed contracts, and other problems if it fails or is conducted poorly. A simple and incremental way to transition to a platform is very important to reduce this risk.
3. The authors propose that the two approaches can be bridges using a virtual platform. This is a framework that supports both clone & own and platform development. Based on the number of variants, organizations can decide to use only a subset of all the variability concepts typically required for a full platform, starting with clone & own and incrementally scaling the development. They facilitate this by recording relevant metadata (e.g., features, feature locations, and clone traces) automatically for the developers. The

virtual platform exploits this metadata for the transition, providing operators that developers can use to handle variability. For example, operators allow developers to add, change, or remove assets, to change the location of an attribute, to clone an asset, to link an asset to a feature, to merge code from one asset into its clone, to create features, to link a feature model to an asset, to remove or move features, to mark a feature as optional, to clone a feature, or to merge the code of one feature into another. These operators maintain traceability between project clones and the original code, ease management of clones, and allow the transition from a clone & own model to a platform over time using this metadata and the operators.

4. The authors evaluate their virtual platform in terms of costs and benefits. Costs are in terms of additional developer effort using the virtual platform. Costs arise from maintaining features, and dealing with omissions during feature maintenance. Benefits relate to saved costs from feature location and clone detection/propagation. Most costs relate to feature creation and adding an asset to a feature, with the cost-per-invocation being low each time this is done. Much larger effort savings are attained from avoiding the need for manual clone detection. Their platform is able to save developers significant effort that would be required through a pure clone & own approach.
5. The existing metadata and, potentially, other data that could be recorded could be used to train a prediction model that can automatically propose potential feature mappings to the user by comparing different clones. Alternatively, this data could be used to suggest operators to the developer when it appears that they could make use of one. For example, if a user is moving code manually, the platform might suggest that an asset is being moved or changed, and suggest the use of the operator to accomplish the task.