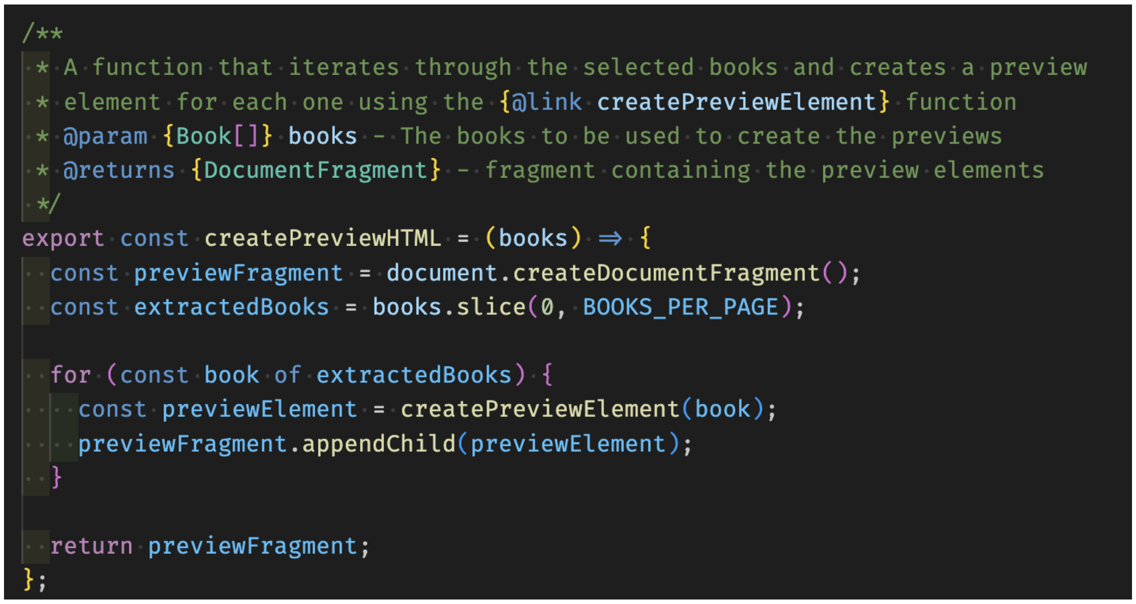
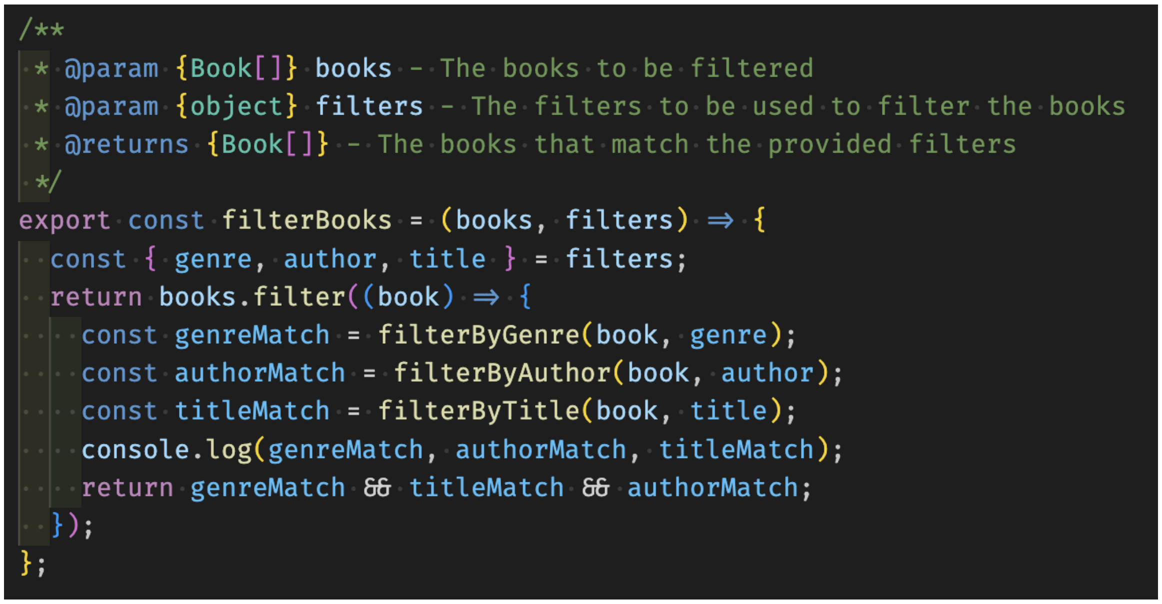
DWA_07.4 Knowledge Check_DWA7

1. Which were the three best abstractions, and why?



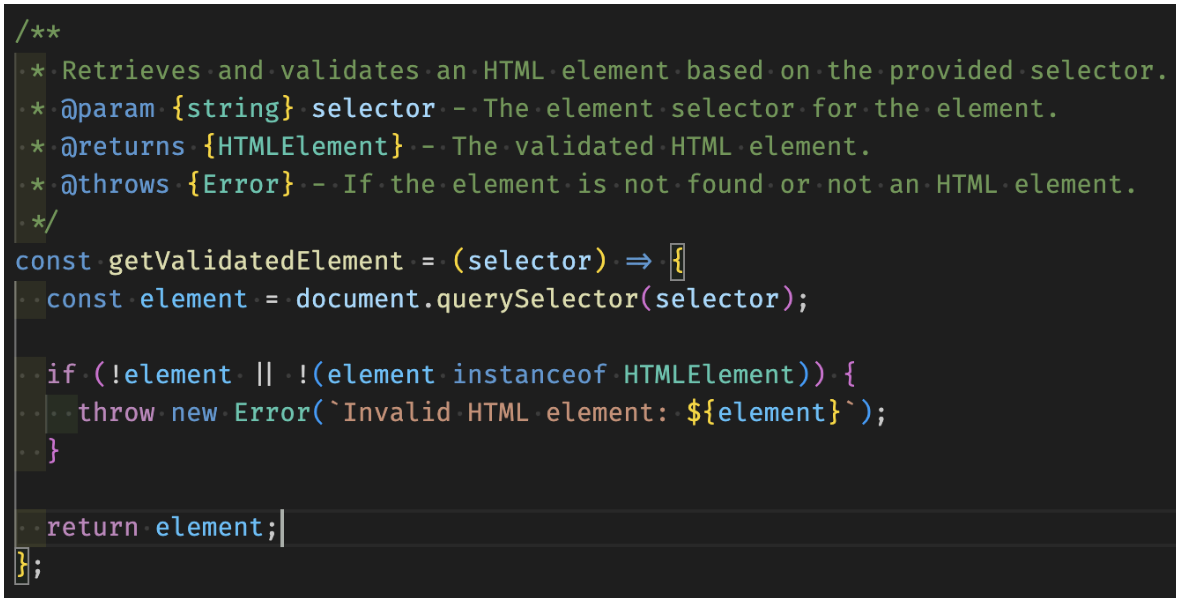
```js
/**
 * A function that iterates through the selected books and creates a preview
 * element for each one using the {@link createPreviewElement} function
 * @param {Book[]} books - The books to be used to create the previews
 * @returns {DocumentFragment} - fragment containing the preview elements
 */
export const createPreviewHTML = (books) => {
  const previewFragment = document.createDocumentFragment();
  const extractedBooks = books.slice(0, BOOKS_PER_PAGE);

  for (const book of extractedBooks) {
    const previewElement = createPreviewElement(book);
    previewFragment.appendChild(previewElement);
  }

  return previewFragment;
};
```

It abstracts the process of creating preview elements from books into a separate function. If in future you need to change how previews are created the function can be modified without altering existing code. The function is also reused a lot throughout the codebase.



```js
/**
 * @param {Book[]} books - The books to be filtered
 * @param {object} filters - The filters to be used to filter the books
 * @returns {Book[]} - The books that match the provided filters
 */
export const filterBooks = (books, filters) => {
  const { genre, author, title } = filters;
  return books.filter((book) => {
    const genreMatch = filterByGenre(book, genre);
    const authorMatch = filterByAuthor(book, author);
    const titleMatch = filterByTitle(book, title);
    console.log(genreMatch, authorMatch, titleMatch);
    return genreMatch && titleMatch && authorMatch;
  });
};
```

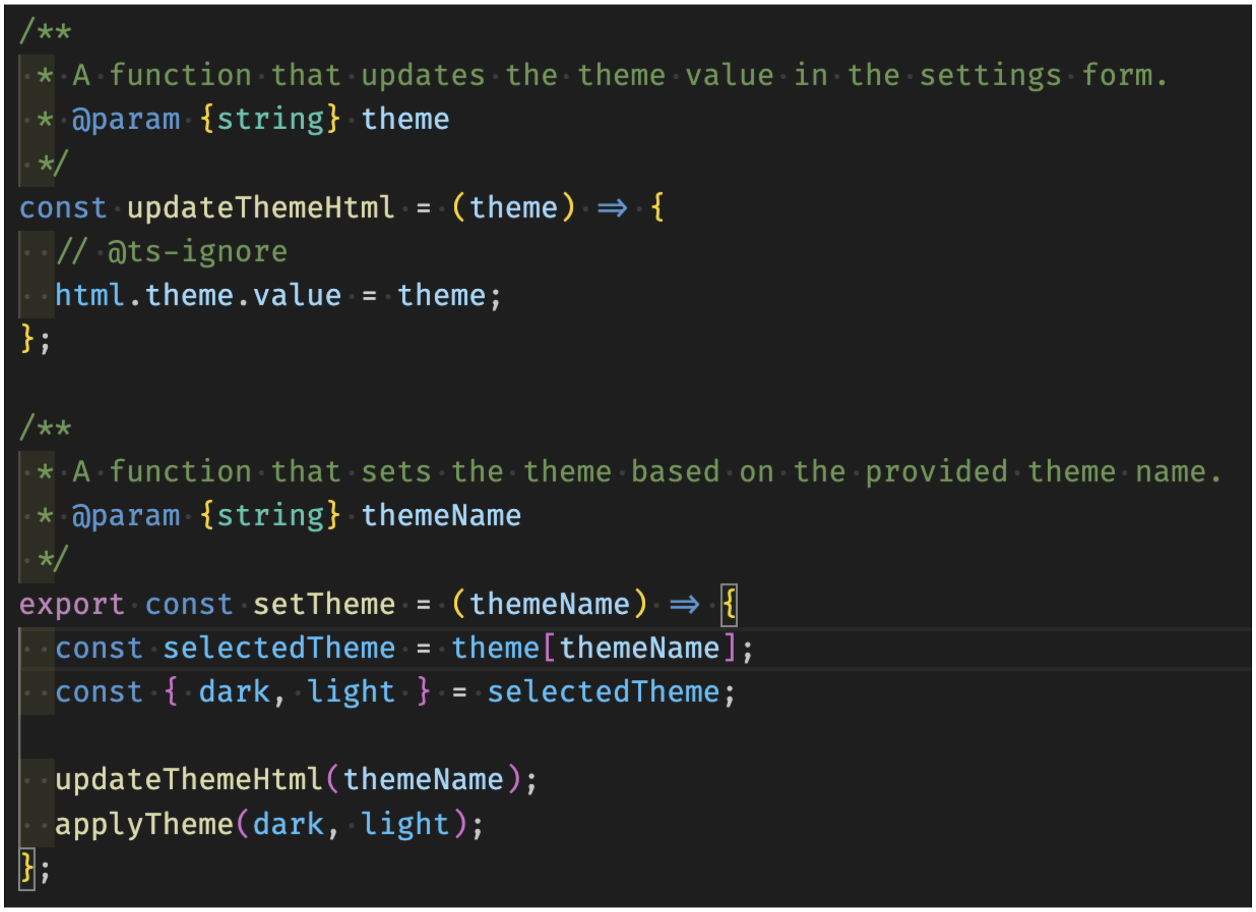
It depends on high-level abstractions rather than low-level details. It doesn't directly interact with DOM elements or handle low-level operations.



```javascript
/**
 * Retrieves and validates an HTML element based on the provided selector.
 * @param {string} selector - The element selector for the element.
 * @returns {HTMLElement} - The validated HTML element.
 * @throws {Error} - If the element is not found or not an HTML element.
 */
const getValidatedElement = (selector) => {
  const element = document.querySelector(selector);

  if (!element || !(element instanceof HTMLElement)) {
    throw new Error(`Invalid HTML element: ${element}`);
  }

  return element;
};
```

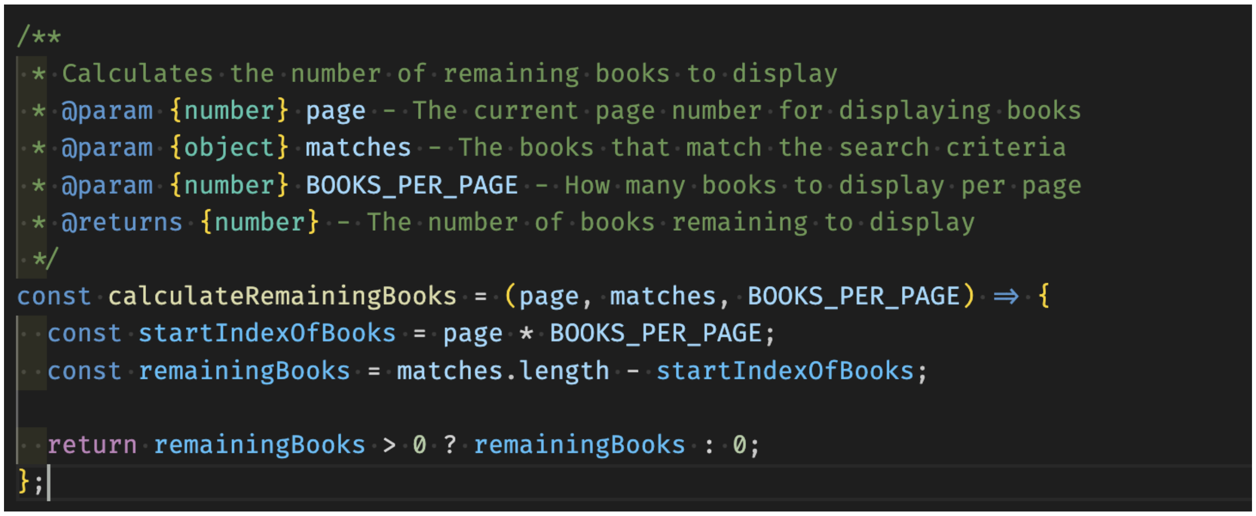
It retrieves and validates HTML elements based on a given selector, following SRP, encapsulating logic, handling errors, and promoting reusability and clarity.

2. Which were the three worst abstractions, and why?

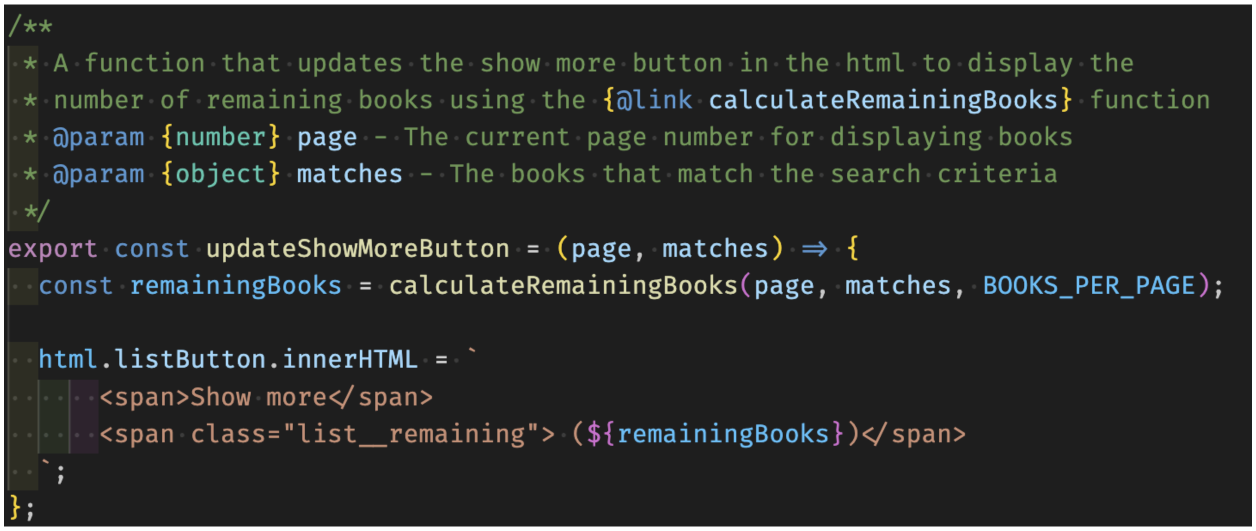


```js
/**
 * A function that updates the theme value in the settings form.
 * @param {string} theme
 */
const updateThemeHtml = (theme) => {
  // @ts-ignore
  html.theme.value = theme;
};

/**
 * A function that sets the theme based on the provided theme name.
 * @param {string} themeName
 */
export const setTheme = (themeName) => {
  const selectedTheme = theme[themeName];
  const { dark, light } = selectedTheme;

  updateThemeHtml(themeName);
  applyTheme(dark, light);
};
```

The separate function, updateThemeHtml, adds unnecessary complexity. Integrating it directly into setTheme streamlines code, improving cohesion and reducing dependencies on specific HTML elements.



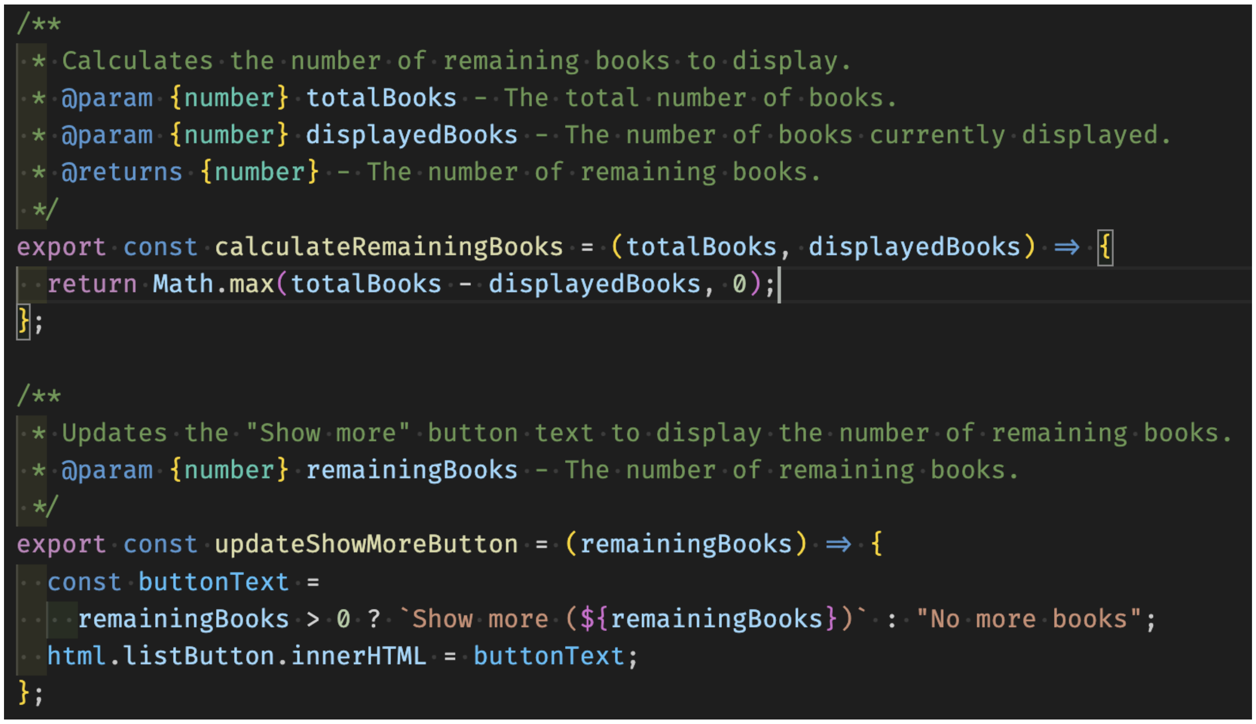
```js
/**
 * Calculates the number of remaining books to display
 * @param {number} page - The current page number for displaying books
 * @param {object} matches - The books that match the search criteria
 * @param {number} BOOKS_PER_PAGE - How many books to display per page
 * @returns {number} - The number of books remaining to display
 */
const calculateRemainingBooks = (page, matches, BOOKS_PER_PAGE) => {
  const startIndexOfBooks = page * BOOKS_PER_PAGE;
  const remainingBooks = matches.length - startIndexOfBooks;

  return remainingBooks > 0 ? remainingBooks : 0;
};
```



```
/**
 * A function that updates the show more button in the html to display the
 * number of remaining books using the {@link calculateRemainingBooks} function
 * @param {number} page - The current page number for displaying books
 * @param {object} matches - The books that match the search criteria
 */
export const updateShowMoreButton = (page, matches) => {
  const remainingBooks = calculateRemainingBooks(page, matches, BOOKS_PER_PAGE);

  html.listButton.innerHTML = `
      <span>Show more</span>
      <span class="list__remaining"> (${remainingBooks})</span>
  `;
};
```

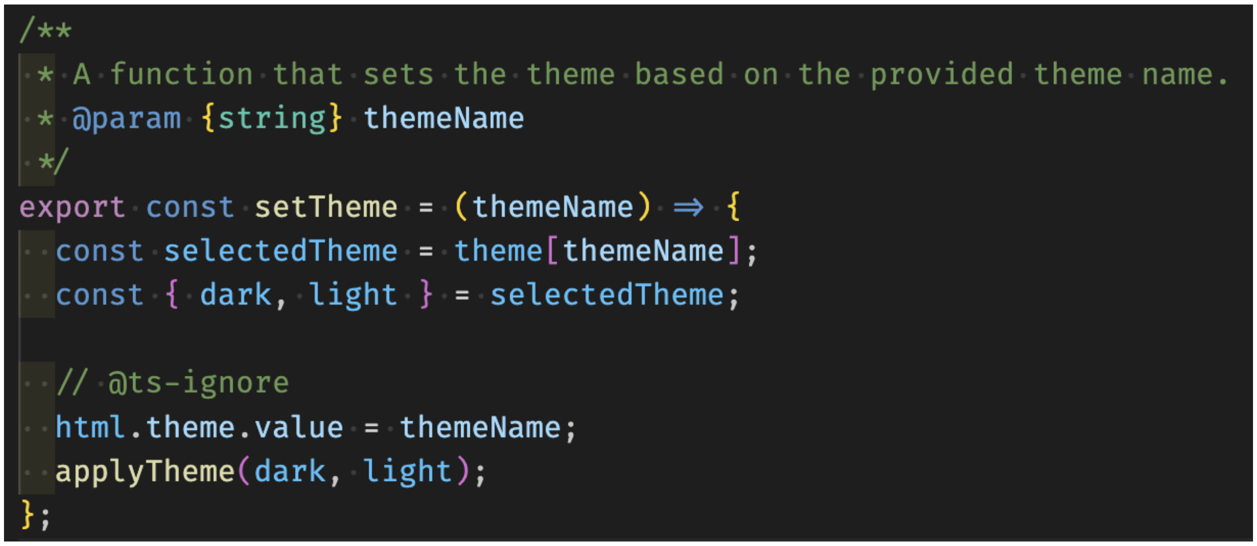
Functions lack cohesion, handle multiple concerns, violating SRP, leading to code complexity and maintainability issues.

3. How can the three worst abstractions be improved via SOLID principles.



```
/**
 * Calculates the number of remaining books to display.
 * @param {number} totalBooks - The total number of books.
 * @param {number} displayedBooks - The number of books currently displayed.
 * @returns {number} - The number of remaining books.
 */
export const calculateRemainingBooks = (totalBooks, displayedBooks) ⇒ {
  return Math.max(totalBooks - displayedBooks, 0);
};

/**
 * Updates the "Show more" button text to display the number of remaining books.
 * @param {number} remainingBooks - The number of remaining books.
 */
export const updateShowMoreButton = (remainingBooks) ⇒ {
  const buttonText =
    remainingBooks > 0 ? `Show more (${remainingBooks})` : "No more books";
  html.listButton.innerHTML = buttonText;
};
```

The fixed abstraction separates concerns better. calculateRemainingBooks works out the remaining books, while updateShowMoreButton updates the button text accordingly, improving maintainability and reducing coupling.



```
/**
 * A function that sets the theme based on the provided theme name.
 * @param {string} themeName
 */
export const setTheme = (themeName) ⇒ {
  const selectedTheme = theme[themeName];
  const { dark, light } = selectedTheme;

  // @ts-ignore
  html.theme.value = themeName;
  applyTheme(dark, light);
};
```

The fixed abstraction adheres better to the Single Responsibility Principle by combining theme setting and HTML update in one function, setTheme, resulting in clearer code organization and reduced coupling.