Better Code Hub[BETA]

SIGN OUT

Your repositories › Your results

Compliance
6
of 10

Miridinia/eclipse-color-theme

Last analysis: 3 minutes ago

Write Short Units of Code ✗

Guideline

› Small units are easier to understand, reuse, and test.

› When writing new units, don't let them grow above 15 lines of code.

› When a unit grows beyond 15 lines of code, you need to shorten it by splitting it in smaller units of no longer than 15 lines of code.

› The list on the right side contains the top 30 of units that violate this guideline, sorted by severity. The severity is indicated by the colors of the checkboxes.

› Further reading: Chapter 2 of Building Maintainable Software

Refactoring candidates

✓ Unit

- 🟥 ColorThemeManager.parseTheme(InputStream,boolean)
- 🟧 ColorThemePreferencePage.createContents(Composite)
- 🟧 ColorThemePreferencePage.performOk()
- 🟧 ColorThemeManager.$constructor()
- 🟧 ColorThemePreferencePage.contributeButtons(Composi...
- 🟨 ColorThemePreferencePage.updateDetails(ColorTheme)
- 🟨 ColorThemeManager.readStockThemes(Map)
- 🟨 Mapping.putPreferences(IEclipsePreferences,ColorTh...
- 🟨 ColorThemeManager.readImportedThemes(Map)

🟩 at most 15 lines of code	🟧 more than 30 lines of code
🟨 more than 15 lines of code	🟥 more than 60 lines of code

Write Simple Units of Code ✓

Guideline

› Keeping the number of branch points (if, for, while, etc.) low makes units easier to modify and test.

› Try to keep the number of branch points in a unit below 5.

You can reduce complexity by extracting

Refactoring candidates

✓ Unit

- 🟧 ColorThemeManager.parseTheme(InputStream,boolean)
- 🟨 ColorThemePreferencePage.performOk()
- 🟨 ColorThemePreferencePage.updateDetails(ColorTheme)
- 🟨 Mapping.putPreferences(IEclipsePreferences,ColorThem...

- sub-branches to separate units of no more than 5 branch points.
- The list on the right side contains the top 30 of units that violate this guideline, sorted by severity. The severity is indicated by the colors of the checkboxes.
- Further reading: Chapter 3 of Building Maintainable Software

🟨 `GenericMapper.map(Map,Map)`

🟨 `Mapping.putPreferences(IEclipsePreferences,ColorThem...`

🟩 at most 5 branch points	🟧 more than 10 branch points
🟨 more than 5 branch points	🟥 more than 25 branch points

Write Code Once ✗

Guideline

- When code is copied, bugs need to be fixed in multiple places. This is both inefficient and error-prone.
- Avoid duplication by never copy/pasting blocks of code.
- Reduce duplication by extracting shared code, either to a new unit or to a superclass.
- The list on the right side contains the top 30 sets of modules (grouped by highlighting) which contain the same duplicated code block.
- Further reading: Chapter 4 of Building Maintainable Software

Refactoring candidates

✓ Module

🟥 `DltkEditorMapper.java`

🟥 `FlashEditorMapper.java`

🟥 `StatetEditorMapper.java`

🟥 `FlashEditorMapper.java`

🟥 `PerlEditorMapper.java`

🟥 `DltkEditorMapper.java`

🟥 `FlashEditorMapper.java`

🟥 `WebEditorMapper.java`

🟥 `StatetEditorMapper.java`

🟩 non-duplicated code	🟥 duplicated code

Keep Unit Interfaces Small ✓

Guideline

- Keeping the number of parameters low makes units easier to understand and reuse.
- Limit the number of parameters per unit to at most 4.
- The number of parameters can be reduced by grouping related parameters into objects.
- The list on the right side contains the top 30 of units that violate this guideline, sorted by severity. The severity is

Refactoring candidates

✓ Unit

🟨 `ColorThemeManager.applyDefault(Map,String,String)` indicated by the colors of the checkboxes.

› Further reading: Chapter 5 of Building Maintainable Software

🟩 at most 2 parameters	🟧 more than 4 parameters
🟨 more than 2 parameters	🟥 more than 6 parameters

Separate Concerns in Modules ✓

Guideline

› Keep the codebase loosely coupled, as it makes it easier to minimize the consequences of changes.

› Identify and extract responsibilities of large modules to separate modules and hide implementation details behind interfaces.

› Strive to get modules to have no more than 10 incoming calls.

› The list on the right side contains the top 30 of modules that violate this guideline, sorted by severity. The severity is indicated by the colors of the checkboxes.

› Further reading: Chapter 6 of Building Maintainable Software

Refactoring candidates

✓ Module

🟥 `ColorThemeSetting.java`

🟧 `ColorThemeMapping.java`

🟨 `ColorTheme.java`

🟨 `Color.java`

🟨 `ParsedTheme.java`

🟩 at most 10 incoming calls	🟧 more than 20 incoming calls
🟨 more than 10 incoming calls	🟥 more than 50 incoming calls

Couple Architecture Components Loosely ✓

Guideline

› Having loose coupling between top-level components makes it easier to maintain components in isolation.

› Do this by minimising the amount of interface code; that is, code in modules that are both called from and call modules of other components (throughput), and code in modules that are called from modules of other components (incoming).

› You can hide a component's implementation details through various means, e.g. using the "abstract factory" design pattern.

› The list on the right side contains the top 30 of modules that violate this guideline,

Refactoring candidates

✓ Module

🟩 hidden code	🟥 interface code

starting with the modules that contain throughput code.

- Further reading: Chapter 7 of Building Maintainable Software

Keep Architecture Components Balanced ✗

Guideline

- Balancing the number and relative size of components makes it easier to locate code.
- Organize source code in a way that the number of components is between 2 and 12, and ensure the components are of approximately equal size (keep component size uniformity less than 0.71).
- Organising components based on functionality makes it easier to divide your code into components.
- Further reading: Chapter 8 of Building Maintainable Software

Components overview

0
Component size uniformity

1
Components

com.github.eclipsecolortheme

Component size (lines of code)

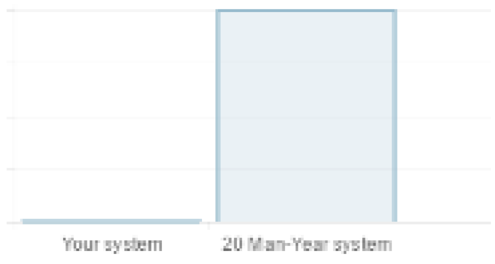
Keep Your Codebase Small ✓

Guideline

- Keeping your codebase small improves maintainability, as it's less work to make structural changes in a smaller codebase.
- Avoid codebase growth by actively reducing system size.
- Refactor existing code to achieve the same functionality using less volume, and prefer libraries and frameworks over "homegrown" implementations of standard functionality.
- Strive to keep volume below 20 Man-years.
- Further reading: Chapter 9 of Building Maintainable Software

Volume overview

2
Man-months



System volume (man-years)

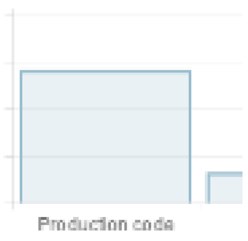
Automate Tests ✗

Guideline

Testing overview

- Automating tests for your codebase makes development more predictable and less risky.
- Add tests for existing code every time you change it.
- For small systems (less than 1,000 lines of code), you should have at least some test code and one assertion (currently only checked for Java and C# systems).
- For medium systems (less than 10,000 lines of code), the total lines of test code should be at least 50% of the total lines of production code, and the assert density (percentage of lines of test code containing assertions) should be at least 1% (currently only checked for Java and C# systems).
- For large systems (more than 10,000 lines of code), the total lines of test code should be at least 50% of the total lines of production code, and the assert density should be at least 5% (currently only checked for Java and C# systems).
- Further reading: Chapter 10 of Building Maintainable Software

1,404 Lines of production code	**23%** Test code percentage
319 Lines of test code	**14%** Assert density



Lines of code

Write Clean Code ✓

Guideline

- Clean code is more maintainable.
- Proactively search and remove code smells.
- Remove useless comments, commented code blocks, and dead code. Refactor poorly handled exceptions, magic constants, and poorly names units or variables.
- The list on the right side contains a selection of violations for this guideline.
- Further reading: Chapter 11 of Building Maintainable Software

Refactoring candidates

✓ Module

🟥 `ColorThemeManager.java`

🟥 `SqlEditorMapper.java`

🟥 `ColorThemePreferencePage.java`

🟥 `ColorThemePreferencePage.java`

| ✓

🟩 clean code 🟥 code smell