Designite - A Software Design Quality Assessment Tool

Tushar Sharma
Athens University of Economics and Business
Athens, Greece
tushar@aueb.gr

Pratibha Mishra
Designite
Bangalore, India
pratibham0709@gmail.com

Rohit Tiwari
Designite
Bangalore, India
trohit9217@gmail.com

ABSTRACT

Poor design quality and huge technical debt are common issues perceived in real-life software projects. Design smells are indicators of poor design quality and the volume of design smells found could be treated as the design debt of the software system. The existing smell detection tools focus largely on implementation smells and do not reveal a comprehensive set of smells that arise at design level. In this paper, we present *Designite* - a software design quality assessment tool. It not only supports comprehensive design smells detection but also provides a detailed metrics analysis. Further, it offers various features to help identify issues contributing to design debt and improve the design quality of the analyzed software system.

CCS Concepts

•Software and its engineering → Software maintenance tools;

Keywords

Design smells, refactoring, DSM, technical debt, design debt.

1. INTRODUCTION

Software design is an inherently complex activity that requires sound knowledge of design principles and more importantly their skilful application. A study of five organizations [6] reported the number of software defects that can be traced back to errors in software design as high as 64%. This statistic highlights the importance of software design in software development. Despite this, we have witnessed real-life projects suffering from poor design quality and huge technical debt [13]. Such issues significantly reduce maintainability of the software.

"Design smells are certain structures in the design that indicate violation of fundamental design principles and negatively impact design quality" [15]. Design smells are indicators of poor design quality and the volume of design smells found could be treated as the design debt (a component of the overall technical debt) of a software system. Therefore, identifying design smells and refactoring them could lead us to a better design quality.

Various tools are available to compute source code metrics and even smells. Most of them use metrics as a primary mean to identify smells. Although, some of the design smells can be inferred from the metrics, a deeper source code analysis is required to identify many design smells. Further, the existing smell detection tools focus on implementation smells largely and do not reveal a comprehensive set of design smells. Therefore, a tool that explicitly detects design smells and provides mechanisms to assess design quality is required but missing.

Designite [1] is a software design quality assessment tool. It offers a comprehensive support to detect a wide variety of design smells and computes various metrics at different granularities. It provides a simple and interactive implementation of DSM (Dependency Structure Matrix) to help analyze the dependencies among the source code entities. Additionally, the tool also supports features such as hotspot analysis, integration with external tools via its console application, code-clone detection, and export the inferred results to MS excel sheets. With all these features, Designite helps us assess the design quality of a software system, identifies design debt instances, and enables improvements in design agility of the software.

Rest of the paper is organized as follows: Section 2 discusses the related work, Section 3 describes the features as well as some implementation details of Designite, Section 4 presents evaluation results carried out on three versions of three open-source projects, and Section 5 concludes and explores possible future directions.

2. RELATED WORK

Although, a great amount of work has been done to catalog smells at different abstraction levels [2], the present set of available tools lack the focus on design quality. The rest of the section discusses related work with respect to two commonly used dimensions (viz. metrics and design smells) to perceive design quality.

Software metrics reveal characteristics of a software and could provide insights towards code quality of the software. Many metrics (such as LCOM, CBO, Fan-in, and Fan-out) help us understand the design aspects of the software. Currently, we can avail services of many metrics tools to compute these metrics. However, metrics tools prove insufficient when one wants to perform a detailed design assessment for a given

Bridge'16, May 17 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4153-0/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2896935.2896938

software due to the following reasons:

- Metrics do not cover many aspects related to design and hence many design smells cannot be identified just by using commonly known metrics (for example, Rebellious Hierarchy [15]).
- Metrics require another level of interpretation to design smells. This interpretation is an extra step, which is directly not supported by the existing metrics tools.
- Metrics tools generate metrics data for a software; however, they do not offer mechanisms to analyze the generated data for the software holistically. For instance, revealing metric violations and threshold customization is also often not supported directly.

There are a few tools available that detect code smells. For example, NDepend [8] (for C#) detects many smells apart from computing metrics. For Java, JDeodorant [5] detects a few smells. Similarly, Resharper [12] provides many rules to improve code quality. However, such tools do not classify the smells based on the abstraction levels and more importantly, they do not support enough smells or rules to assess design quality sufficiently. For instance, JDeodorant only support detection of 5 smells and not all the supported smells can be classified as design smells. Therefore, a tool focusing on design quality with comprehensive support for design smells is required and Designite is an attempt towards the goal.

3. DESIGNITE

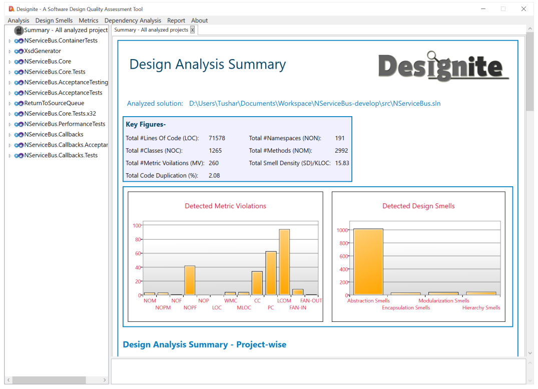
Designite takes source code written in C# as input, analyzes it, and presents design assessment information interactively. Designite uses NRefactory [9] to parse C# code and prepares Abstract Syntax Tree (AST). Designite accesses the AST and prepares a simple hierarchical meta-model. The meta-model contains objects of projects containing information about analyzed projects. In turn, each project-object contains the objects of namespaces implemented in the project. Similarly, each namespace-object contains objects of types that are part of the namespace and so on. The meta-model captures the required source code information, which is used by Designite, for instance, to infer design smells and compute metrics. Designite also carries out a code-clone detection analysis that identifies sets of code-clones present in the software.

An analysis can be initiated by either choosing a .NET solution file or providing a batch file that contains path of a C# project per line. After the analysis, Designite presents a summary of the analysis as shown in Figure 1.

The rest of the section discusses key features of the tool along with their implementation details briefly.

3.1 Design Smell Detection

Once the source code meta-model is prepared, Designite infers design smells from the meta-model. The design smells catalog suggested by Girish et al. [15] is considered the most comprehensive catalog for structural design smells. The catalog consists of 25 design smells classified based on the design principle (viz. Abstraction, Encapsulation, Modularization, and Hierarchy) they violate. Currently, Designite detects 19 design smells from the catalog. Table 2 lists the design smells supported by Designite and corresponding definitions.


Designite - A Software Design Quality Assessment Tool
Analysis
Design Smells
Metrics
Dependency Analysis
Report
About
Summary - All analyzed project
NServiceBus.ContainerTests
XsdGenerator
NServiceBus.Core
NServiceBus.Core.Tests
NServiceBus.AcceptanceTesting
NServiceBus.AcceptanceTests
ReturnToSourceQueue
NServiceBus.Core.Tests.x32
NServiceBus.PerformanceTests
NServiceBus.Callbacks
NServiceBus.Callbacks.Acceptar
NServiceBus.Callbacks.Tests
Summary - All analyzed projects
Design Analysis Summary
Designite
Detected Metric Violations
NOM
NOPM
NOF
NOPF
NOP
LOC
WMC
MLOC
CC
PC
LCOM
FAN-IN
FAN-OUT
Detected Design Smells
Abstraction Smells
Encapsulation Smells
Modularization Smells
Hierarchy Smells
Design Analysis Summary - Project-wise

Analyzed solution: D:\Users\Tushar\Documents\Workspace\NServiceBus-develop\src\NServiceBus.sln

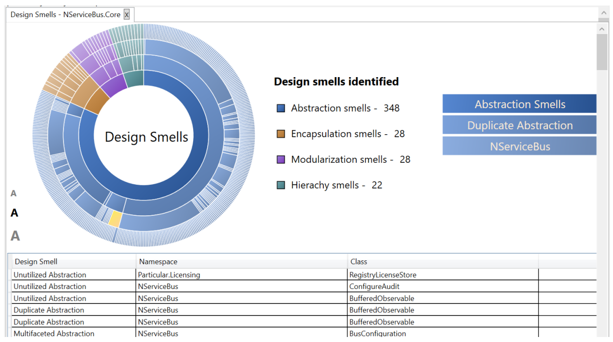
Key Figures-

Total #Lines Of Code (LOC):	71578	Total #Namespaces (NON):	191
Total #Classes (NOC):	1265	Total #Methods (NOM):	2992
Total #Metric Voilations (MV):	260	Total Smell Density (SD)/KLOC:	15.83
Total Code Duplication (%):	2.08		

Figure 1: Analysis summary produced by Designite

Some of the smells are relatively easier to detect. For instance, Multifaceted Abstraction [15] is detected using metric LCOM (Lack of Cohesion between Methods). Similarly, metrics NOPM (Number Of Public Methods), NOF (Number Of Fields), and WMC (Weighted Methods per Class) are used to detect Insufficient Modularization [15]. However, many design smells such as Rebellious Hierarchy [15] and Cyclically-dependent Modularization [15] require a deeper source code analysis beyond the standard software metrics. It is important to note that there are a few design smells, for example, Missing Encapsulation [15] and Speculative Hierarchy [15], that are extremely difficult to detect if not impossible using static analysis.

The detected design smells are presented using a sunburst diagram (see Figure 2). The sunburst diagram is an interactive, beautiful, yet effective way to navigate and filter the detected design smells. Designite's sunburst diagram has four rings. Each ring represents a dimension associated with detected design smells. The first ring shows the distribution of smells based on the violated principle. The second ring represents the distribution of specific design smells detected. Similarly, the third and fourth rings represent the namespace and the class respectively in which the specific design smell has been detected. If one clicks on the abstraction fragment, only the abstraction smells are shown in the grid below. Similarly, if one wish to see only Deficient Encapsulation smell instances, she can click on the relevant fragment to achieve the same. Further, the description as well as the cause of a smell can be seen in the document pan upon selecting a row in the grid.



- 🟦 Abstraction smells - 348
- 🟫 Encapsulation smells - 28
- 🟪 Modularization smells - 28
- 🟩 Hierachy smells - 22

Design Smell	Namespace	Class
Unutilized Abstraction	Particular.Licensing	RegistryLicenseStore
Unutilized Abstraction	NServiceBus	ConfigureAudit
Unutilized Abstraction	NServiceBus	BufferedObservable
Duplicate Abstraction	NServiceBus	BufferedObservable
Duplicate Abstraction	NServiceBus	BufferedObservable
Multifaceted Abstraction	NServiceBus	BusConfiguration

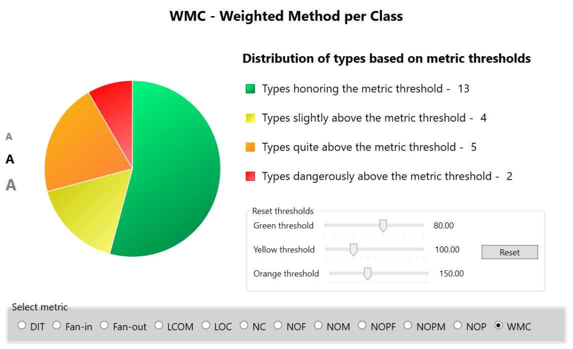
Figure 2: Sunburst representation of design smells

Table 1: Design Smells Supported by Designite and Their Corresponding Definitions

Design Smell	Definition
Unnecessary Abstraction	This smell occurs when an abstraction which is actually not needed (and thus could have been avoided) gets introduced in a software design.
Imperative Abstraction	This smell arises when an operation is turned into a class.
Multifaceted Abstraction	This smell arises when an abstraction has more than one responsibility assigned to it.
Unutilized Abstraction	This smell arises when an abstraction is left unused (either not directly used or not reachable).
Duplicate Abstraction	This smell arises when two or more abstractions have identical names and/or identical implementation.
Deficient Encapsulation	This smell occurs when the declared accessibility of one or more members of an abstraction is more permissive than actually required.
Unexloited Encapsulation	This smell arises when client code uses explicit type checks instead of exploiting the variation in types already encapsulated within a hierarchy.
Broken Modularization	This smell arises when data and/or methods that ideally should have been localized into a single abstraction are separated and spread across multiple abstractions.
Insufficient Modularization	This smell arises when an abstraction exists that has not been completely decomposed and a further decomposition could reduce its size, implementation complexity, or both.
Hub-like Modularization	This smell arises when an abstraction has dependencies (both incoming and outgoing) with a large number of other abstractions.
Cyclically-dependent Modularization	This smell arises when two or more abstractions depend on each other directly or indirectly.
Wide Hierarchy	This smell arises when an inheritance hierarchy is too wide indicating that intermediate abstractions may be missing.
Deep Hierarchy	This smell arises when an inheritance hierarchy is excessively deep.
Multipath Hierarchy	This smell arises when a subtype inherits both directly as well as indirectly from a supertype leading to unnecessary inheritance paths in the hierarchy.
Cyclic Hierarchy	This smell arises when a supertype in a hierarchy depends on any of its subtypes.
Rebellious Hierarchy	This smell arises when a subtype rejects the methods provided by its supertype(s).
Unfactored Hierarchy	This smell arises when there is unnecessary duplication among types in a hierarchy.
Missing Hierarchy	This smell arises when a code segment uses conditional logic to explicitly manage variation in behavior.
Broken Hierarchy	This smell arises when a supertype and its subtype conceptually do not share an IS-A relationship resulting in broken substitutability.

3.2 Detailed Metric Analysis

Designite computes 30 commonly used metrics at different granularities (viz. analyzed projects or solution, project, type, and method). The computed metrics at type and method granularities are presented in the form of an interactive pie chart (see Figure 3). The pie chart has four partitions viz. green, yellow, orange, and red. Designite shows the distribution of entities that fall in each partition based on their values and pre-defined thresholds. The pie chart provides an overall idea about the project code quality from the selected metrics point of view instantly.


WMC - Weighted Method per Class
Distribution of types based on metric thresholds
A
A
A
Reset thresholds
Reset
Select metric

🟩 Types honoring the metric threshold - 13
🟨 Types slightly above the metric threshold - 4
🟧 Types quite above the metric threshold - 5
🟥 Types dangerously above the metric threshold - 2

Green threshold 80.00
Yellow threshold 100.00
Orange threshold 150.00

○ DIT ○ Fan-in ○ Fan-out ○ LCOM ○ LOC ○ NC ○ NOF ○ NOM ○ NOPF ○ NOPM ○ NOP [✓ WMC]

Figure 3: Detailed metric analysis by Designite

Additionally, the pie chart can also be used as a navigation and filtering mechanism. For instance, if one clicks on the red pie, classes that are dangerously above the metric threshold will only be shown in the grid below the pie chart. One can see other associated metrics for the filtered classes and may target them first for refactoring. Even further, one can change the thresholds to customize the analysis.

3.3 Dependency Analysis Using DSM

Dependency Structure Matrix (DSM) is a compact and visual representation of dependencies (with corresponding strengths) among software entities. Designite offers a DSM implementation that not only support dependency analysis for types but also for namespaces and projects. Further, one can change the scope as well to focus on a specific project.

3.4 Hotspot Analysis

Hotspot analysis is a derived analysis based on the detected smells that highlights the classes suffering from various design smells and could be chosen for refactoring first. For this inference, Designite chooses a smallest set of classes that contribute up to 20% of the detected smells.

3.5 Code-clone Detection

Designite employs a block index-based clone detection algorithm based on algorithm proposed by Hummel et al. [4] with block size of 20 lines of code and detects type-1 clones. Apart from revealing the code duplication, the identified

Table 2: Results of the Evaluation Performed on Three Open-source Projects

	MonoDevelop			NUnit			GitExtensions		
Version	4.2.0.7	5.6.3.3	5.10.0.871	2.9.1	2.9.4	2.9.7	2	2.33	2.48.05
LOC	605,422	674,869	674,501	31,595	89,286	613,045	33,175	60,037	86,528
Abstraction smells	2,977	3,047	3,108	326	843	5,529	82	150	234
Encapsulation smells	432	355	307	5	49	406	24	23	32
Modularization smells	1,949	1,769	1,803	33	70	512	69	159	309
Hierarchy smells	546	560	566	178	400	2,065	63	86	59
Smell density	10.39	9.92	9.84	14.18	16.26	13.88	7.17	6.96	7.33

clone-sets are also used in detecting design smells Duplicate Abstraction [15] and Unfactored Hierarchy [15].

3.6 Integration with External Tools

Designite also supports a console application to execute the analysis and generate output as an excel sheet or XML file. The console application could be used to integrate Designite with in-place build process. It is useful to enforce quality guidelines within a software development team or organization. Further, the smells detected by Designite could be imported within SonarQube [14] using Designite plugin [11] for consolidated technical debt management.

3.7 Export

Export is another important feature of the tool that could be used to export the detected smells, generated metrics, and identified code-clones to an MS excel sheet. This useful feature allows one to share the result of the tool with other stakeholders in his/her team and analyze the results further.

4. EVALUATION

In this section, we present our brief evaluation of the tool on three open-source C# based projects viz. MonoDevelop [7], Nunit [10], and GitExtensions [3]. The analysis has been performed on three versions of each project to show how these projects evolved from the perspective of design quality. Table 2 shows the detected design smells classified based on the principle they violate and corresponding smell density, which is measured by total smells detected per thousand lines of code. Apart from detecting design smells in individual versions, the result can also be used to infer insights from the trend. For instance, although the codebase size of NUnit increased significantly from version 2.9.4 to 2.9.7, the smell density registers considerable decline in the newer version.

The evaluation shows that the tool can reveal structural design smells even for large codebases and thus it could play an effective role to improve software design quality.

5. CONCLUSIONS AND FUTURE WORK

The paper discussed a software design quality assessment tool viz. Designite. The tool supports comprehensive design smells detection, detailed interactive metrics analysis, DSM for allowing dependency analysis for three granularities, code-clone detection, integration with external tools such as SonarQube, and exporting detected smells and computed metrics to an excel sheet. With all these features, Designite assesses the design quality of a software system and helps improve the design agility of the software. The evaluation shows that the tool is able to detect various smells in large codebases and thus is useful in reducing design debt of a software. In future, we plan to carry out an extensive evaluation to compute metrics such as recall and precision of the tool. We also plan to extend the tool to support features such as differential analysis and trend analysis.

6. REFERENCES

[1] Designite. http://www.designite-tools.com/, 2016. [Online; accessed 22-Jan-2016].

[2] S. Ganesh, T. Sharma, and G. Suryanarayana. Towards a principle-based classification of structural design smells. *Journal of Object Technology*, 12(2):1:1–29, June 2013.

[3] GitExtensions. https://github.com/gitextensions, 2016. [Online; accessed 01-Feb-2016].

[4] B. Hummel, E. Juergens, L. Heinemann, and M. Conradt. Index-based code clone detection: incremental, distributed, scalable. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–9, Sept 2010.

[5] JDeodorant. https://marketplace.eclipse.org/content/jdeodorant, 2016. [Online; accessed 01-Feb-2016].

[6] C. Jones. Software Quality in 2012: A Survey of the State of the Art. http://sqgne.org/presentations /2012-13/Jones-Sep-2012.pdf, 2012. [Online; accessed 01-Feb-2016].

[7] MonoDevelop. https://github.com/mono/monodevelop, 2016. [Online; accessed 01-Feb-2016].

[8] NDepend. http://www.ndepend.com/, 2016. [Online; accessed 01-Feb-2016].

[9] NRefactory. https://github.com/icsharpcode/NRefactory, 2016. [Online; accessed 01-Feb-2016].

[10] Nunit. http://www.nunit.org/, 2016. [Online; accessed 01-Feb-2016].

[11] Designite's plugin for SonarQube. https://github.com/Designite/sonar-designite-plugin, 2016. [Online; accessed 01-Feb-2016].

[12] Resharper. https://www.jetbrains.com/resharper/, 2016. [Online; accessed 01-Feb-2016].

[13] T. Sharma, G. Suryanarayana, and G. Samarthyam. Challenges to and solutions for refactoring adoption: An industrial perspective. *Software, IEEE*, 32(6):44–51, Nov 2015.

[14] SonarQube. http://www.sonarqube.org/, 2016. [Online; accessed 22-Jan-2016].

[15] G. Suryanarayana, G. Samarthyam, and T. Sharma. *Refactoring for Software Design Smells: Managing Technical Debt*. Morgan Kaufmann, 1 edition, 2014.