Managing Software Dependencies using Design Structure Matrices

Matthias Stevens[1]*, Andy Kellens[1]**, Johan Brichau[2], and Theo D'Hondt[1]

[1] Programming Technology Lab
Vrije Universiteit Brussel
{mstevens—akellens—tjdhondt}@vub.ac.be
[2] Département d'Ingénierie Informatique
Université catholique de Louvain
johan.brichau@uclouvain.be

1 Introduction

Modularity plays an important role in increasing the evolvability and maintainability of software systems. If a system is structured in such a way that the different components are loosely coupled, this makes it possible for developers – or teams of developers – to adapt and maintain a particular component in relative isolation of the other components of the system. Conversely, if components are tightly coupled this can constrain the evolvability of the system. For instance, low-level implementation dependencies between the different components often translate to dependencies among people or develop teams, resulting in that changes local to one implementation component can have a drastic impact throughout the system.

In this paper we present DSMBrowser [1], a novel approach and accompanying source-code browser for managing dependencies in source code. Our approach is based on the notion of Design Structure Matrices (DSM), a technique from the domain of project management. Our tool allows for the computation of such DSMs directly from source code, thus providing a visualisation of the dependencies between different components in a system. Moreover, our approach maintains the causal link with the actual source code. This, in combination with a meta-programming interface, makes it possible to reason about the different dependencies in the source code.

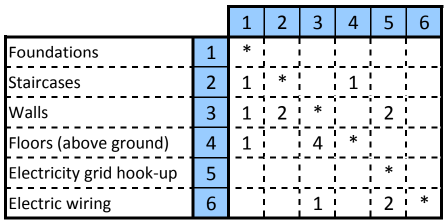
In the following sections we give a brief overview of the topic of Design Structure Matrices and our tool, DSMBrowser. Furthermore, we demonstrate one application of our meta-programming interface, namely the automatic identification of refactoring opportunities in order to remove dependencies between components.

* Matthias Stevens is a Research Assistant of the Fund of Scientific Research, Flanders (Aspirant van het Fonds Wetenschappelijk Onderzoek – Vlaanderen).

** Andy Kellens is funded by the "Institute for the Promotion of Innovation through Science and Technology in Flanders" (IWT Vlaanderen).

2 Design Structure Matrices

At the core of our approach lies the notion of Design Structure Matrices (DSMs) [2]. A DSM diagram names the constituent parts of a system and visualises their dependencies using a square adjacency matrix containing dependency values. These values can be binary, expressing only the existence or absence of dependency, or numerical, expressing a degree of dependency between parts of the studied system. To illustrate the concept of a DSM, consider the example in Figure 1. This



		1	2	3	4	5	6
Foundations	1	*					
Staircases	2	1	*		1		
Walls	3	1	2	*		2	
Floors (above ground)	4	1		4	*		
Electricity grid hook-up	5					*	
Electric wiring	6			1		2	*

Fig. 1. A DSM diagram for the design of a house

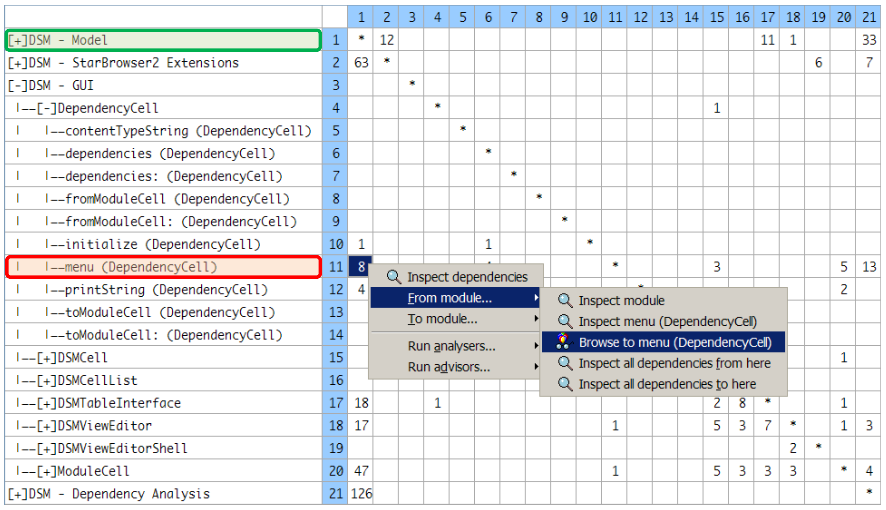
example shows a DSM for a familiar system: a house. We see that some parts of the house, namely the staircases, the walls, the floors above ground level and the electrical wiring, depend on other parts, while others, namely the foundations and the hook-up to the electric grid, are self-reliant. The different degrees of dependencies can indicate the number of individual dependencies and/or the weight of specific types of dependencies.

In our approach, we use similar diagrams as a means to visualise the dependencies between source-code entities. The different rows and columns of a DSM represent the source-code entities present in a system; as for dependencies between such source-code entities relations such as direct referencing, subclassing, message sends, and so on are used.

3 DSMBrowser

As a proof of concept, we have implemented DSMBrowser, an extension to the VisualWorks Smalltalk development environment and the StarBrowser2 framework [3]. For a user-selected collection of arbitrary source-code entities, our tool can automatically calculate a DSM based on the static relations between the entities.

Figure 2 shows a screenshot of DSMBrowser opened on a part of its own implementation. For each of the modules of DSMBrowser, the dependencies between this module and the other modules of the system are displayed. Internally, DSMBrowser uses an hierarchical representation of abstract modules. This representation aligns with the different hierarchical levels of (Smalltalk) source code – packages, classes and methods. These different levels are also reflected in DSMBrowser's interface. For example, the tree-view on the left in Figure 2 shows the



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
[+]DSM - Model	1	*	12															11	1			33
[+]DSM - StarBrowser2 Extensions	2	63	*																	6		7
[-]DSM - GUI	3			*																		
|--[-]DependencyCell	4				*											1						
| |--contentTypeString (DependencyCell)	5					*																
| |--dependencies (DependencyCell)	6						*															
| |--dependencies: (DependencyCell)	7							*														
| |--fromModuleCell (DependencyCell)	8								*													
| |--fromModuleCell: (DependencyCell)	9									*												
| |--initialize (DependencyCell)	10	1					1				*											
| |--menu (DependencyCell)	11	8										*				3					5	13
| |--printString (DependencyCell)	12	4											*								2	
| |--toModuleCell (DependencyCell)	13																					
| |--toModuleCell: (DependencyCell)	14																					
|--[+]DSMCell	15																				1	
|--[+]DSMCellList	16																					
|--[+]DSMTableInterface	17	18			1											2	8	*			1	
|--[+]DSMViewEditor	18	17										1				5	3	7	*		1	3
|--[+]DSMViewEditorShell	19																		2	*		
|--[+]ModuleCell	20	47										1				5	3	3	3		*	4
[+]DSM - Dependency Analysis	21	126																				*

Inspect dependencies
From module...
To module...
Run analysers...
Run advisors...
Inspect module
Inspect menu (DependencyCell)
Browse to menu (DependencyCell)
Inspect all dependencies from here
Inspect all dependencies to here

Fig. 2. Screenshot of DSMBrowser opened on part of its own implementation

different hierarchical levels of the `DSM - GUI` package – from the top package-level down to the level of methods. This hierarchical representation allows users to focus on the dependencies between a relevant, fine-grained subset of the system while abstracting away the details of other parts.

DSMBrowser maintains the causal connection between the representation of dependencies in the interface and the actual, underlying source-code entities. As such, a user can browse the different source-code entities related to a particular module or dependency directly from within the DSMBrowser tool (as shown by the contextual menu in the screenshot).

Moreover, DSMBrowser offers a meta-programming interface that allows users to define scripts that reason about the different dependencies in the system. Furthermore, DSMBrowser offers the possibility to access these scripts from within the contextual menu to target specific sets of dependencies. For reasons of brevity, we will not discuss the meta-programming interface in detail in this paper, however we briefly demonstrate its utility by means of a small scenario.

4 Example of meta-programming: minimizing dependencies using indirections

Using our meta-programming interface we have implemented a reusable script that allows for the minimization of message send dependencies between two packages by introducing indirections. For example, our script can suggest the introduction of proxy classes.

If message sends originate from *many* classes in a package A and target *few* classes in a package B, then this dependency from package A to package B can be lowered by applying the Proxy design pattern [4]. The idea is to introduce a local proxy-class in package A, for each of the *few* classes which are targeted in

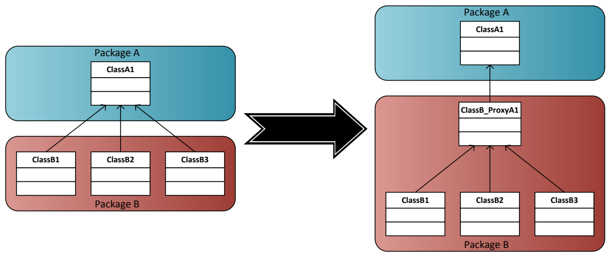


Fig. 3. Minimizing dependencies by introducing a proxy class

package B. Each proxy-class should contain wrapper methods for all methods of the class of package B it represents and which are called from package A. That way all calls to package B can be rerouted via local proxies. As a result there will only be a single inter-package message send dependency for every targeted method. This mechanism is shown in Figure 3.

When executed, our script will propose a refactoring opportunity to minimize the dependency, as illustrated by the following transcript:

```
Message sends from package Intensional Relations Model
    to package SoulEvalPrintLoop: 30

 Analysing for Class level advice...
  The message sends originate from 7 classes and target 4 classes

  Advice: The degree of dependency from package
    Intensional Relations Model
    to package SoulEvalPrintLoop can be lowered by introducing
    4 PROXY classes in package Intensional Relations Model which
    represent classes EmptyEvaluator, Results, Evaluator, Binding
    of package SoulEvalPrintLoop.
```

5 Summary

In this paper, we briefly highlighted some of the features of DMSBrowser, our novel approach for source-code dependency management as a means to support software maintenance and evolution.

References

1. Stevens, M.: Design Structure Matrices for Software Development. Licentiate thesis, Vrije Universiteit Brussel, Faculty of Science, Department of Computer Science, Pleinlaan 2, B-1050 Brussels, Belgium (2007)
2. Steward, D.V.: The Design Structure System: A Method for Managing the Design of Complex Systems. IEEE Transactions on Engineering Management **28**(3) (1981) 71–74
3. Wuyts, R., Ducasse, S.: Unanticipated integration of development tools using the classification model. Computer Languages, Systems & Structures **30**(1-2) (2004) 63–77
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Addison-Wesley, Boston, MA, USA (1994)