



Turun yliopisto  
University of Turku

# ***DebtFlag*: Technical Debt Management with a Development Environment Integrated Tool**

MTD 2013, San Francisco, CA, USA

**Johannes Holvitie & Ville Leppänen**

TUCS – Turku Centre for Computer Science  
UTU – Department of Information Technology  
Turku, Finland

## Motivation and Problem

- **Availability and clarity of technical debt information is a key factor in successful technical debt management**
- **Software products are highly complex, self-emergent and experience constant updates**
  - Two approaches to TD information production
    - Automatic: accommodates update rates; incapable of capturing entire requirements space
    - Manual: capable of capturing the entire requirements space; incapable of accommodating update rates

## Solution

- **Combine the two approaches**
  - Manual assessment to identify source points
  - Partially automate the documentation process
  - Fully automate the processes of propagation and technical debt information maintenance

## DebtFlag

- **DebtFlag links structured observations about technical debt to related parts of the software implementation and**
- **uses implementation technique specific information to maintain them.**
- **DebtFlag produced technical information pursues technical debt management at**
  - the implementation level (micromanagement)
  - the project level (adherence to TDMF)

# DebtFlag

- **DebtFlag mechanism**
  - software implementation process and technique independent, system abstraction
    - structure of documented technical debt
    - automation of technical debt propagation
    - technical debt management
- **DebtFlag tool**
  - implementation of the DebtFlag mechanism
  - Java (Eclipse plug-in + Vaadin web-app)

## DebtFlag Mechanism

### Structure of Documented Technical Debt

- **Uses the documentation structure from Technical Debt Management Framework\***
- **Extends this with an additional layer**
  - TDIs composed from DebtFlag elements
    - user may define a TDI as an unlimited collection of implementation technique specified elements
    - each element may introduce a unique rule set for the propagation of its technical debt

\*Seaman, C., & Guo, Y. (2011). Measuring and monitoring technical debt. *Advances in Computers*, 82, 25-46.

## DebtFlag Mechanism

### Automation of Technical Debt Propagation

- **DebtFlag's dynamic functionality requires that two processes are automated**
  - identifying source points for TD propagation
  - propagation of technical debt according to rule sets and dependencies

## DebtFlag Mechanism

### Technical Debt Management

- **Project level management**
  - Support for the TDMF
    - DebtFlag mechanism is designed to be able to efficiently construct and maintain the TDL for the software implementation artifact
- **Implementation level micromanagement**
  - DebtFlag maintains an implementation level representation of technical debt

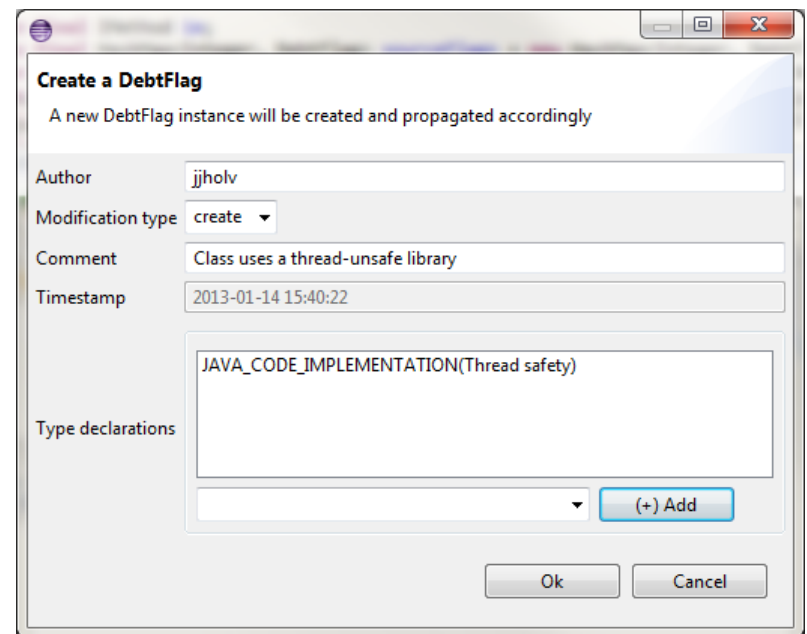


## DebtFlag Tool

- **Implementation of the DebtFlag mechanism**
- **DebtFlag plug-in**
  - for the Eclipse IDE
  - supports developing in Java
  - TD micromanagement
- **DebtFlag web-application**
  - dynamic representation of the TDL
  - project level management

## DebtFlag Plug-In – Capturing Technical Debt

- Triggered through interaction with Java elements in the Eclipse
- Partially automated documentation of technical debt (creation of DebtFlag elements)



**Create a DebtFlag**  
A new DebtFlag instance will be created and propagated accordingly

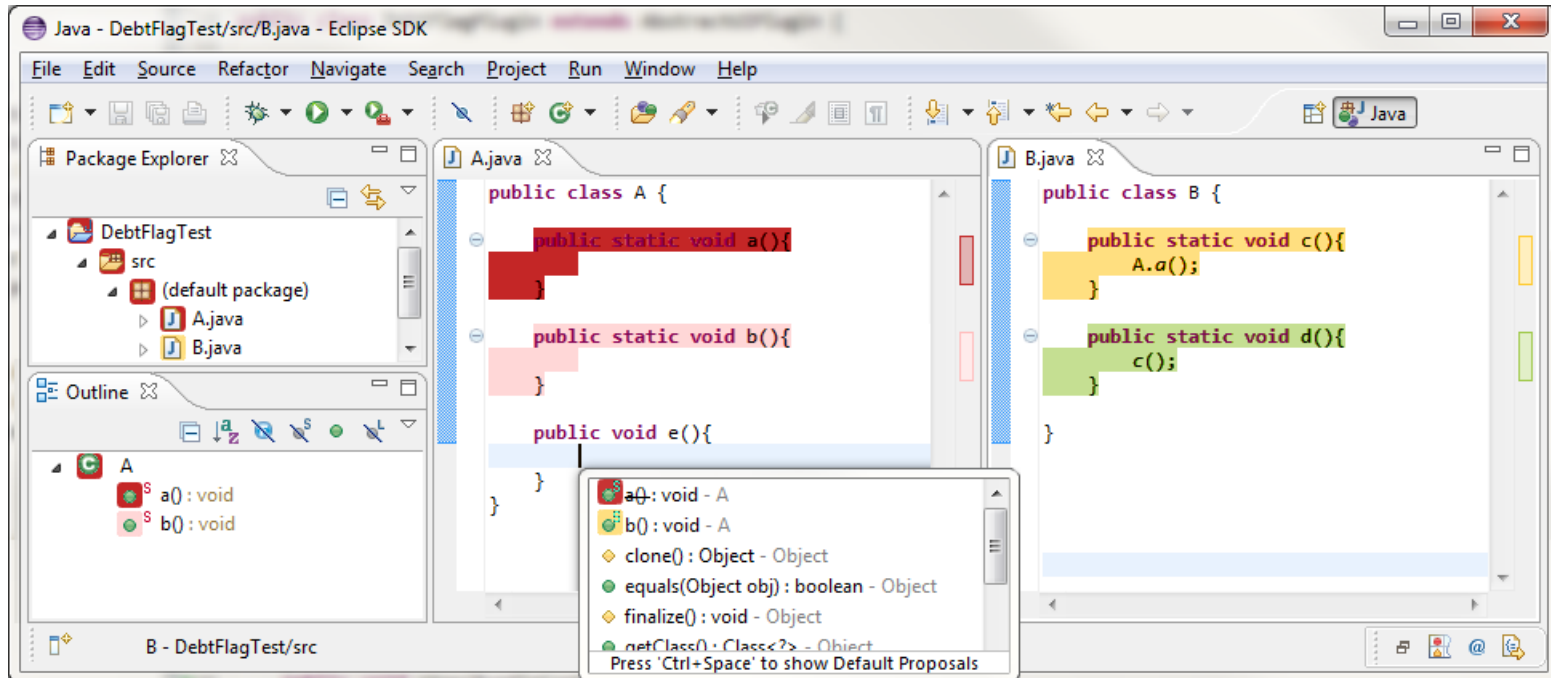
Author: jjholv  
Modification type: create  
Comment: Class uses a thread-unsafe library  
Timestamp: 2013-01-14 15:40:22

Type declarations:  
JAVA\_CODE\_IMPLEMENTATION(Thread safety)  
(+) Add

Ok Cancel

# DebtFlag Plug-In – Implementation Level Representation of Technical Debt

- **Micromanagement; visualization and restriction**



```
Java - DebtFlagTest/src/B.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
  DebtFlagTest
    src
      (default package)
        A.java
        B.java
Outline
  A
    a(): void
    b(): void
  B
    c(): void
    d(): void
  B - DebtFlagTest/src

public class A {
    public static void a(){
    }

    public static void b(){
    }

    public void e(){
    }
}

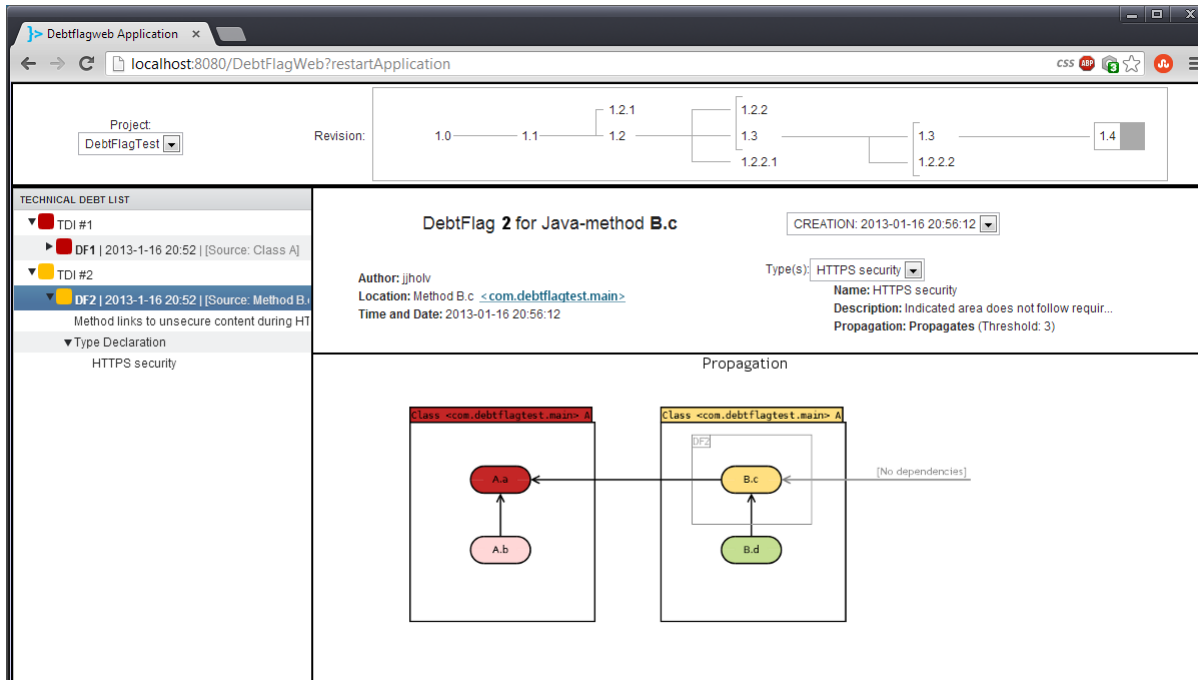
public class B {
    public static void c(){
        A.a();
    }

    public static void d(){
        c();
    }
}
```

a(): void - A  
b(): void - A  
clone(): Object - Object  
equals(Object obj): boolean - Object  
finalize(): void - Object  
netClass0.Class<?>. - Object  
Press 'Ctrl+Space' to show Default Proposals

# DebtFlag Web Application – Dynamic Representation of the TDL

- Project level TD management
  - changes propagated back to implementation



The screenshot displays the DebtFlag web application interface. At the top, there is a navigation bar with a "Project" dropdown set to "DebtFlagTest" and a "Revision" section showing a version history diagram from 1.0 to 1.4. The main content area is divided into two panels. The left panel, titled "TECHNICAL DEBT LIST", shows a tree view of technical debt items (TDI #1, DF1, TDI #2, DF2) with details for the selected item "DF2 | 2013-1-16 20:52 | [Source: Method B.]". The right panel, titled "DebtFlag 2 for Java-method B.c", shows the details of the selected debt item, including its creation date, author, location, and type (HTTPS security). Below this, a "Propagation" diagram illustrates the flow of the debt from the implementation (Class <com.debtflagtest.main> A) back to the source code (Class <com.debtflagtest.main> A). The diagram shows a flow from B.c to A.a, and from B.d to B.c, with a note indicating "No dependencies".

## Future work

- **Mechanism Improvement and Validation**
  - Department software projects
  - Experimentation and case studies with industrial partners
    - Nokia ICM (legacy code project in Pro\*C)
- **Propagation model improvement**
  - Variety of possible inputs and models to use
- **Extending technique support**

## Discussion

- **Benefits**
- **Drawbacks**
- **Application in software development**
- **Propagation modeling**
- **Propagation analysis**

## Discussion

### **Expected benefits of using the DebtFlag**

- **captures human made observations**
- **documents the structure of technical debt**
- **presents technical debt at the implementation level**
- **makes continued use of higher level technical debt management approaches possible**

## Discussion

### **Foreseen drawbacks to using the DebtFlag**

- **may endorse technical debt accumulation**
- **places the burden of technical debt management onto the end user**
- **does not protect the information from propagation rule set bias**
- **is heavily dependent onto outside services**



## Application in Software Development

- **Integrate into parts of the software process where relevant observations are made**
  - Implementation process
  - Developers are the end users
- **Produced documentation serves as the integration point for further technical debt management approaches**
  - TDMF or other SW dev. approaches

## Propagation Modeling

- **How does dependency propagation affect TD principal and interest?**
  - forms of propagation
  - types of implementation elements
- **Modeling the effect of diminishing TD**
- **Supporting TD management**
- **Possibility in link structure algorithms**
  - PageRank

# Propagation Analysis

