



PROTOTYPICAL REALIZATION AND VALIDATION OF AN INCREMENTAL SOFTWARE PRODUCT LINE ANALYSIS APPROACH

Präsentation zur Masterarbeit

Letztes Update: 16. Juli 2018

Moritz Flöter

Universität Hildesheim

GLIEDERUNG

1. Idee & Motivation

2. Grundlagen

3. Konzept & Implementierung

4. Dead Code Analyse

5. Evaluation

6. Fazit

IDEE & MOTIVATION

AUSGANGSSITUATION

- Analysen unterstützen Softwareentwicklung
- Analysen für Software Produktlinien (SPL) sind rechenintensiv [TAK⁺14]
- Relativ späte Verfügbarkeit von Ergebnissen
- Entwickler macht mittlerweile was anderes

Inkrementelle Analysen

- Nicht jedes mal alles neu analysieren
- Über eingeführte Veränderungen bestimmen, welcher Teil analysiert werden muss

Umsetzung auf Basis von KernelHaven

GRUNDLAGEN

LINUX BUILD PROCESS

Variability Model

KConfig

Build Model

KBuild

Code Model

CPP Code

VARIABILITÄT - VARIABILITY MODEL

Variability Model

KConfig

Build Model

KBuild

Code Model

CPP Code

```
config NETWORK_AUTHENTICATION
    bool "Network Authentication"
    depends on USER_AUTHENTICATION
    select NETWORK_SUPPORT
    default y
    help
        Network authentication support ...
```


VARIABILITÄT - BUILD MODEL

Variability Model

KConfig

Build Model

KBuild

Code Model

CPP Code

Makefile for network device drivers.

`obj-$(CONFIG_NETWORK_SUPPORT) += generic-driver.o`

`obj-$(CONFIG_NETWORK_SUPPORT) += other-drivers/`

Beispiel: NETWORK_SUPPORT wurde ausgewaehlt

`obj-y += generic-driver.o`

`obj-y += other-drivers/`

VARIABILITÄT - CODE MODEL

Variability Model

KConfig

Build Model

KBuild

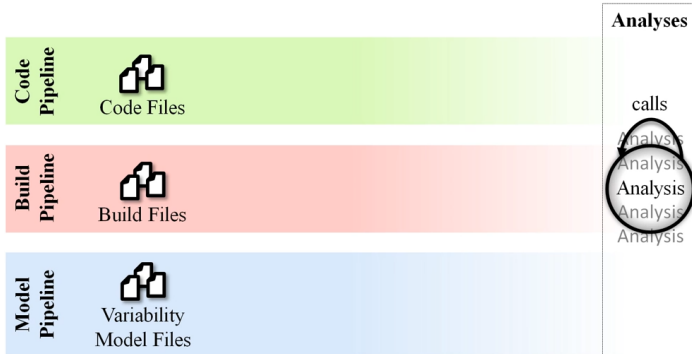
Code Model

CPP Code

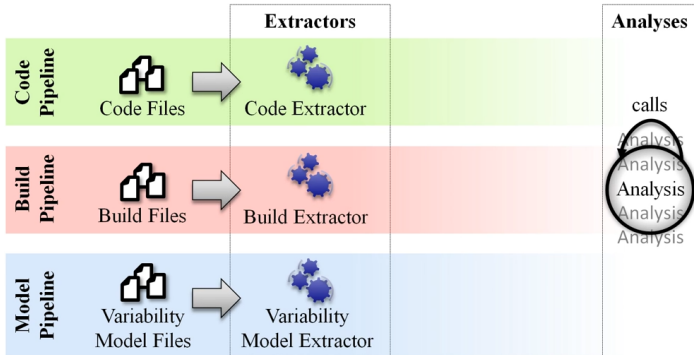
```
#ifdef CONFIG_NETWORK_SUPPORT  
    return 0;  
#else  
    return 1;  
#endif
```

Verständnis von Variabilitätsinformation nach Kröher et al.
[KGS18]

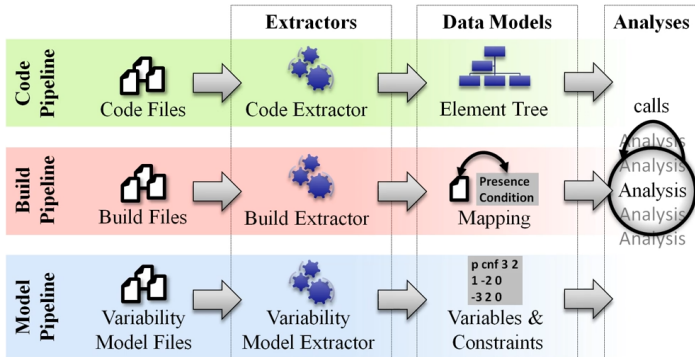
KERNELHAVEN



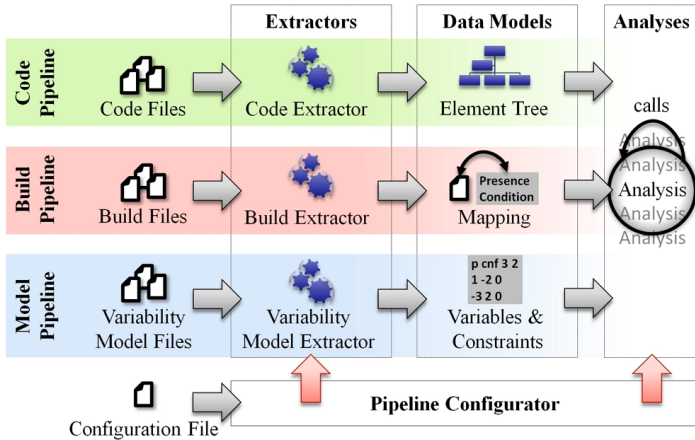
KERNELHAVEN



KERNELHAVEN



KERNELHAVEN



DEAD CODE ANALYSE - SPL

```

#ifdef CONFIG_NETWORK_AUTHENTICATION
    #ifdef !CONFIG_NETWORK_SUPPORT
        printf("You can not sign in");
    #endif
#endif

config NETWORK_AUTHENTICATION
    bool "Network Authentication"
        ...
    select NETWORK_SUPPORT
        ...

```

KONZEPT & IMPLEMENTIERUNG

REDUZIERUNG DES AUFWANDS

Neu eingeführte Veränderungen bestimmen, was erneut verarbeitet werden muss.

REDUZIERUNG DES AUFWANDS

Neu eingeführte Veränderungen bestimmen, was erneut verarbeitet werden muss.

1. Extraktion

Die Extraktion der Modelle muss nicht vollständig neu durchgeführt werden

REDUZIERUNG DES AUFWANDS

Neu eingeführte Veränderungen bestimmen, was erneut verarbeitet werden muss.

1. Extraktion

Die Extraktion der Modelle muss nicht vollständig neu durchgeführt werden

2. Analyse

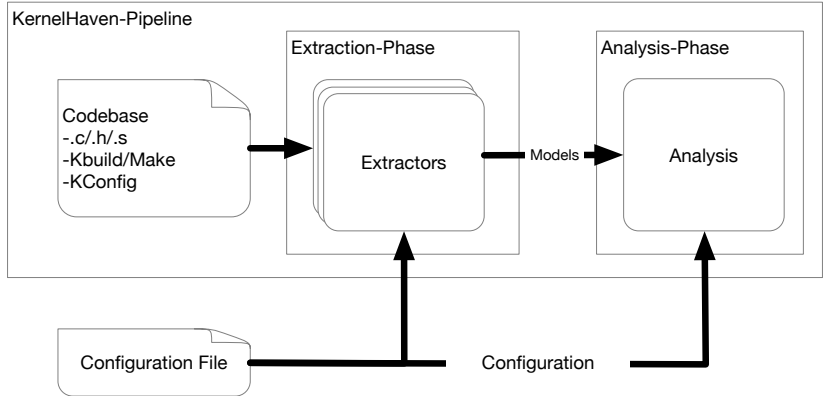
Die Analyse muss nicht vollständig neu durchgeführt werden

ANFORDERUNGEN

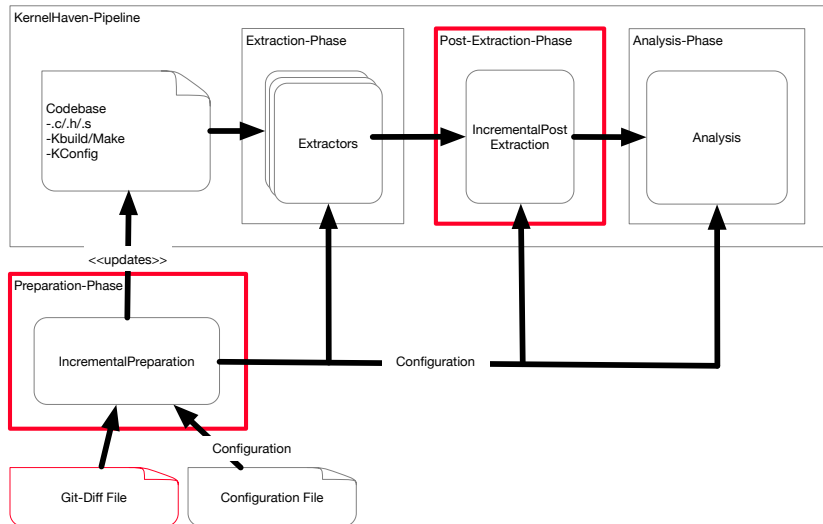
Im Vorfeld wurden Anforderungen an die Implementierung festgelegt. Dazu gehören unter anderem:

- Rollback
- Unterstützung von Block-basierten Analysen
- Konkrete Umsetzung von Dead Code Analyse

NICHT INKREMENTELLE ANALYSEN



INKREMENTELLE ANALYSEN



PREPARATION-PHASE

- Umsetzung als Preparation-Plugin
- Aufgaben der Preparation-Phase
 1. Anwenden der Änderungen auf die Codebase
 2. Filtern von Dateien für Extraktion
 3. Anpassen der Konfiguration

PREPARATION-PHASE

1. Anwenden der Änderungen auf die Codebase

```
$ git apply path/to/git.diff
```

```
$ git apply --reverse path/to/git.diff
```


PREPARATION-PHASE

2. Filtern von Dateien für Extraktion

Filter arbeiten auf Basis von folgenden Informationen:

PREPARATION-PHASE

2. Filtern von Dateien für Extraktion

Filter arbeiten auf Basis von folgenden Informationen:

- Dateiveränderung
Hinzufügen, Modifizieren, Löschen

PREPARATION-PHASE

2. Filtern von Dateien für Extraktion

Filter arbeiten auf Basis von folgenden Informationen:

- Dateiveränderung
Hinzufügen, Modifizieren, Löschen
- Variabilitätsänderung
Wurde Variabilitätsinformation verändert?

PREPARATION-PHASE

2. Filtern von Dateien für Extraktion

Filter arbeiten auf Basis von folgenden Informationen:

- Dateiveränderung
Hinzufügen, Modifizieren, Löschen
- Variabilitätsänderung
Wurde Variabilitätsinformation verändert?
- Codebase
Ein Filter hat Zugriff auf alle Dateien in der Codebase

PREPARATION-PHASE

2. Filtern von Dateien für Extraktion

Filter arbeiten auf Basis von folgenden Informationen:

- Dateiveränderung
Hinzufügen, Modifizieren, Löschen
- Variabilitätsänderung
Wurde Variabilitätsinformation verändert?
- Codebase
Ein Filter hat Zugriff auf alle Dateien in der Codebase
- Regulärer Ausdruck
Welche Dateien (.c/*.h/KConfig etc.) sollen berücksichtigt werden?*

PREPARATION-PHASE - FILTERTYPEN

Jeder Filter gibt eine *Liste von Dateien* zurück, welche für die Extraktion berücksichtigt werden.

Code, Build und Variability Dateien werden separat gefiltert.

Umgesetzte Filter:

- `DefaultFilter`
Typ der Datei

PREPARATION-PHASE - FILTERTYPEN

Jeder Filter gibt eine *Liste von Dateien* zurück, welche für die Extraktion berücksichtigt werden.

Code, Build und Variability Dateien werden separat gefiltert.

Umgesetzte Filter:

- `DefaultFilter`
Typ der Datei
- `ChangeFilter`
Typ der Datei, Dateiveränderung

PREPARATION-PHASE - FILTERTYPEN

Jeder Filter gibt eine *Liste von Dateien* zurück, welche für die Extraktion berücksichtigt werden.

Code, Build und Variability Dateien werden separat gefiltert.

Umgesetzte Filter:

- `DefaultFilter`
Typ der Datei
- `ChangeFilter`
Typ der Datei, Dateiveränderung
- `VariabilityChangeFilter`
Typ der Datei, Variabilitätsveränderung

PREPARATION-PHASE

3. Anpassen der Konfiguration

- Liste von Build Dateien nicht leer
→ Build Model **komplett** neu extrahieren

PREPARATION-PHASE

3. Anpassen der Konfiguration

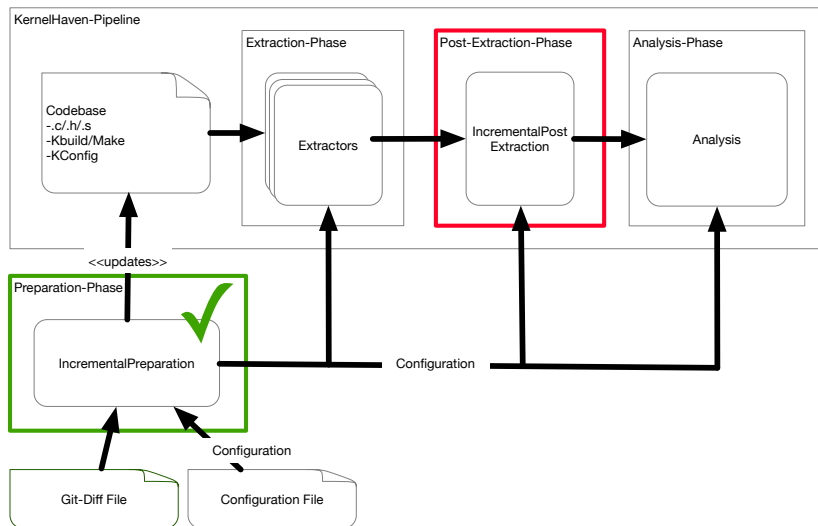
- Liste von Build Dateien nicht leer
→ Build Model **komplett** neu extrahieren
- Liste von Variability Dateien nicht leer
→ Variability Model **komplett** neu extrahieren

PREPARATION-PHASE

3. Anpassen der Konfiguration

- Liste von Build Dateien nicht leer
→ Build Model **komplett** neu extrahieren
- Liste von Variability Dateien nicht leer
→ Variability Model **komplett** neu extrahieren
- Liste von Code Dateien nicht leer
→ Code Model **für diese Dateien** neu extrahieren

INKREMENTELLE ANALYSEN



POST-EXTRACTION-PHASE

- Umsetzung als AnalysisComponent
- Aufgaben der PostExtraction-Phase
 1. Einholen der Ergebnisse von Extraktoren
 2. Zusammenführen mit Ergebnissen der vorigen Extraktion

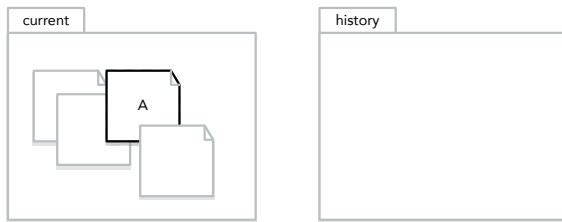
POST-EXTRACTION-PHASE

1. Einholen der Ergebnisse von Extraktoren

```
extractor.getNextResult();
```

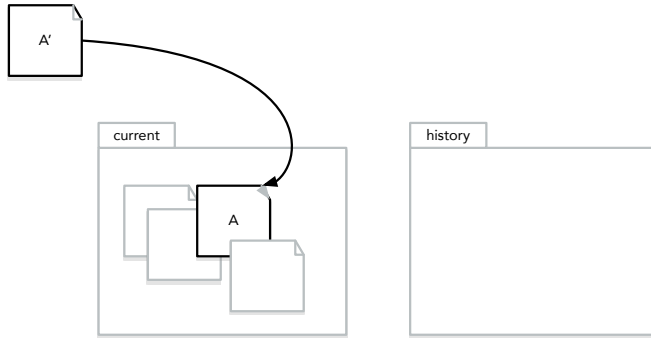
POST-EXTRACTION-PHASE

2. Zusammenführen mit Ergebnissen der vorigen Extraktion Konzept des HybridCache



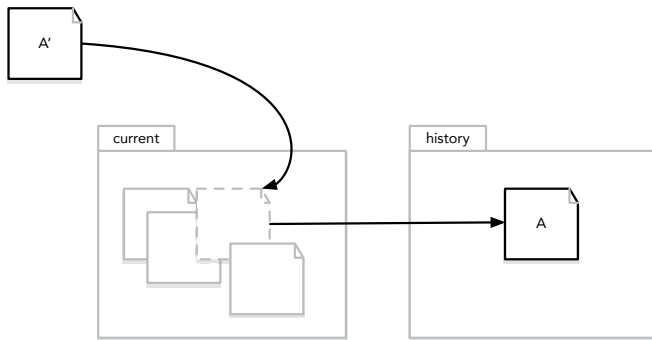
POST-EXTRACTION-PHASE

2. Zusammenführen mit Ergebnissen der vorigen Extraktion Konzept des HybridCache



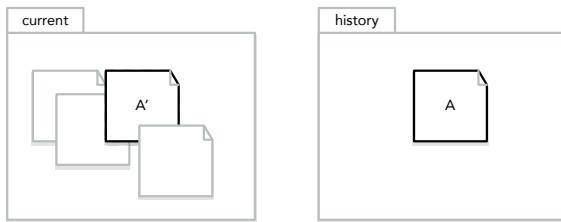
POST-EXTRACTION-PHASE

2. Zusammenführen mit Ergebnissen der vorigen Extraktion Konzept des HybridCache



POST-EXTRACTION-PHASE

2. Zusammenführen mit Ergebnissen der vorigen Extraktion Konzept des HybridCache

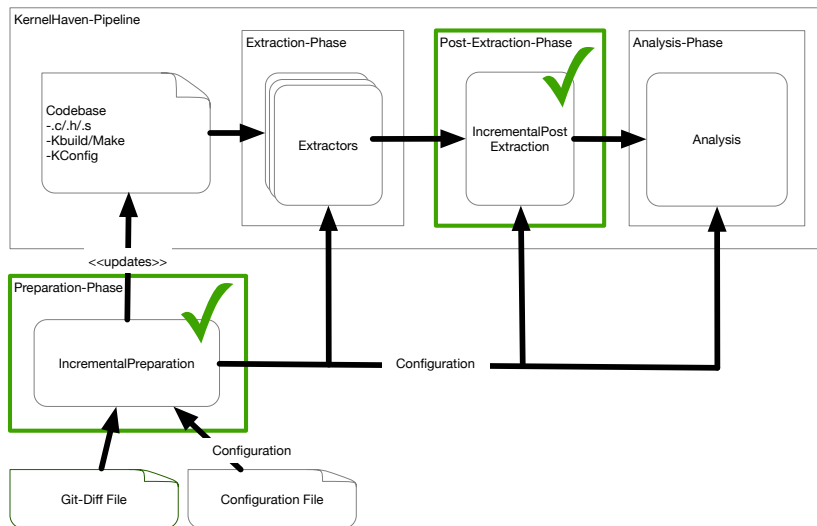


POST-EXTRACTION-PHASE

Was leistet der HybridCache?

- Verwaltung von (bis zu) zwei Versionen der Modelle
- Information über neu extrahierte Modelle
- Identifizierung von veränderten Teilen innerhalb der Modelle

INKREMENTELLE ANALYSEN



DEAD CODE ANALYSE

DEAD CODE ANALYSE - AUSGANGSSITUATION

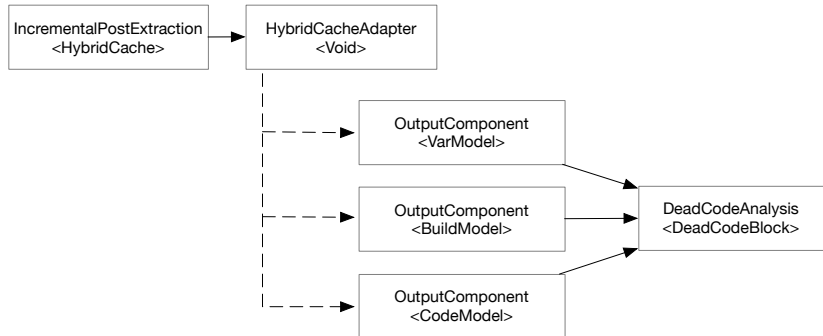
- nicht inkrementelle Variante existiert bereits

DEAD CODE ANALYSE - AUSGANGSSITUATION

- nicht inkrementelle Variante existiert bereits
- HybridCache ist bei bestehender Implementierung nicht als Input möglich

HYBRID CACHE ADAPTER

- ▶ AnalysisComponent is used as direct input for next component
- ▶ getters are used to obtain components



HYBRID CACHE ADAPTER

Das Code Model kann

- a) komplett
- b) partiell (nur für neu extrahierte Dateien)

weitergegeben werden.

DEAD CODE ANALYSE

Die Dead Code Analyse verarbeitet das Code Model

- a) komplett
wenn *variability* oder *build model* **verändert** wurden
- b) partiell (nur für neu extrahierte Dateien) wenn *variability* oder *build model* **nicht verändert** wurden

weitergegeben werden.

EVALUATION

EVALUATION

Vier Konfigurationen:

- Ref
- FilterOff
- Change
- VarChange

angewendet auf 306 Commits Linux Kernels (ca. 3 Monate)

EVALUATION - KONSISTENZ DER ERGEBNISSE

FilterOff

enthält **alle** Dead Code Blöcke, die Ref enthält

Change

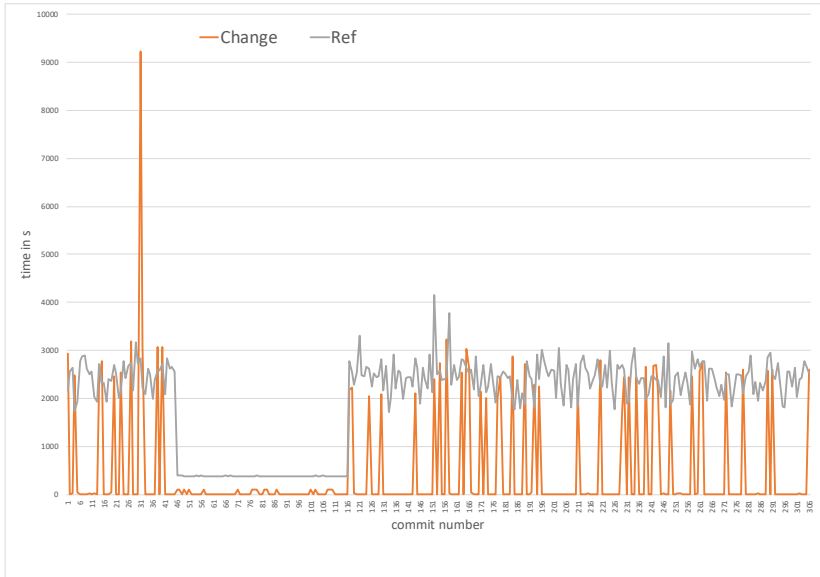
enthält **alle neuen** Dead Code Blöcke

VarChange

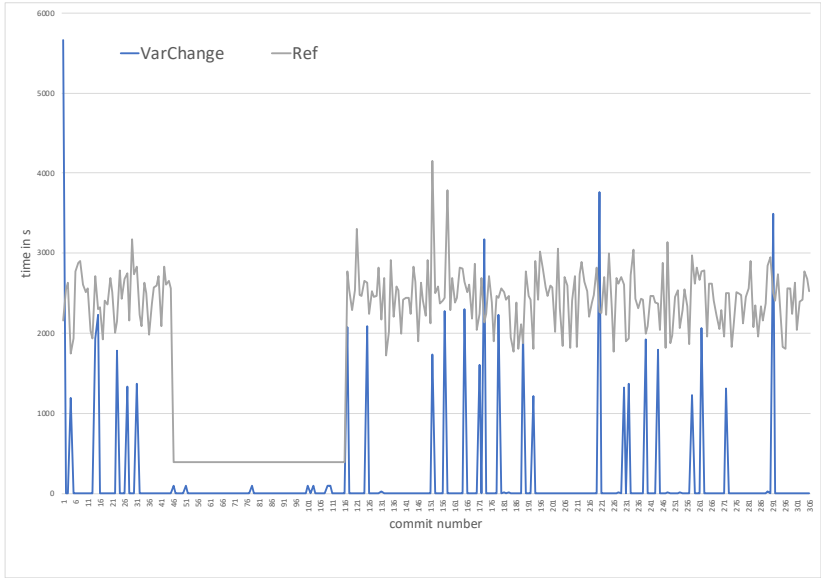
enthält **alle neuen variabilitätsbezogenen** Dead Code Blöcke

weicht teilweise in Zeilennummern der Dead Code Blöcke ab

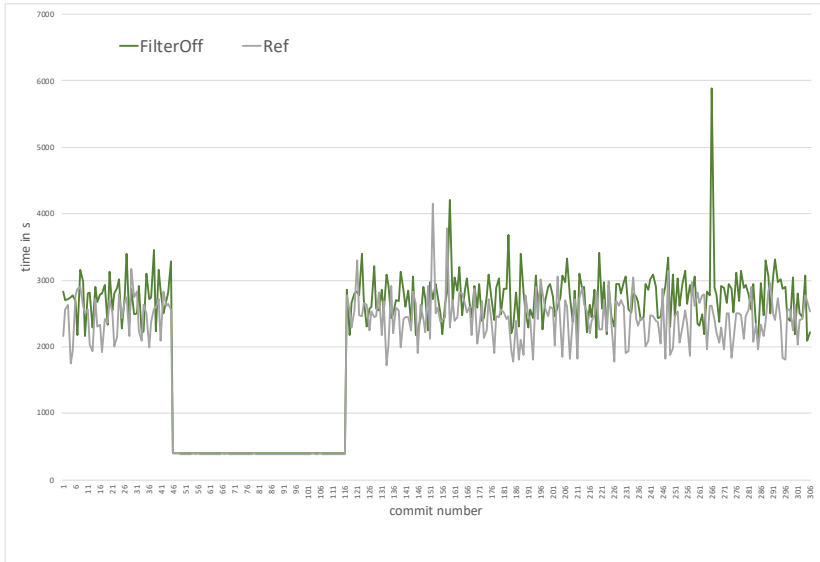
EVALUATION - CHANGE



EVALUATION - VARCHANGE



EVALUATION - REF



EVALUATION - NUMBERS

	Change	VarChange	Ref
Preparation	0.07 s	11.72 s	0.00 s
<u>Ex</u> traction	26.22 s	11.70 s	608.80 s
<u>A</u> nalysis	407.94 s	157.25 s	1792.80 s
<u>O</u> verlap	0.00 s	0.00 s	501.45 s
E + A - O	434.16 s	168.95 s	1900.15 s
Post-Extraction	1.91 s	0.50s	0.00 s
Total	7 min 16 s	3 min 2 s	32 min 47 s

- 10% (VarChange) - 20% (Change) brauchten länger als 60 Sekunden
- Die restlichen 80-90% der Ausführungen benötigten im Durchschnitt 6.6s (Change) bzw. 2.3s (VarChange)
- FilterOff war im Schnitt ca. 10% langsamer als Ref

FAZIT

FAZIT

Inkrementelle Analysen

- 5-10fache Performanz
- konsistente Ergebnisse

Future Work

- partielle Extraktion von Build & Variability Model
- Anwendung des Konzeptes auf andere Analysen

LITERATUR

- [KESS18] Christian Kröher, Sascha El-Sharkawy, and Klaus Schmid. Kernelhaven - an experimentation workbench for analyzing software product lines. In *Proceedings of the 40th International Conference on Software Engineering, ICSE '18*, 2018.
- [KGS18] Christian Kröher, Lea Gerling, and Klaus Schmid. Identifying the intensity of variability changes in software product line evolution. In *SPLC '18: Proceedings of the 23rd International Conference on Software Product Line*, 2018.
- [TAK⁺14] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47(1):6:1–6:45, June 2014.