



Пловдивски университет
„Паисий Хилендарски“

Факултет по математика и информатика

Катедра „Компютърни системи“

**Инструменти за статичен и
динамичен анализ в
домейн-специфичните визуални
езици за програмиране**

Автореферат

на дисертационен труд

за присъждане на образователна и научна степен “доктор”

в област на висше образование 4. Природни науки, математика и информатика;

Професионално направление 4.6. Информатика и компютърни науки;

докторска програма Информатика

Редовен докторант:

Мартин Георгиев Василев

Научен ръководител:

доц. д-р Александър Пенев

Пловдив, 2020

Дисертационният труд е обсъден и насочен за защита от катедрения съвет на Катедра „Компютърни системи“ на Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“ на 07.02.2020 г.

Дисертационният труд Инструменти за статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране съдържа 148 страници. Библиографията включва 114 източника. Списъкът на авторските публикации се състои от 4 заглавия. В автореферата номерацията на формулите, цитиранията и примерите съвпада с тяхната номерация в дисертационния труд.

Защитата на дисертационния труд ще се състои на 09.06.2020 г. от 13:00 часа в Заседателната зала на Нова сграда на Пловдивски университет „Паисий Хилендарски“ (бул. “България” No 236). Материалите по защитата са на разположение на интересуващите се в секретариата на Факултет по математика и информатика на Пловдивски университет „Паисий Хилендарски“, (бул. “България” No 236, Пловдив), каб. 330, всеки работен ден от 8:30 до 17:00 ч.

Рецензенти:

проф. д-р Аврам Моис Ескенази, Икономически университет - Варна

проф. д-р Станимир Недялков Стоянов, ФМИ на ПУ “Паисий Хилендарски”

Автор:

Мартин Георгиев Василев

Заглавие:

Инструменти за статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране

Тираж 100 бр.

Университетско издателство “Паисий Хилендарски”

Пловдив, 2020 г.

Съдържание

1	Обща характеристика на дисертационния труд	1
1.1	Актуалност на тематиката	2
1.2	Цели и задачи на дисертационния труд	2
1.3	Структура на дисертационния труд	4
2	Кратко съдържание на дисертационния труд	5
Глава 2	Обзор на проблемната област	5
2.1	Основни понятия	5
2.2	Сравнителен анализ	8
2.3	Изводи и перспективи	9
Глава 3	Модел на инструмент за статичен и динамичен анализ в домейн-специфични визуални езици за програмиране	9
3.1	Основен йерархичен многомерен модел	9
3.2	Статичност и динамичност – функция на времето	11
3.3	Методи за извличане на информация и работа на системата	13
3.4	Визуализация на алгоритми и данни	13
3.5	Алгоритъм за диференциално дебъгиране (сравняване)	13
3.6	Изводи и перспективи	15
Глава 4	Архитектура и прототипна реализация на инструмент за ста- тичен и динамичен анализ в домейн-специфичните ВЕП	15
4.1	Съпътстващи разработки	16
4.2	Визуално диференциално дебъгиране	16
4.3	Използвани технологии и политики за разработка	17
4.4	Изводи и перспективи	18
Глава 5	Приложения и резултати	18
5.1	Визуален, домейн-специфичен статичен и динамичен ана- лиз със SolidIDE, SolidReflector и др.	18
5.2	Домейн-специфично диференциално дебъгиране с IDD	19
5.2.1	Текстови езици за програмиране	19
5.2.2	Визуални езици за програмиране	20
3	Заклучение	21
3.1	Резултати и приноси	21
3.1.1	Публикации по дисертационния труд	21
3.1.2	Участия в проекти	22
3.1.3	Доклади на конференции и семинари	23
3.1.4	Аудиторни занятия във висши училища	23
3.2	Изводи и перспективи	23
	Библиография	29

Обща характеристика на дисертационния труд

Докторската дисертация описва модел, архитектура и прототипна реализация на инструмент, подпомагащ създаването на отворени, интерактивни и динамични системи за анализ. Представя се кратко въведение в проблемната област, последвано от обстоен анализ на основните проблеми и съществуващите решения. Поставените цели са на основата на добре анализирани обобщения и съответстващите им изводи. За постигането им е описана подробно теоретична рамка, която позволява да се контролира (и често намалява) сложността при изграждането на такъв тип отворени инструменти. Теоретичната рамка разглежда модел на софтуерна система като поредица от взаимосвързани подмодели от различни домейни. Подмоделите целят да съставят единна база за създаване на анализиращи системи за инспекция на софтуер.

Наред с концептуалната обосновка, работата предлага прототипна реализация, целяща да докаже практическите възможности и ползи в сферата на текстовите и визуалните езици за програмиране. Реализацията на компонентен дебъгер, *IDD*, предоставя общоприложима, отворена, разширяема, гъвкава, слабо свързана платформа за инспектиране и анализиране чрез интерактивни и динамични способности. Дискутират се и няколко съществуващи приложения на дебъгера в изследователски проекти. Описани са и други прототипи, базирани на теоретичната рамка като *SolidReflector* и *DataMorphose*.

Работата предлага алгоритъм за диференциален динамичен анализ. Реализацията му в *IDD* показва неговото използване при анализ на големи и сложни софтуерни системи.

В заключение се излагат направените приноси, подкрепени от публикации, участия в конференции, семинари и др.

1.1 Актуалност на тематиката

През последните десетилетия софтуерните системи стават все по-сложни и по-големи. Това влияе негативно върху качеството им и възможността за реализиране на нова функционалност. Развитието на езиците за програмиране (ЕП) цели намаляване на сложността на програмите и подобряването на производителността на програмистите чрез въвеждането на различни парадигми за програмиране като обектно-ориентирано програмиране (ООП). В последните години се създават немалко визуални ЕП (ВЕП) като голяма част от тях са домейн-специфични.

Нерядко се срещат системи, състоящи се от няколко милиона реда програмен код, който се разработва от стотици програмисти. Често промени в един компонент предизвикват неочаквано поведение другаде, което е много трудно да бъде изолирано и възпроизведено [1]. Минимално описание за възпроизвеждане на проблем често се използва като средство за комуникация между програмисти или групи от програмисти. Този подход намалява дискусии дали новото поведение е проблем или нова функционалност. Описанието за възпроизвеждане на нововъзникналия проблем (регресия) прави проблема очевиден и понякога поправянето му е тривиално. С други думи, времето отделено за изолирането и описанието по възпроизводим начин на проблема отнема повече усилия отколкото отстраняването му. За разлика от времената, когато хардуера е по-скъп от работната ръка, последното десетилетие описва стабилна тенденция за евтин хардуер и скъп човешки труд за разработка на софтуер. Тази тенденция постави производителността на програмистите като важно поле за изследвания в академичните и индустриални среди. Широко разпространено е схващането, че голяма част от времето за разработка е съсредоточено при откриването и отстраняването на софтуерни проблеми в съществуващи системи, което оскъпява поддръжката им [2]. Отстраняването на проблеми (на програмистки жаргон дебъгиране) е времеемко начинание, което е недобре обезпечено от съществуващите инструменти.

1.2 Цели и задачи на дисертационния труд

Основна цел на дисертационния труд е да се създаде модел и прототип на инструмент за статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране

За постигането на това са дефинирани следните подцели и задачи:

- Ц1** Да се създаде концептуален модел за създаване на интерактивна софтуерна система, осъществяваща статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране. Задачи:
- Ц1.1 Да се конструира концептуален модел;
- Ц1.2 Да се дефинира архитектура.
- Ц2** Да се покаже приложимостта на предложения концептуален модел в областта на динамичния анализ на софтуер. Задачи:
- Ц2.1 Да се създаде алгоритъм, извършващ динамичен диференциален анализ, базиран на предложения модел.
- Ц2.2 Да се създаде прототип на инструмент, извършващ динамичен анализ, базиран на предложения алгоритъм;
- Ц3** Да се покаже приложимостта на предложения концептуален модел в областта на статичния анализ на софтуер. Задачи:
- Ц3.1 Да се създаде прототип на инструмент, извършващ различни видове статичен анализ, базиран на предложения модел.
- Ц4** Да се покаже приложимостта на предложения концептуален модел в различни домейн-специфични области. Задачи:
- Ц4.1 Да се създаде прототип на инструмент, който да извършва интерактивен домейн-специфичен динамичен анализ на системи на базата на изпълним код чрез модели от различни ниво на абстракция;
- Ц4.2 Да се визуализират съставените модели – CIL, control flow graph, call graph.
- Ц5** Да се покаже приложимостта на предложения концептуален модел във визуалния анализ на данни. Задачи:
- Ц5.1 Да се създаде прототип на инструмент, извършващ визуален анализ на данни;
- Ц5.2 Да се визуализират съставените модели на данните.

Реализационни цели на работата:

- **Общоприложимост** – основополагащите концепции и принципи трябва да са приложими върху широк набор от приложни области;
- **Отвореност** – способността на инструмента да бъде разширяван в едно или повече направления;
- **Разширимост** – способността да бъде добавяна нова функционалност чрез модули;

1.3 Структура на дисертационния труд

Настоящият дисертационен труд е организиран в няколко глави както следва:

- Глава 1, “Увод”, въвежда в проблемната област като поставя проблеми свързани с анализа на сложни софтуерни системи, откриване на грешки и др.
- Глава 2, “Обзор на проблемната област”, проследява развитието на проблемната област, чрез изследване на подобни системи и извършва обстойно сравнение между тях. Преглежда различни методи за визуализация и визуални езици за програмиране.
- Глава 3, “Модел на инструмент за статичен и динамичен анализ в домейн-специфични визуални езици за програмиране”, навлиза в дълбочина на основополагащите концепции за създаването на теоретичен модел на широкоприложим инструмент за извършване на статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране. Представя алгоритъм за динамичен диференциален анализ на две версии на една програма.
- Глава 4, “Архитектура и прототипна реализация на инструмент за статичен и динамичен анализ в домейн-специфичните ВЕП”, дискутира покомпонентно базираната архитектура на прототип на инструмент за статичен и динамичен анализ в домейн-специфични визуални езици за програмиране. Описва реализацията на механизми за визуално диференциално дебъгване в контекста на разработените инструменти.
- Глава 5, “Приложения и резултати”, описва няколко прототипни реализации на инструменти в различните подобласти на анализ на софтуерни системи.
- Глава 6, “Заключение”, формулира цялостните приноси и резултати от работата. Изброява направените от автора постижения в областта на дисертацията.

Почти всяка глава започва с кратко описание на нейното съдържание, последвано от аргументирано и добре развито изложение и завършва с анализ под формата на изводи.

Кратко съдържание на дисертационния труд

Глава 2 Обзор на проблемната област

Настоящата глава развива мотивацията на дисертационния труд по-детайлно. Проследява развитието на областта, чрез дефиниране на видовете анализ на съществуващи софтуерни системи с цел идентификация на софтуерни грешки и др. Дефинира фундаментални за работата понятия като *програма*, *визуално програмиране*, *програмна визуализация*, *визуален език за програмиране*, *статичен анализ*, *динамичен анализ*, *интерактивно и диференциално дебъгване*. Прави обстоен преглед и сравнение на съществуващи системи. Преглежда различни методи за визуализация и визуални езици за програмиране.

През последните десетилетия софтуерните системи продължиха да се развиват както като размер така и като сложност. Не е необичайно за съвременните софтуерни системи да превишават няколко милиона реда код и тази система да е разработена от широка общност от програмисти. Промените в един компонент могат да доведат до непредвидени последствия в поведението на други компоненти в системата и тези странични ефекти са трудни за идентифициране и пресъздаване (*reproducing*). Изолирането на минимален пример, пресъздаващ непредвиденото поведение често се използва като средство за комуникация между екипите за разработка, за да може ефективно да се стигне до определянето дали страничният ефект е бгг или нова функционалност. Когато достатъчно информация е налице, тогава справянето с бгга обикновено е тривиална задача.

2.1 Основни понятия

Дефиниции на основни понятия:

- програма – “множество оператори, които могат да бъдат разглеждани като модул от гледна точка на КС и използвани за насочване на нейното поведение.” [5, приблизителен превод]. Изчислимостта на “всичко” не е задължителна. Системата трябва да поддържа възможността за обработка на итерационни и условни оператори, най-малко неявно;
- визуално програмиране – се отнася до всяка система, която позволява на потребителя да обособи програма в две (или повече) измерения. Конвенционалните текстови езици не се възприемат като двумерни, тъй като компилаторът или интерпретаторът ги обработва като един едномерен низ. Визуалното програмиране включва конвенционални блок схеми (flow charts) и графични езици за програмиране. То не включва системи, които използват конвенционални (линейни) ЕП да дефинират изображения. Това елиминира повечето графични редактори, като Sketchpad [6];
- програмна визуализация – това е напълно различна концепция от визуалното програмиране. При визуалното програмиране, графиката е самата програма, а при програмната визуализация, програмата е описана на конвенционален текстови ЕП и графиката се използва за илюстриране на някой аспект на програмата или нейното изпълнение. За съжаление, в миналото, много системи за програмна визуализация са били некоректно определени като визуално програмиране. Системите за програмна визуализация могат да бъдат разположени върху две оси: дали илюстрират кода или данните на програмата и дали са *статични* или *динамични*. Динамични се отнасят до системи, които показват анимации на изпълняващата се програма, а статични – набор от снимки на състоянието (snapshot) на програмата в определени времеви точки. Ако програма създадена посредством визуално програмиране трябва да се визуализира или дебъгира, това не би следвало да се счита за програмна визуализация. Въпреки че тези два термина са близки и обхващащи, те се използват широко в литературата и би било добре да продължат да се използват по този начин;
- визуален език за програмиране – език за програмиране, съдържащ конструкции, които са или наследствено (inherently) или с наложено (superimposed) многомерно представяне[приблизителен превод];
- визуална среда за програмиране – интегриран набор от инструменти, които поддържат редактиране, анализ и изпълнение на визуални програми [7, приблизителен превод].

Дефиниции на разгледаните видове анализи:

- статичен анализ – представлява извличане на семантична информация за програмата чрез инспекция на форма на кода. Освен по време на трансляция [9] статичен анализ може да бъде извършван и без самата система да се обработва от изпълнител [10];
- динамичен анализ – представлява метод за анализ на поведението на софтуерни системи по време на тяхното изпълнение. Анализът включва изпълнение на системата и включва извършване на допълнителни операции като инструментiranje (instrumentation) и инспектиране на вътрешните състояния на системите [31];
- делта дебъгиране – представлява методология за откриване на грешки чрез поетапно отстраняване на части от програмата. Обикновено за база се използва тестов пакет с неуспешен тест, който служи за основа, над която ще се извършва отстраняването на частите от програмата, които не влияят на неочакваното поведение;
- диференциално дебъгиране – представлява сравнителен анализ между външни или вътрешни състояния на две различни версии на софтуерна система. Този метод на анализ се използва за локализиране на регресивни грешки;
- интерактивно дебъгиране – представлява средство за динамичен анализ на програми с цел ръчно откриване и отстраняване на грешки. Анализът дава най-добри резултати в случаите когато целевата програма е компилирана заедно с допълнителна помощна информация, наречена дебъг символи (debug symbols);
- инструментiranje – представлява модификация на програма с цел събиране на информация за нейното изпълнение. Поставянето на инструментиращ код може да се извърши на ниво сорс код и на ниво машинен код. В зависимост от ситуацията инструментиращ код може да бъде поставен по време на компилация от компилатора или от свързващия редактор или от външни инструменти – под формата на машинни инструкции [50];
- производителност – представлява форма за определяне на ефективността на програмата, която тя развива върху дадена ВИМ (виртуална изчислителна машина). Ефективността се измерва чрез метрики като бързодействие и консумация на ресурси като оперативна памет, физически размер, енергоефективност.

2.2 Сравнителен анализ

Сравнителната характеристика има за цел да обърне внимание на определени страни на разглежданите системи за локализиране на грешки и да оцени (понякога субективно) характеристики като:

- отвореност (О) – оценява възможността и лекотата за добавяне на нова, потребителски дефинирана функционалност;
- статичност (СД) – възможността за извършване на статичен анализ;
- динамичност (ДД) – възможността за извършване на динамичен анализ;
- интерактивност (И) – степента на развитие на диалога между потребителя и системата;
- лиценз (Л) – режим на ползване;
- визуалност (В) – степента на развитие на визуалното взаимодействие с потребителя;
- домейн-специфичност (ДС) – специализация в особена проблемна област;
- различни нива на абстракция (А) – възможност за работа с мултимодели;
- диференциращ подход (Ди) – възможност за сравнение между базова и референтна модификация.

Име	О	СД	ДД	И	Л	В	ДС	А	Ди
Coverity	+	+	–	~	~	+	–	–	–
C-reduce	–	+	+	–	+	–	–	–	–
DBDB	–	–	+	~	+	–	+	–	+
Delta	–	+	+	–	+	–	–	–	–
DPDebugger	–	–	+	–	+	–	+	–	+
ESC/JAVA	–	+	–	–	+	+	–	–	–
Gcov	–	–	+	–	+	–	–	–	–
GNU GDB	+	–	+	–	+	+	–	–	–
Visual Studio Debugger	–	–	+	+	–	+	–	–	–
NDepend	+	+	+	–	+	–	–	+	–
PolySpace	–	+	–	–	–	+	–	–	–
RADAR	–	–	+	–	+	+	–	–	+
Strace	–	–	+	–	+	–	–	–	–
Perf	–	–	+	–	+	–	–	–	–
Valgrind	+	–	+	–	+	–	+	–	–
MPCGO	+	–	–	+	+	+	+	+	–

Таблица 2.1: Сравнителен анализ между разглеждани системи за локализиране на грешки

Легенда: да: +; не: –; частично: ~;

Проучени са над 80 на брой системи за анализ на софтуер, но те не предоставят достатъчно добри механизми за визуализация на данните и/или изпълнението. Решението на този проблем е свързан с интегрирането на домейн-специфични способности за визуализацията на данните и изпълнението на програмата. Разгледани бяха над 100 визуализационни метода. За част от тях бяха описани основните им области на приложение във визуализацията при анализ на софтуер. В допълнение са разгледани и класифицирани визуални среди от различни поколения.

2.3 Изводи и перспективи

Направеният обзор изследва широк спектър от инструменти за анализ в разнообразни проблемни области. От направения обстоен преглед на системи за извършване на анализ се очертават няколко предизвикателства, свързани с общоприложимостта, отвореността, визуалността.

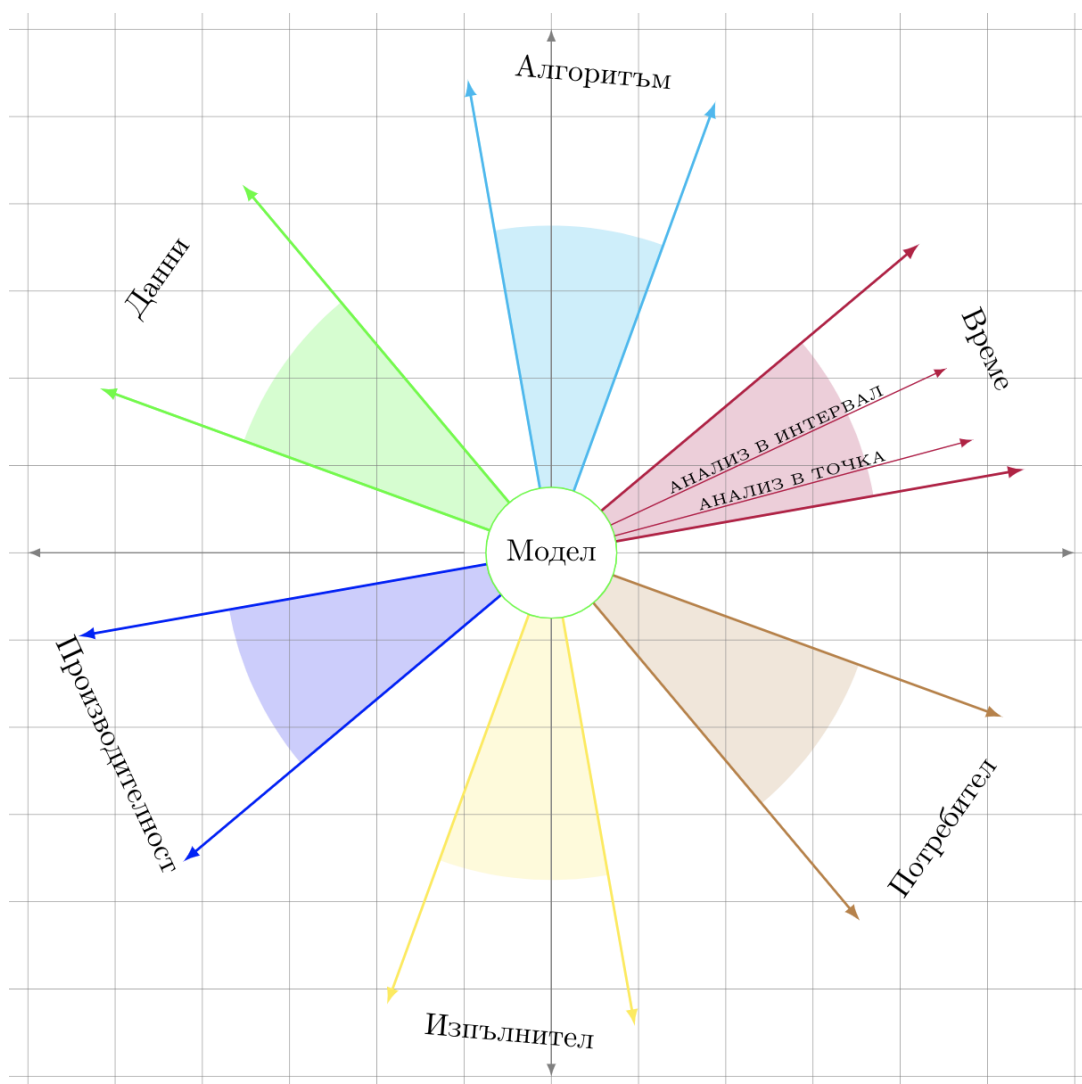
Глава 3 Модел на инструмент за статичен и динамичен анализ в домейн-специфични визуални езици за програмиране

Настоящата глава описва теоретична основа, целяща да подпомогне и улесни реализацията на инструмент за статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране. Дефинира се отворен модел за реализация на инструмент обработващ разнородна информация от общи и специфични целеви области. Аргументира използването на архитектура, базирана на компоненти. Въвежда основни подобласти като диференциално дебъгиране. Представя алгоритъм за динамичен диференциален анализ на две версии на една програма.

3.1 Основен йерархичен многомерен модел

Предложеният модел може да се разглежда като съставен йерархичен модел в многомерното пространство. По всяка ос се описват абстрактни и конкретни характеристики за специфичен информационен домейн. Използването на модел, включващ в себе си разнородна информация се обуславя от дефинираното условие за широкоприложимо приложение на модела в изследването на софтуерни системи. Информация за програмата се извлича чрез модели на кода

(визуален или текстов), както по време на компилация така и по време на изпълнение. Схематично осите на многомерния модел са показани във Фигура 2.1.



Фигура 2.1: Йерархичен многомерен модел.

По различните оси могат да се нанасят различни видове характеристики, описващи модела:

- времева – всички измервани и изчислявани характеристики обикновено са актуални (измерени) в даден момент или интервал от времето;
- алгоритми – различни аспекти на реализирания алгоритъм под формата на код в различни представяния, абстрактни нива и др.;
- данни – мултимоделно представяне на програмните данни. Например стек на извикванията, стек фрейм, локални променливи, регистри и др.;

- количествени метрики – за измерване на производителност (консумация на памет, енергия, размер, бързодействие и др.) и др.;
- качествени метрики – измерване на комплексност на алгоритъма (цикломатична комплексност, брой редове, изпълним код и др.) и др.;
- изпълнител – информация за различни параметри на ВИМ като възможност за паралелност, физически машини, виртуални машини, хетерогенни машини и др.;
- потребител – директни или индиректни измервания на физическото състояние и поведение на потребителя при разработка (разработчик) и използване (краен потребител) на програмата.

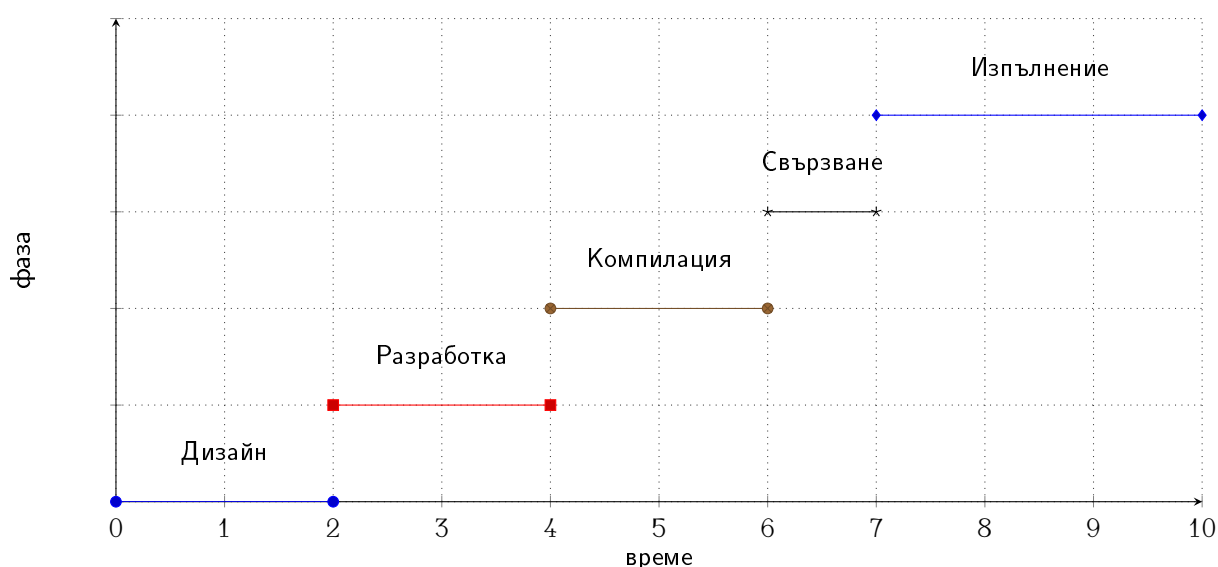
3.2 Статичност и динамичност – функция на времето

Времето е критичен компонент от модела при статичния, динамичния и производителния анализи на системата и представлява основна компонента от многомерния йерархичен модел на системата. Времето е неразделна част от развойната дейност при създаването на софтуер и в различните фази могат да се извършват редица анализи на софтуерния продукт. Жизненият цикъл на създаване и използване на програма, чрез ЕПВН може да се раздели във фазите дизайн, разработка, трансляция, свързване, зареждане, изпълнение и др. В зависимост от конкретната фаза от цикъла, в който разглеждаме програма, могат да се извършват анализи по време на:

- дизайн – визуалните езици и среди за програмиране предоставят графични потребителски интерфейси, чрез които може да се програмира поведението на системата посредством визуално описание на поведението на отделните структури на програмата;
- разработка – основна фаза от разработката на софтуер, при която разработчикът реализира алгоритъма, чрез който програмата обработва входни данни. Съвременните среди за програмиране са в състояние да извеждат диагностики за зададени метрики на кода в реално време;
- трансляция/компиляция – фазата на превод от език за програмиране от високо ниво до машинен език. Голяма част от транслаторите реализират силно развити оптимизатори, които работят на базата на статични анализи на подадените входни програми.
- свързване – събиране на програмни модули посредством свързващ редактор. Тази фаза също е подходяща за оптимизации, но извършващи се на по-високо ниво на абстракция;

- изпълнение – зареждане и изпълнение в средата на ВИМ, за която е предназначена програмата. Съществуват системи за откриване на грешки (дебъгери), които са в състояние да се включат към изпълняваща се програма (attaching) и да поемат контрола над управлението на програмата. Това позволява да се прочита вътрешното състояние, което е основата на динамичния анализ.

Обща схема на фазите от жизнения цикъл на програмите е описан чрез Фигура 2.2. Разработването на софтуерната система се разглежда като функция в двумерното пространство, където по оста време се разполага времето, докато по оста фаза се онагледява конкретното състояние на софтуерната система в термините на нивото на нейната завършеност.



Фигура 2.2: Разпределение на фазите във времето при компилиране

В зависимост от типа на транслационния процес (компиляция, интерпретация) на езика за програмиране (или ВЕП) фазите могат да се припокриват частично или изцяло във времето. Времето е основната характеристика в предложения модел, която внася разделение между статичния и динамичния анализ. Анализите в точка от времето се разглеждат като статични поради фактическото прекъсването на работа на програмата. В дисертационния труд са разгледани и типове анализи на база на интервали от времето. Описани са приложенията им в изследването на изменението на метрики и начините, по които се обработва информацията за тях, като агрегация и филтрация и други техники, спомагащи при анализа на данните от изпълнението в различни домейни и на различни нива на абстракция.

3.3 Методи за извличане на информация и работа на системата

Поради естеството на типовете анализи, статичните методи за анализ разглеждат максималното пространство от състояния, в които вътрешното състояние на алгоритъма може да попадне по време на изпълнение. Това е така, защото при статичен анализ не са известни точните стойности на данните на променливите и параметрите на функциите, а само диапазоните на типовете им и други ограничения. За разлика от статичните динамичните действат по време на изпълнение на програмата и имат пълна информация за точните стойности на данните, което означава, че разглежданото пространство от стойности на вътрешното състояние при динамичния анализ на алгоритъма е винаги подмножество на пространството по време на статичен анализ за съответния параметър.

Софтуерните метрики, които се извличат посредством статичен анализ могат да бъдат количествени като броя на редовете код, полета, методи, променливи, параметри, дълбочина на наследниците (depth of inheritance), но могат да бъдат и качествени. Динамичните методи за извличане на данни включват операции за извличане по време на изпълнение на програмата. За целите на текущото изследване са избрани и разгледани някои класически техники за статичен и динамичен анализ.

3.4 Визуализация на алгоритми и данни

Съществуват сложни софтуерни системи, при които класическия анализ на алгоритмите и данните им е практически неприложим. Това са проекти, които работят освен с огромен обем междинни данни, така и с бързоизменящи се такива, което обуславя необходимостта от използването на ускоряващи структури. Дискутирана е приложимостта на поэтапното визуализиране на алгоритмите и данните. Например при постъпково изпълнение на програмата (line by line) е възможно постепенното визуализиране на алгоритъма като по този начин да се проследи еволюцията на изпълнението. Резултатната визуализация е възможно, но не е задължително, да представлява ВЕП.

3.5 Алгоритъм за диференциално дебъгиране (сравняване)

За да се извърши домейн-специфичен анализ на целева софтуерна система с предишна нейна версия във времето се налага използването на концепцията

диференциално дебъгиране. Интерактивната диференциална дебъгираща система трябва да може да съгласува и синхронизира изпълнението на външен и вътрешен анализ на две версии на програма. Нека дефинираме алгоритъма за диференциално дебъгиране на изпълнението на две системи:

Algorithm 1 Алгоритъм за диференциално дебъгиране

```

procedure DiffDebug( $a, b$ ) ▷ The differential debugging
  while  $\exists \langle SP_1, OP_1 \rangle \leftarrow P_1(IP_1) \vee \exists \langle SP_2, OP_2 \rangle \leftarrow P_2(IP_2)$  do
    for each  $i \in 0 \dots N$  do
      if  $\mu(D_i P_1(\langle SP_1, OP_1 \rangle), D_i P_2(\langle SP_2, OP_2 \rangle)) > \epsilon$  then
        Visualize( $\Delta(\langle SP_1, OP_1 \rangle, \langle SP_2, OP_2 \rangle)$ )
      end if
       $E_i \leftarrow E(D_i P_1(\langle SP_1, OP_1 \rangle), D_i P_2(\langle SP_2, OP_2 \rangle))$ 
      if  $E_i \neq \emptyset$  then
        Notify( $E_i, D_i P_1(\langle SP_1, OP_1 \rangle), D_i P_2(\langle SP_2, OP_2 \rangle)$ )
      end if
    end for
  end while
end procedure

```

Нека P_1 и P_2 са две версии на дадена програма.

Нека IP_k и OP_k са входните и изходните данни на $P_k, k \in \{1, 2\}$.

Нека TP_k е преход от IP_k до OP_k (означено като $IP_k \rightarrow OP_k, k \in \{1, 2\}$).

Нека SP_k е множество вътрешни състояния с преход $IP_k \rightarrow OP_k, k \in \{1, 2\}$. μ е теглова функция за откриване на разлики, където теглото зависи от домейна D_i . ϵ представлява домейн-специфичен стойност за допустимост, чрез която се определя дали разликата между двете програми трябва да се визуализира. Δ е функция за откриване на разлики в данните.

Нека E_i е списък със събития, активирани при наличие на разлики (divergence events). N е броят на инструментите за анализ. $D_1 P_1 \dots D_N P_1$ е броя на инструменти за анализ на P_1 , които предоставят информация за $\langle SP_1, OP_1 \rangle$ и $\langle SP_2, OP_2 \rangle$, а $D_1 P_2 \dots D_N P_2$ са същото множество от инструменти за анализ, но за P_2 .

Забележка 1: $D_N P_1 = D_N P_2 \Leftrightarrow IP_1 = IP_2 \wedge OP_1 = OP_2 \wedge (SP_1 = SP_2) \vee (SP_1 \approx SP_2)$; $SP_1 \approx SP_2 \Leftrightarrow SP_1 \neq SP_2 \wedge SP_1 \triangleq SP_2$, където $\triangleq$ е потребителско дефиниран предикат.

Забележка 2: В тези термини можем да дефинираме регресия като $IP_1 = IP_2 \wedge TP_1 \neq TP_2 \wedge OP_1 \wedge OP_2$ и възпроизводимост като множеството SP_2 където $OP_1 \neq$

OP_2 за краен брой IP_2 . Минимална възпроизводимост е $\min(SP_1)$ и $\min(SP_2)$ за $OP_1 \neq OP_2$. Обикновено се постига чрез минимизация на IP_2 стъпка по стъпка.

Забележка 3: Инструментът е зависим от времето тъй като запитванията към него са в определен момент по време на изпълнението. За базираните на *trace* инструменти, това време се дефинира от точката на извикване в програмата. За интерактивните инструменти това е точката където изпълнението се прекъсва и се инспектира текущото състояние.

3.6 Изводи и перспективи

Предложен е широкоприложим модел за събиране и анализиране на информация за програма. Моделът позволява дефинирането на архитектура за отворена система, в която лесно може да се добави използването на външни инструменти.

Тя позволява различни методи на анализ, от сравнения на относително прости метрики до диференциални сравнителни методи за анализ. Поддържат се статични и динамични способности за изследване на програмата и нейното поведение.

Моделът е силно повлиян, насочен и съобразен с приложението му в различни домейн-специфични области. Дефиниран е алгоритъм съчетаващ диференциален подход посредством динамичен анализ при откриване на широк спектър от регресионни проблеми.

Глава 4 Архитектура и прототипна реализация на инструмент за статичен и динамичен анализ в домейн-специфичните ВЕП

Обсъжда подробно архитектурата на няколко инструменти за статичен и динамичен анализ в домейн-специфичните визуални езици за програмиране. Обосновава архитектурните решения като детайлизира технически и реализационни особености. Изброява редица съпътстващи разработки на автора и изследователската група като *SolidOpt*, *SolidV* и *DataMorphose*. Описва реализацията на механизми за диференциално дебъгиране в контекста на разработените инструменти.

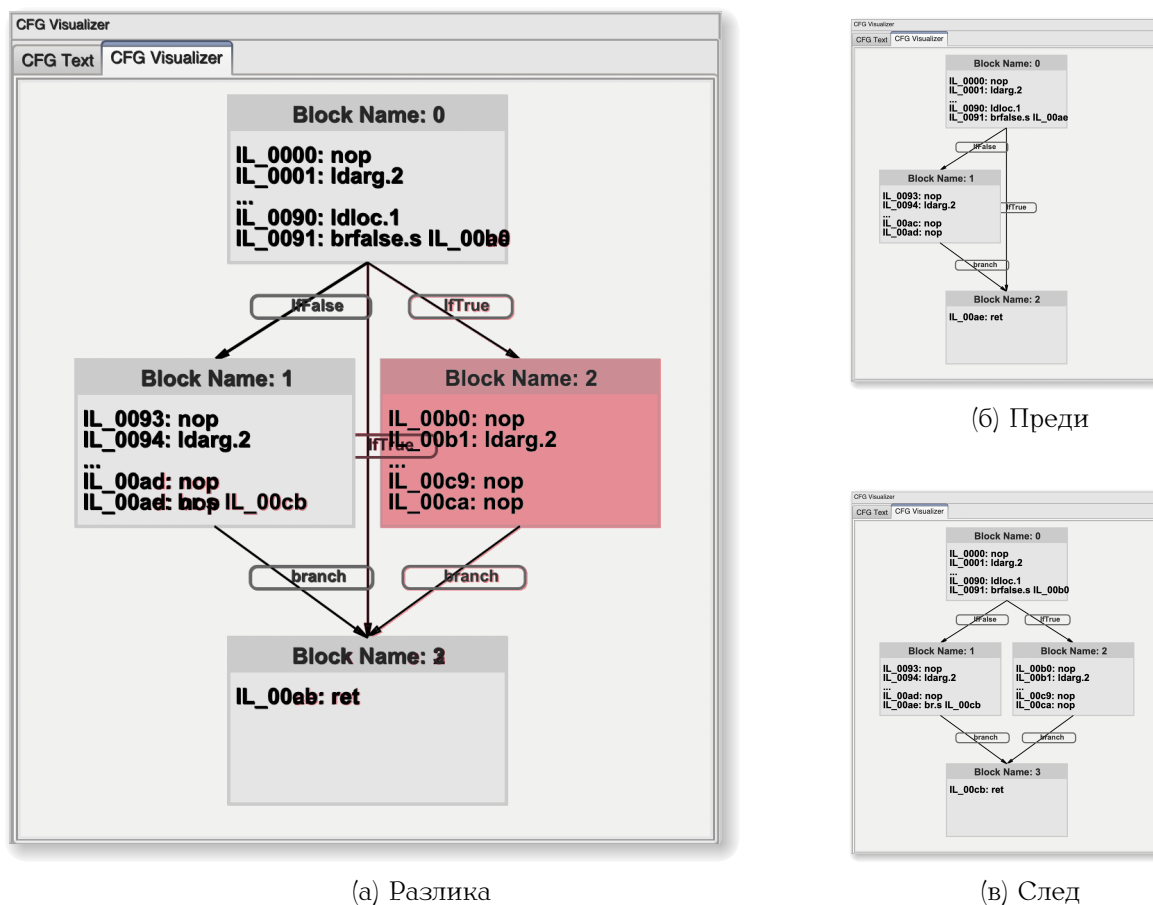
4.1 Съпътстващи разработки

Реализираният набор от инструменти използва широко разпространени архитектурни шаблони и техники за реализация, спомагащи за реализирането на отворени за добавяне на допълнителна функционалност системи. Използвани са технологии, улесняващи преносимостта на инструментите, така че да не се ограничават от потребителските операционни системи.

Прототипните реализации се базират на архитектурни решения, реализирани като част от предишни разработки на автора и екипа, разработващ веригата от инструменти (toolchain) SolidOpt.

4.2 Визуално диференциално дебъгиране

Диференциалното дебъгиране в областта на текстовите ЕП е удобен инструмент за откриване на регресии. В емпиричните ни изследвания се забелязва подобрена продуктивност при дебъгиране чрез използването на инструмента за диференциално дебъгиране IDD, показан чрез Фигура 2.6.

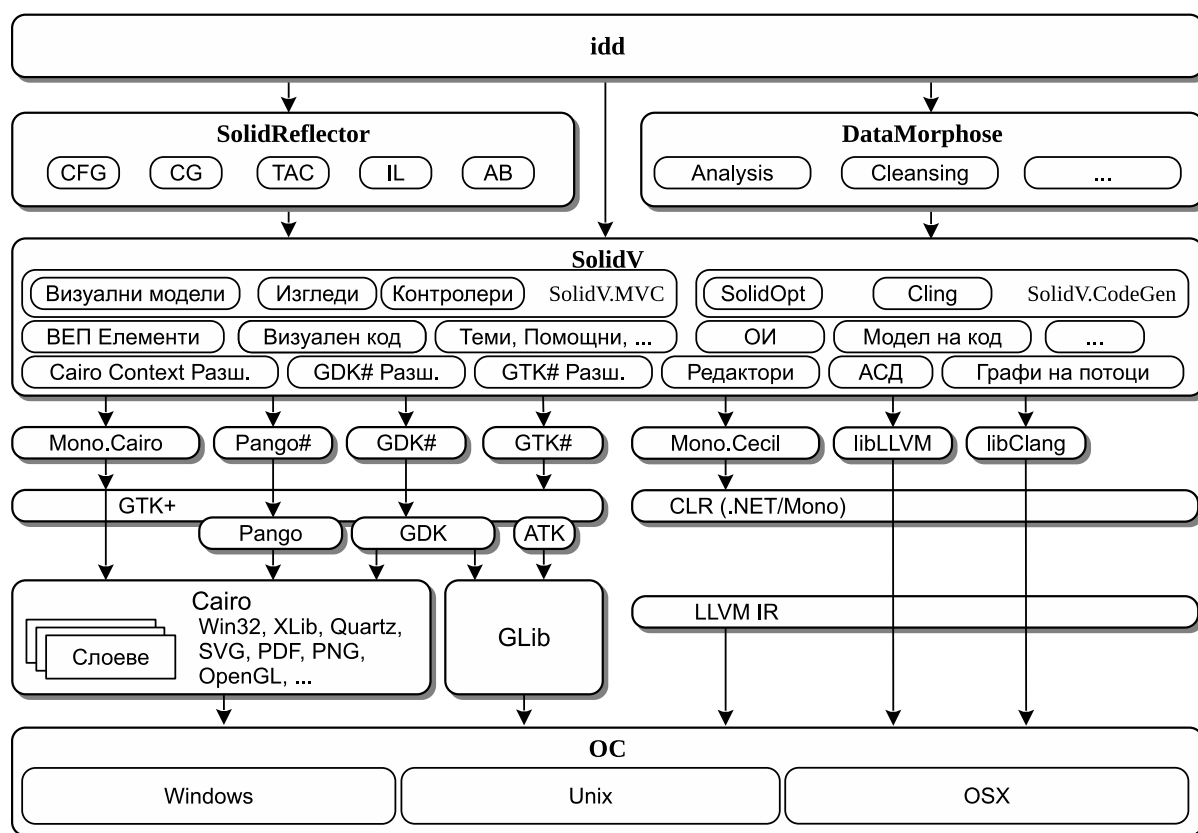


Фигура 2.3: Статично визуално диференциално дебъгиране на графа на потока на данни

Естествен въпрос е дали тази тенденция е подобна при ВС и дали диференциалното дебъгиране във ВС като цяло е възможно за реализация с предложения модел и разработения технологичен стек. Докато генерализиране на диференциалното дебъгиране до произволен брой канали е практически невъзможно с настоящия технологичен стек, е възможно да се приложи за няколко канала. Подобно на ДРАКОН, ВС SolidReflector позволява обособяване на програма в два канала – текстови (инструкции за виртуалната машина CLR) и свързаност на базовите блокове показващи потока на изпълнение. На фигура 2.3 SolidReflector анализира собствения си поток от данни. Фигура 2.3б и 2.3в показват как инструмента може да се “саморедактира” и промени своя поток от във своята изолирана среда.

4.3 Използвани технологии и политики за разработка

Избраните технологии и тяхната връзка с архитектурата на прототипната реализация може да се изобрази схематично чрез Фигура 2.4.



Фигура 2.4: Технологичен стек на IDD и взаимодействието със SolidReflector и SolidOpt

4.4 Изводи и перспективи

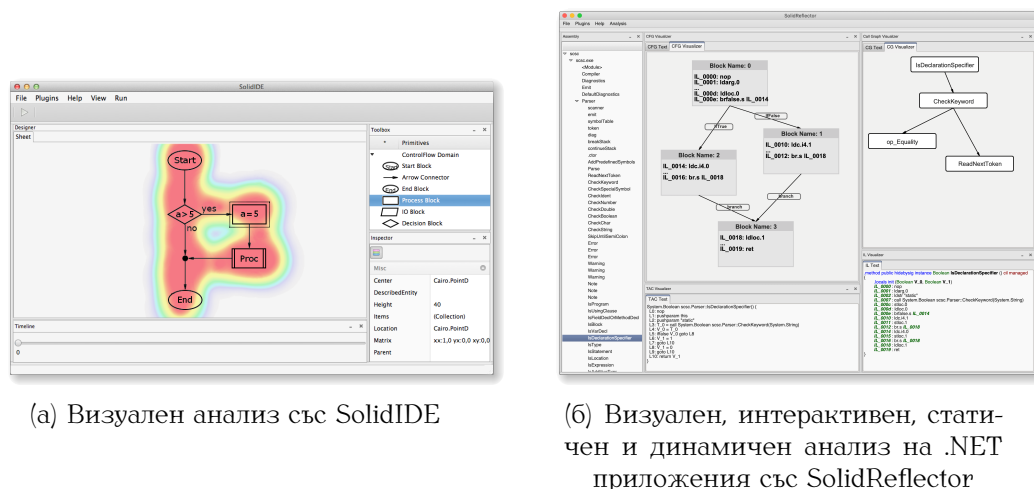
Предложената обща архитектура е достатъчно гъвкава и може да се използва за база за създаване на интерактивни визуални анализатори, извършващи статичен и динамичен анализ. Описаните в главата методологии за извършване на визуален анализ на данни и диференциално дебъгиране са изследвани в детайли в Глава 5.

Глава 5 Приложения и резултати

Главата описва прототипни проекти, които реализират концептуалния модел и следват насоките на архитектурата. Моделът и архитектурата предполагат няколко приложения и резултати в областта на тематиката. Те са в различни области свързани с тематиката – визуализация и дебъгиране чрез статичен и динамичен анализ в домейн-специфичен ВЕП чрез *SolidOpt/SolidV*; и дебъгиране чрез диференциален, статичен и динамичен анализ в ЕП с общо предназначение и ВЕП.

5.1 Визуален, домейн-специфичен статичен и динамичен анализ със SolidIDE, SolidReflector и др.

Множество прототипи са разработени в различни домейн-специфични области (това включва както статичен, така и динамичен анализ).



Фигура 2.5: Визуален, домейн-специфичен статичен и динамичен анализ със SolidIDE и SolidReflector

Във Фигура 2.5а е показан прототипът SolidIDE, който използва домейн-специфична визуализация на статични или динамични анализи. Визуализационният метод на фигурата е *heat map* и той може да бъде използван за анотации както на статични така и на динамични анализи. Във Фигура 2.5б е показан прототипът SolidReflector за визуален, домейн-специфичен статичен и динамичен анализ на .NET приложения.

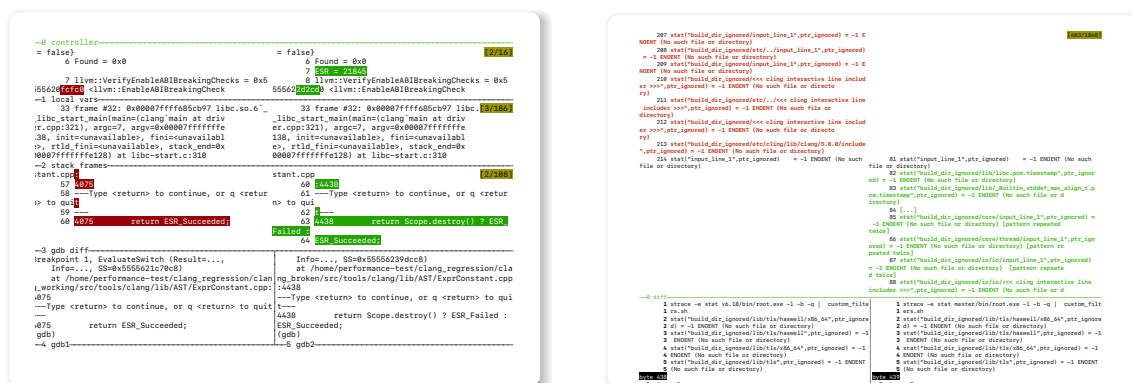
5.2 Домейн-специфично диференциално дебъгиране с IDD

Диференциалното дебъгиране представлява статичен и динамичен сравнителен анализ на две версии на една и съща програма. Този тип анализ позволява по-бързото откриване на регресии – нови проблеми възникнали от разработката на следващата версия на програма.

Създаденият прототип за диференциално дебъгиране, *IDD*, е в състояние да анализира сложни системи. Беше използван за анализ на възникнала регресия в компилаторът *clang*. Анализирана е регресия в производителността на системата за анализиране на данни в областта на физиката на високите енергии *ROOT* [107]. Също така беше демонстрирано как се намират регресии в производителността при крехката автоматична векторизация в примерна програма.

5.2.1 Текстови езици за програмиране

PR43674 [108] представлява доклад за функционална регресивна грешка в компилатора *clang* и по-конкретно в *constexpr* интерпретатора, заедно с включен като допълнение код за възпроизводимост. Кодът за възпроизводимост обхваща около 30000 реда нетривиален код.



(а) Анализ на функционална регресия в компилатора Clang

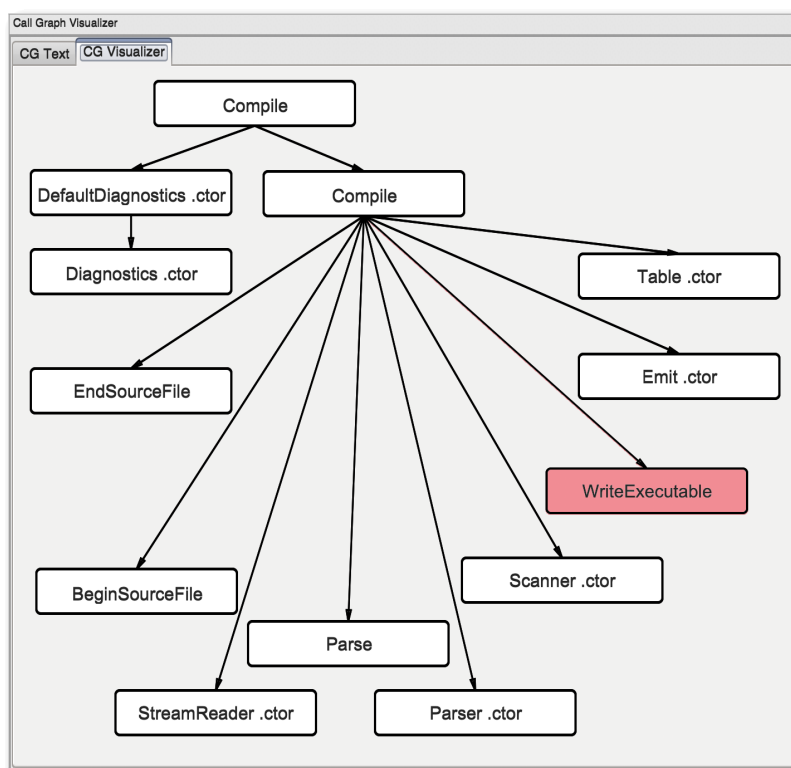
(б) Анализ на регресия в ефективност на изпълнение в ROOT

Фигура 2.6: Домейн-специфичен статичен и динамичен анализ със IDD

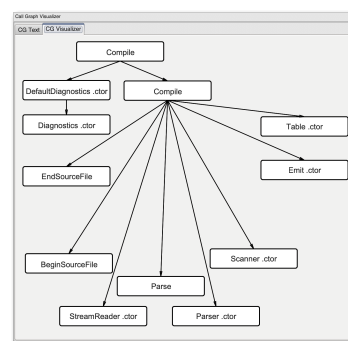
Интеграцията между *IDD* и *strace* е демонстрирана на Фигура 2.66. Използвани са две версии на системата *ROOT* – v6.18 и актуалната към дата 02.01.2020 версия *master*. Анализирани са времеемкото системно извикване *stat* за инициализирането и деинициализирането на *root.exe*. От резултатите са видни извикванията на *stat*, които връщат като резултат *ENOENT* (*No such file or directory*). Оказва се, че: *ROOT v6.18* извършва 251 извиквания, а *ROOT master* извършва 3522 извиквания.

5.2.2 Визуални езици за програмиране

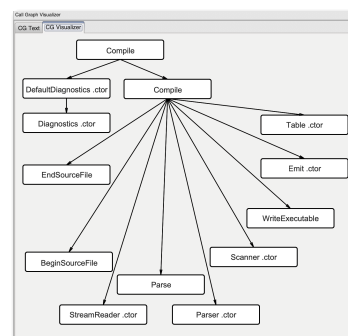
Взаимодействието между *SolidReflector* и *IDD* е полуавтоматизирано. Експортираното статично изображение от ВЕП в *SolidReflector* се анализира от *IDD*.



(а) Разлика



(б) Преди



(в) След

Фигура 2.7: Статично визуално диференциално дебъгиране на граф на извикванията

На Фигура 2.7 е анализиран графът на извикванията в демонстрационния компилатор *scsc*, използван в обучението по Методи на трансляция в Пловдивски университет „Паисий Хилендарски“.

Заклучение

Основен извод е, че дисертационният труд разглежда обстойно и предлага аргументирани решения на всяка една от поставените в секция 1.2 – Цели и задачи на дисертационния труд. По време на изследванията на проблемната област бяха направени няколко приноса, описани по-подробно в следващите подсекции.

3.1 Резултати и приноси

В настоящата секция се описват някои от приносите, които са свързани пряко или косвено с дисертационния труд или някои негови реализационни аспекти. Приносите под формата на публикации са по-тясно свързани с теоретичната част, а приносите под формата на презентации – с реализационната част на дисертацията. При решаването на поставените цели бяха получени основни научно-приложни и приложни приноси, показани в Таблица 3.1.

3.1.1 Публикации по дисертационния труд

- C1** Vassilev M., Vassilev V., Penev A., “IDD – A Platform Enabling Differential Debugging”, The journal of Cybernetics and Information Technologies, 14 pp., (SJR2018 = 0.215) (in print);
- C2** Vassilev V., Penev A., **Vassilev M.**, “SolidOpt – A Multi-Model Software Optimization Framework”, International Journal of Computer Science Issues, Issue March 2nd 2015, pp. 32-41, [ISSN (Print): 1694-0814, ISSN (Online): 1694-0784];
- K1** Vassilev V., **Vassilev M.**, Petrova P., “SolidReflector: A Multistage, Interactive Decompilation Framework”, in Proceedings of the International Conference “From DeLC to Velspace”, Plovdiv, 26-28 March 2014, pp. 49-58 [ISBN 0-9545660-2-5];
- K2** Vassilev V., Petrova P., **Vassilev M.**, “DataMorphose: An Interactive Data Migration Framework”, in Proceedings of the International Conference “From DeLC to Velspace”, Plovdiv, 26-28 March 2014, pp. 59-68 [ISBN 0-9545660-2-5];

#	Резултати	Цел	Секция	Публикации
1	Дефиниран е концептуален модел за създаване на интерактивна софтуерна система, осъществяваща статичен и динамичен анализ в домейн-специфични езици за програмиране	Ц1	Глава 3	С1, С2
2	Дефиниран е алгоритъм на интерактивен диференциален дебъгер, извършващ динамичен анализ на базата на предложения концептуален модел	Ц2	Глава 3	С1
3	Разработен е прототип на интерактивен диференциален дебъгер IDD, извършващ динамичен анализ върху взаимосвързани версии на голяма и сложна софтуерна система, базиран на дефинирания алгоритъм	Ц2	Глава 4, Глава 5	С1
4	Разработени са прототипи на: Инструмент SolidReflector за интерактивен домейн-специфичен динамичен и статичен анализ на системи, визуализиращ изпълним код чрез модели от различно ниво на абстракция; Инструмент DataMorphose за домейн-специфичен анализ, визуализация и миграция на данни;	Ц3, Ц4, Ц5	Глава 5	К1, К2

Таблица 3.1: Връзка между цели и приноси

3.1.2 Участия в проекти

Пловдивски университет „Паисий Хилендарски“:

- Научен проект СП15-ФМИИТ-007/24.04.2015 *“Надграждане на знания и формиране на компетенции у студентите за работа със специализиран математически софтуер за решаване на приложни математически задачи”*, – Фонд “Научна изследвания” при ПУ “Паисий Хилендарски”, “Научни изследвания” – 2015/2016 г.
- Научен проект НИ15-ФМИИТ-004/24.04.2015 *“Иновативни фундаментални и приложни научни изследвания по компютърни науки, математика и педагогика на обучението”* – Фонд “Научни изследвания” при ПУ “Паисий Хилендарски”, “Научни изследвания” – 2015/2016 г.

3.1.3 Доклади на конференции и семинари

- П1 Vassilev M., Vassil Vassilev, Alexander Penev, Petrova P., "HEPCon – A Cross-Platform Mobile Application for HEP Events", 23rd International Conference On Computing in High-Energy Physics, July 2018, Sofia, Bulgaria, представящ
- П2 Vassilev V., Vassilev M., Petrova P., "SolidReflector: A Multistage, Interactive Decompilation Framework", "International Conference From DeLC to Velspace", March 2014, Plovdiv, представящ

3.1.4 Аудиторни занятия във висши училища

Важна част от докторските програми е работата със студенти и развитието и подобряването на университетските програми за обучение. Чл. 25. (1) от ПРАС на ПУ насърчава приноса на докторантите в това направление. В този смисъл настоящият дисертационен труд допринесе за развитието на някои курсове:

- Методи на трансляция (упражнения), Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“;
- Създаване и обработка на видео и анимации, Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“;
- Анализ и оптимизация на софтуерни приложения (избираема), Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“;
- Графика и презентации (упражнения), Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“;
- Паралелно програмиране (упражнения), Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“;
- Компютърна Графика (упражнения), Факултет по математика и информатика при Пловдивски университет „Паисий Хилендарски“;
- Облачни и грид технологии, Бургаски свободен университет.

3.2 Изводи и перспективи

Настоящата работа подробно описва веригата инструменти в областта, SolidV, SolidReflector и IDD. Досега одобрената политика за развитие на SolidV се основава на еволюционно развитие на базата на потребителските нужди.

SolidReflector като инструмент тясно свързан със SolidV, би следвало да приложи подобна политика на разработка. Основни насоки за бъдещо развитие са

подобряване на визуализацията, потребителския интерфейс и добавянето на нови модели на представяне.

Авторът смята, че IDD е интересен от гледна точка на изследователски цели, тъй като позволява обединяването на информация от много източници. Това количествено натрупване води до възможността за предоставяне на качествено различни функционални елементи. Например:

- Метаточки на прекъсване – създаване на точки на прекъсване на базата на метасъбития като “място на разделяне” на последователността на изпълнение; разлика в данните на двете програми в началото на подпрограми и други;
- Подобряване на ефективността на интерфейса – с цел намаляване на усилията на разработчика до локализиране на проблема;
- Подобряване на средствата за откриване на семантични разлики при изпълнение;
- Реверсивно диференциално дебъгиране – при него програмата се изпълнява отзад напред като това позволява да се изследват по-сложни систематични ефекти при проследяване на потока от инструкции и/или данни.

Основните идеи в настоящия дисертационен труд бяха плод на усилена работа преди и по време на обучението за придобиване на образователна и научна степен “доктор”. Идеите за прилагането на архитектурата и алгоритъмът за диференциално дебъгиране се подобряваха с времето. Това може да се забележи в дипломните работи, съответно за придобиване на образователна степен бакалавър [111] и магистър [106] на автора, както и работата на други изследователи от екипа [112, 113, 71, 3, 114].

На базата на тези основи дисертационният труд успя да предложи генерално и добре реализирано решение на поставените цели. Успя да реализира и усъвършенства различни компоненти, които намериха приложение дори извън контекста на работата и академичната общност. Част от решенията бяха публикувани на международни конференции, а друга е в процес на рецензия за публикация. Реализираните приложения бяха качествено различни:

- SolidReflector – среда за анализ и визуален език за програмиране, ориентиран към потока инструкции (control flow) и др.;
- DataMorphose – визуален език за програмиране, ориентиран към потока данни (data flow);

- Домейн-специфичен визуален дебъгер – дебъгер, улесняващ откриването на проблеми в домейн-специфични условия в контекста на SolidReflector, показващ гъвкавостта и широката приложимост на метайнструментариума SolidV;
- IDD – диференциален дебъгер, улесняващ откриването на проблеми между различни версии на големи и сложни софтуерни системи.

Библиография

- [1] WONG, W. E. et al. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering*. 2016, vol. 42, no. 8, с. 707–740.
- [2] CHANG, S. K. (ed.). *Handbook of Software Engineering and Knowledge Engineering - Vol 3: Recent Advances*. World Scientific Publishing Co. Pte. Ltd., 2005.
- [3] ВАСИЛИЕВ, В. *Отворени, интерактивни и динамични визуални езици за програмиране*. Пловдив, България: Пловдивски Университет, 2015. Дисертация. Пловдивски Университет "Паисий Хилендарски". Дисертация. Пловдив, 2015. 163 л.
- [4] MYERS, B. A. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. в: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Boston, Massachusetts, USA: ACM, 1986, с. 59–66. CHI '86.
- [5] DAINITH, J. et al. *A Dictionary of Computing*. 6th ed. New York, NY, USA: OXFORD UNIV PR, 2008.
- [6] SUTHERLAND, I. E. Sketchpad: A Man-machine Graphical Communication System. в: *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*. Detroit, Michigan: ACM, 1963, с. 329–346. AFIPS '63 (Spring).
- [7] ROZENBERG, G. (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1997.
- [8] BURNETT, M. M. et al. A Classification System for Visual Programming Languages. *Journal of Visual Languages & Computing*. 1994, vol. 5, no. 3, с. 287–300.
- [9] LANDI, W. Undecidability of static analysis. *ACM Letters on Programming Languages and Systems*. 1992, vol. 1, no. 4, с. 323–337.
- [10] JACKSON, D. et al. Software analysis: a roadmap. в: *Proceedings of the conference on The future of Software engineering - ICSE '00*. ACM Press, 2000.
- [11] TOMASDOTTIR, K. et al. Why and how JavaScript developers use linters. в: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017.
- [12] SANTOS, A. L. D. et al. Using JavaScript Static Checkers on GitHub Systems: A First Evaluation. в: *III Workshop on Software Visualization, Evolution and Maintenance*. 2015.
- [13] AL., S. L. et. Django Web Framework Software Metrics Measurement Using Radon and Pylint. в: *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*. IEEE, 2018.
- [14] SubMain / CodeIt.Right / Code Quality Analysis, Automated Code Review and Refactoring. 2020-02-04. също дост. на: <https://submain.com/products/codeit.right.aspx>.
- [15] BANSAGHI, A. et al. Source code scanners in software quality management and connections to international standards. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae. Sectio Computatorica*. 2012.
- [16] CodeRush: IDE Productivity Tools for Visual Studio. 2020-02-04. също дост. на: <https://www.devexpress.com/products/coderush/>.
- [17] StyleCop. 2020-02-04. също дост. на: <https://github.com/StyleCop/StyleCop>.
- [18] Modern Mainframe Development and Testing. 2020-02-04. също дост. на: <https://www.compuware.com/topaz-modern-mainframe-development-testing/>.
- [19] NIERSTRASZ, O. и др. The story of moose. *ACM SIGSOFT Software Engineering Notes*. 2005, т. 30, № 5, с. 1.

- [20] MURTHY, N. Codiscope SecureAssist™: The Developer's Security Assistant. в: *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 2016.
- [21] *splint - secure programming lint*. 2020-02-04. също дост. на: <https://sourceforge.net/projects/splint/>.
- [22] *Coverity*. 2020-02-03. също дост. на: <https://www.synopsys.com/>.
- [23] *RIPS*. 2020-02-04. също дост. на: <https://www.ripstech.com/>.
- [24] *Code Dx software application vulnerability management suite*. 2020-02-04. също дост. на: <https://codedx.com/>.
- [25] TRIPP, O. et al. Finding your way in the testing jungle: a learning approach to web security testing. в: *Proceedings of the 2013 International Symposium on Software Testing and Analysis - ISSTA 2013*. ACM Press, 2013.
- [26] BEYER, D. et al. The software model checker Blast. *International Journal on Software Tools for Technology Transfer*. 2007, vol. 9, no. 5-6, с. 505–525.
- [27] *ECLAIR*. 2020-02-05. също дост. на: <https://www.bugseng.com/>.
- [28] BRIAND, P. e. a. Software Quality Objectives for Source Code. В: 2010.
- [29] SHU, G. et al. JavaPDG: A New Platform for Program Dependence Analysis. в: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 2013.
- [30] BALL, T. et al. From symptom to cause. *ACM SIGPLAN Notices*. 2003, vol. 38, no. 1, с. 97–105.
- [31] BALL, T. The Concept of Dynamic Analysis. в: *Software Engineering – ESEC/FSE '99*. Springer Berlin Heidelberg, 1999, с. 216–234.
- [32] *Introduction to Gcov*. 2020-01-29. също дост. на: <https://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html#Gcov-Intro>.
- [33] *VB Watch v2: Profiler, Protector and Debugger for VB6*. 2020-01-29. също дост. на: <https://www.aivosto.com/vbwatch.html>.
- [34] *JaCoCo Java Code Coverage Library*. 2020-01-29. също дост. на: <https://www.eclemma.org/jacoco>.
- [35] DALLMEIER, V. et al. Lightweight Defect Localization for Java. в: *ECOOP 2005 - Object-Oriented Programming*. Springer Berlin Heidelberg, 2005, с. 528–550.
- [36] SEREBRYANY, K. et al. AddressSanitizer: A Fast Address Sanity Checker. в: *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 2012.
- [37] LUECKE, G. R. et al. A survey of systems for detecting serial run-time errors. *Concurrency and Computation: Practice and Experience*. 2006, vol. 18, no. 15, с. 1885–1907.
- [38] NETHERCOTE, N. et al. Valgrind. *ACM SIGPLAN Notices*. 2007, vol. 42, no. 6, с. 89.
- [39] *Automated vulnerability scanning of web applications with Rational AppScan*. 2020-02-05. също дост. на: <https://www.ibm.com/developerworks/rational/library/automated-vulnerability-scanning-web-applications/index.html>.
- [40] FLANAGAN, C. et al. Adversarial memory for detecting destructive races. *ACM SIGPLAN Notices*. 2010, vol. 45, no. 6, с. 244.
- [41] SORRENTINO, F. et al. PENELOPE: Weaving threads to expose atomicity violations. в: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*. ACM Press, 2010.
- [42] ABREU, R. et al. On the Accuracy of Spectrum-based Fault Localization. в: *Testing: Academic and Industrial Conference Practice and Research Techniques – MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007.
- [43] FLANAGAN, C. et al. Velodrome. *ACM SIGPLAN Notices*. 2008, vol. 43, no. 6, с. 293.
- [44] FLANAGAN, C. et al. Atomizer. в: *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '04*. ACM Press, 2004.
- [45] ERNST, M. D. et al. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*. 2007, vol. 69, no. 1-3, с. 35–45.
- [46] CSALLNER, C. et al. DSD-Crasher. в: *Proceedings of the 2006 international symposium on Software testing and analysis - ISSTA'06*. ACM Press, 2006.
- [47] *Delta Project home*. 2020-01-29. също дост. на: <http://delta.tigris.org/>.
- [48] *Using Lithium*. 2020-01-29. също дост. на: <http://www.squarefree.com/lithium/using.html>.
- [49] CHEN, Y. et al. C-Reduce. 2020-01-14. също дост. на: <https://embed.cs.utah.edu/creduce/>.
- [50] LARUS, J. R. et al. Rewriting executable files to measure program behavior. *Software: Practice and Experience*. 1994, vol. 24, no. 2, с. 197–218.

- [51] EDWARDS, J. C. *Method, system, and program for logging statements to monitor execution of a program*. пат. на САЩ, 6539501. 2003-03-25.
- [52] *OpenClover*. също дост. на: <http://openclover.org/>.
- [53] DEBROY, V. et al. A consensus-based strategy to improve the quality of fault localization. *Software: Practice and Experience*. 2011, vol. 43, no. 8, c. 989–1011.
- [54] BAUDRY, B. et al. Improving test suites for efficient fault localization. в: *Proceeding of the 28th international conference on Software engineering - ICSE '06*. ACM Press, 2006.
- [55] ARTZI, S. et al. Practical fault localization for dynamic web applications. в: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*. ACM Press, 2010.
- [56] CHESLEY, O. C. et al. Crisp—A Fault Localization Tool for Java Programs. в: *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007.
- [57] *GNU gprof*. 2020-01-29. също дост. на: <https://sourceware.org/binutils/docs/gprof/>.
- [58] MACHADO, P. et al. MZoltar: automatic debugging of Android applications. в: *Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile - DeMobile 2013*. ACM Press, 2013.
- [59] JANSSEN, T. et al. Zoltar. в: *Proceedings of the 2009 ESEC/FSE workshop on Software integration and evolution @ runtime - SINTER '09*. ACM Press, 2009.
- [60] CHEN, K. et al. Aspect-Based Instrumentation for Locating Memory Leaks in Java Programs. в: *31st Annual International Computer Software and Applications Conference - Vol. 2 - (COMPSAC 2007)*. IEEE, 2007.
- [61] *NDepend*. 2020-02-03. също дост. на: <https://www.ndepend.com/>.
- [62] GOUGH, B. J. et al. *An Introduction to GCC: For the GNU Compilers GCC and G++*. Network Theory Ltd., 2004.
- [63] LEHMANN, D. et al. Feedback-directed differential testing of interactive debuggers. в: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. ACM Press, 2018.
- [64] ZELLER, A. *Delta Project Home*. 2019-12-26. също дост. на: <http://delta.tigris.org/>.
- [65] ZELLER, A. Isolating cause-effect chains from computer programs. в: *Proceedings of the tenth ACM SIGSOFT symposium on Foundations of software engineering - SIGSOFT '02/FSE 10*. ACM Press, 2002.
- [66] FLANAGAN, C. et al. Extended static checking for Java. *ACM SIGPLAN Notices*. 2002, vol. 37, no. 5, c. 234.
- [67] GDB. *GDB: The GNU Project Debugger*. 2019-12-29. също дост. на: <https://www.gnu.org/software/gdb/>.
- [68] STALLMAN, R. et al. Debugging with GDB. *Free Software Foundation*. 2002, vol. 51, c. 02110–1301.
- [69] MICROSOFT INC. *Debugging in Visual Studio*. също дост. на: <https://docs.microsoft.com/en-us/visualstudio/debugger/debugging-in-visual-studio?view=vs-2015&redirectedfrom=MSDN>.
- [70] PASTORE, F. et al. RADAR: A tool for debugging regression problems in C/C++ Software. в: IEEE, 2013, c. 1335–1338.
- [71] ЛЕСЕВ, Х. *Модулни разширяеми системи за глобално осветяване*. Пловдив, България: Пловдивски Университет, 2015. Дисертация. Пловдивски Университет "Паисий Хилендарски". 160 л.
- [72] *A Periodic Table of Visualization Methods*. 2020-01-31. също дост. на: https://www.visual-literacy.org/periodic_table/periodic_table.html.
- [73] *Collection of Data Visualization Tools*. 2020-02-02. също дост. на: <https://datavizproject.com/>.
- [74] CHRISTENSEN, C. On the Implementation of AMBIT, a Language for Symbol Manipulation. *Communications of the ACM*. 1966, vol. 9, no. 8, c. 570–573.
- [75] ELLIS, T. O. et al. *The GRAIL Project: An Experiment in Man-machine Communication*. Rand, 1969. Memorandum (Rand Corporation).
- [76] SMITH, D. C. *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*. Birkhäuser, 1977. Interdisciplinary systems research.
- [77] PIETRZYKOWSKI, T. et al. The Programming Language PROGRAPH: Yet Another Application of Graphics. в: *Graphics Interface*, 1983, c. 143–145.

- [78] VAN REETH, F. et al. IGIP: a framework towards open-ended visual programming. в: *Visual Languages, 1988, IEEE Workshop on*. 1988, с. 239–247.
- [79] KURLANDER, D. et al. Inferring Constraints from Multiple Snapshots. *ACM Trans. Graph.* 1993, vol. 12, no. 4, с. 277–304.
- [80] MITKIN, S. *DRAGON – Erlang: Visual functional programming* [онлайн]. 2012 [дата на посещ. 2019-02-19]. достъпен на: <http://drakon-editor.sourceforge.net/drakon-erlang/intro.html>.
- [81] TAENTZER, G. et al. Amalgamated Graph Transformations and Their Use for Specifying AGG - an Algebraic Graph Grammar System. в: *Proceedings of the International Workshop on Graph Transformations in Computer Science*. London, UK, UK: Springer-Verlag, 1994, с. 380–394.
- [82] KLEIN, P. et al. Constructing SDEs with the IPSEN Meta Environment. в: *Proceedings of the 8th International Conference on Software Engineering Environments (SEE '97)*. Washington, DC, USA: IEEE Computer Society, 1997, с. 2–10. SEE '97.
- [83] SOFTWARE, M. *OpenWire* [онлайн]. 2001 [дата на посещ. 2019-05-08]. достъпен на: <http://www.mitov.com/products/openwire>.
- [84] *KTechLab* [онлайн]. 2019 [дата на посещ. 2019-12-01]. достъпен на: <http://sourceforge.net/projects/ktechlab/>.
- [85] BRESSON, J. et al. OpenMusic: Visual Programming Environment for Music Composition, Analysis and Research. в: *Proceedings of the 19th ACM International Conference on Multimedia*. Scottsdale, Arizona, USA: ACM, 2011, с. 743–746. MM '11.
- [86] BAYCASTLE SOFTWARE LTD. *DataSlave* [онлайн]. 2019 [дата на посещ. 2019-12-01]. достъпен на: <http://www.baycastle.co.uk/V2/DataSlave/DataSlave.htm>.
- [87] SOI, S. et al. Conceptual Development of Custom, Domain-Specific Mashup Platforms. *ACM Transactions on the Web*. 2014, vol. 8, no. 3, с. 1–35.
- [88] YQL. *Yahoo!Pipes* [онлайн] [дата на посещ. 2019-06-12]. достъпен на: <http://pipes.yahoo.com/pipes/>.
- [89] KENT, B. R. Visualizing Astronomical Data with Blender. *Publications of the Astronomical Society of the Pacific*. 2013, vol. 125, no. 928, с. 731–748.
- [90] RADEMAKERS, F. et al. ROOT: An Object-oriented Data Analysis Framework. *Linux J.* 1998, vol. 1998, no. 51.
- [91] HEESCH, D. Doxygen: Source code documentation generator tool. 2008.
- [92] VASSILEV, V. et al. SolidReflector: A Multistage, Interactive Decompilation Framework. в: *Proceedings of the International Conference "From DeLC to Velspace"*. 2014, с. 49–58.
- [93] CYTRON, R. et al. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*. 1991, vol. 13, no. 4, с. 451–490.
- [94] HALL, M. W. et al. Efficient call graph analysis. *ACM Letters on Programming Languages and Systems*. 1992, vol. 1, no. 3, с. 227–242.
- [95] *Image Comparison using ImageMagick*. 2020-02-10. също дост. на: <https://imagemagick.org/script/compare.php>.
- [96] IRETON, M. et al. *System and method for performing software patches in embedded systems*. 5901225. 1999-05-04.
- [97] ERLINGSSON, Ú. et al. Low-Level Software Security by Example. в: *Handbook of Information and Communication Security*. Springer Berlin Heidelberg, 2010, с. 633–658.
- [98] JAY, G. et al. Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship. *Journal of Software Engineering and Applications*. 2009, vol. 02, no. 03, с. 137–143.
- [99] VASSILEV, V. et al. SolidOpt – A Multi-Model Software Optimization Framework. *International Journal of Computer Science Issues*. 2015, vol. 12, с. 32–41.
- [100] PENEV, A. et al. Applying Model-View-Controller in Geometric Modelling Methodology. в: *Varna, Bulgaria, 2008*, с. 59–66.
- [101] SCOTT CHACON, B. S. *Pro Git*. 2nd ed. Berkely, CA, USA: APRESS L.P., 2014.
- [102] DABBISH, L. et al. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. в: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. Seattle, Washington, USA: ACM, 2012, с. 1277–1286. CSCW '12.
- [103] HEESCH, D. van. *Doxygen* [онлайн]. 2019 [дата на посещ. 2019-12-01]. достъпен на: <http://www.stack.nl/~dimitri/doxygen/>.

- [104] LIEBERMAN, H. Programming by Example (Introduction). *Communications of the ACM*. 2000, vol. 43, no. 3, с. 72–74.
- [105] ST. AMANT, R. et al. Programming by Example: Visual Generalization in Programming by Example. *Communications of the ACM*. 2000, vol. 43, no. 3, с. 107–114.
- [106] БАСИЛЕВ, М. *SolidReflector – Визуален, интерактивен, статичен и динамичен анализ на .NET приложения*. Пловдив, 2013. Дипломна работа за магистър. Пловдивски Университет "Паисий Хилендарски". 87 л.
- [107] BRUN, R. et al. ROOT — An object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*. 1997, vol. 389, no. 1-2, с. 81–86.
- [108] ZAITSEV, A. Regression in the constant evaluation of 'switch' statements. 2019-10-15. също дост. на: https://bugs.llvm.org/show_bug.cgi?id=43674.
- [109] VASILEV, V. G. *Installed ROOT makes stats to its Jenkins build folder*. 2020-01-08. също дост. на: <https://sft.its.cern.ch/jira/browse/ROOT-10497>.
- [110] VASILEV, V. G. *ROOT master does redundant stats on virtual files*. 2020-01-08. също дост. на: <https://sft.its.cern.ch/jira/browse/ROOT-10496>.
- [111] БАСИЛЕВ, М. *SolidReflector – мултимоделен интерактивен анализ и декомпиляция на .NET приложения*. Пловдив, 2012. Дипломна работа за бакалавър. Пловдивски Университет "Паисий Хилендарски". 62 л.
- [112] БАСИЛЕВ, В. *SolidOpt – Инструментарии за автоматизирани оптимизации на софтуерни приложения*. 2009. Дипломна работа за бакалавър. Пловдивски Университет "Паисий Хилендарски". 99 л.
- [113] БАСИЛЕВ, В. *SolidOpt – Инструментарии за автоматизирани оптимизации на софтуерни приложения. Оптимизации, зависещи от приложната област*. 2010. Дипломна работа за магистър. Пловдивски Университет "Паисий Хилендарски". 127 л.
- [114] ПЕТРОВА, П. *DataMorphose – Визуален анализ и трансформации на данни*. Пловдив, 2013. Дипломна работа за магистър. Пловдивски Университет "Паисий Хилендарски". 60 л.