www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 7 July 2015, Page No. 13273-13278

Assessing Version Consistency by Identifying systematic Code Changes during Runtime

P.Swapna Shankar, P.Niranjan, P.Shireesha

M.Tech(SE), Department of CSE,
KITS Warangal,T.S, INDIA
swaps2644@gmail.com

Prof&head Department of CSE,
KITS Warangal, T.S,INDIA
npolala@yahoo.co.in

Asst.professor, Department of CSE,
KITS Warangal,T.S,INDIA
Siriniru55@gmail.com

Abstract: In compelling software revisions there survive various degrees of potential behavior change. The easiest strategy of adapting an request is to alter the performance of a system body, i.e., upgrading the method body to a classic version without modifying the entire request, for instance if a bug inside a single technique gets unchanging, or a quicker algorithm through the same features gets arranged. A next move in the direction of absolute revisions is the ability to alter the method signature, where not just the internals of a method, but also the endeavor signature alone, i.e., the quantity as well as types of variables, the review type or the technique name get altered. The final step regarding random modifications and completely energetically updateable techniques is the sustain for modifying global fields also fields inside of schemes, for instance in the case of class-based techniques the fields of elements as specific by their individual sessions. In this device we provide a runtime evaluation version persistence to evaluate the impact of the dynamic software revisions.

Keywords: *Software evaluation, dynamic software loaded, JRE, Component bug prediction*

1. Introduction

Dynamic component load is frequently used in software technique progress to build standard as well as pliant software method. Java run-time setting (JRE) generally provides appropriate strategy calls to connect dynamic attributes. The internal JRE solves also much the certain part; each beginning technique phone is invoked. Part quality depends upon the process the half is certain what is much from laterally the scheduled part's complete path or its file-name. Presented an entire path, the JAVA Runtime environment just utilizes it for quality. That sequence of websites to search is handling at run-time by the unique index search order at that chance of system quality invocation? The ability of the standard strategy of part loading can incorporate a price associate degree built-in security apprehension is revealed by it. For runtime security as well as fortification, correlate degree demand should merely fill its planned elements. However, as a component is resolute by the JRE entirely during its name,

development errors may result in the initiation of correlate degree accidental part with effectively the equivalent name.

2. Recent work [12] has recognized that dangerous loadings are common and should origin remote code execution attacks. Associate quantity approach was recommended to seek out dangerous half loadings. It then executes an analysis to find 2 sorts of dangerous loadings: decision and backbone failure hijacking. Once the target half is not exposed, though resolution hijacking happens once unusual sites are looked earlier than the listing wherever the half lives a top excellence failure happens.
3. We demonstrate this perplexity victimization deferred loading, associate degree optimization to holdup the loading of seldom used components till their appallingly initial use. Since it's fatiguing to activate all delayed loadings at runtime delayed loading is durable for dynamic detection.
4. Within this document, we are inclined to current the appallingly initial static analysis to seek out risky loadings from program binaries. 2 things of essential proposal are required: one) all components which can be crowded at each loading resolution web site, and 2) the protection of every possible loading From these findings, we tend to fashion a - amount analysis: checking and extraction. The elimination stage is demand driven, operating backwards from each loading assessment web site} to calculate the collect of potential loadings; the stage establishes the security of the loading by analyzing the appropriate directory search order within the begin site.

Context-Sensitive Emulation We start context sensitive emulation, a unique mix of emulation as well as segmenting, to comprehend the vulnerable computation of constraint values

throughout the elimination amount. For a specific decision website, we come with a tendency to remove its context prone efficient blocks in significance its pointers, one for every effecting context. We have an inclination to later copy the blocks to evaluate the limitation values.

Incremental and typical Segmenting: One specific challenge is that the compliment of evaluate diffident locks scalable. Ordinary segmenting techniques [1, 5, 9, 16, 20, 21] are centered on strategy a program's whole system addiction graph (SDG) a priori also are subsequently limited in measurability. Through we have an appreciation to merely must be obligated to consider loading decision websites additionally because the performance pathways to determine the constraint values to the describe are generally relatively short, exclusively a bit a part of the definite SDG is appropriate for our evaluation. This conjures up the usage of AN in tiny stages as well as normal sectioning guideline (cf. Section 3)--incremental as a result of we have a possibility to establish the blocks idly when required; typical due to once we notice a complete decision foo (x,y), we come with an affinity to usage AN conditional determine concerning what dependence foo's variables and return worth have in exploratory the caller. In the end, we have a inclination to join the executes function blocks inside the traditional manner by concerning real as well as formal variables.

Emulation of Context-sensitive Slice: when we've supposed the piece s regarding an explicit loading selection web site, we come with an affinity to must determine values for the essential pointers. One unrefined solution might be to perform basic delegate evaluation on the piece to determine the values. The main disadvantage for this strategy is that the concern in reckoning symbolically about tactic calls as the appropriate variables often views complicated, lower level setup calls.

To overcome this concern, we apply emulation. In particular, we generate, from the backward piece s, a collection of context sensitive feasible sub blocks, which we eventually emulate to determine the limitation values (cf. Area three). s 's sub-blocks $s_1,...,s_n$. Directives in each sub-slice s_i are then follow topologically, about their data and control-flow dependencies.

For evaluation, we usually apply our strategy in a fairly model system for Windows enclose. We often evaluate our tool's efficiency beside the above dynamic method [12] regarding exactitude, quantifiability, as well as coverage. Listings on 9 frequent programs reveal that our system is scalable as well as appropriate (cf half 4). Such as, it obtained but 2 occasions to seem at each of the 9 choose a check issues, collectively with significant programs as participant Reader, Safari, and QuickTime. The outcomes also reveal that our estimated context sensitive emulation obtains orders of magnitude reduction inside the size of the code essential to be analyzed and critically provide to the quantifiability of our method. In provisions of coverage, our tool identifies more prospective perilous loadings also comfortably matches the dynamic strategy.

Main Contributions:

We've urbanized the main static twin assessment to choose unsafe component loadings. Attributable to its quantifiability as well as advanced code coverage, our strategy efficiently enhances the obtainable dynamic strategy.

We have estimated context-sensitive emulation, Correlate in nursing affordable strategy that mixes segmenting as well as emulation for the proper also ascendible evaluation of runtime specifications of program factors.

The rest of this premise is prepared as observe. Section a pair of demonstrates our strategy with a running model. Section three provides complete information of our static appreciation algorithmic strategy. We often illustrate our implementation as well as assessment in Section four. Lastly, Section five studies further associated work, also Section six ends with a spoken language of upcoming work.

1. OVERVIEW

This section demonstrates our strategy. Our strategy performs on binaries, but for presentational purpose, we reveal the sample in C-like pseudo code.

Extraction Phase: We foremost evaluate selection sites for component loading. Inside the sample, line 23 refers to a decision website from the Load Library controller call direction. The program call's only limitation target_api establishes that component should be encumbered. We obtain a possibility to utilize context-sensitive emulation to evaluate its possible values.

Incremental and standard Segmenting: System segmenting generally looks at point stream dependencies also information to eliminate a piece. In our option, since the initial objective is to work out doable values of target_api, we obtain a possibility to produce the piece as well as think about data dependencies. To determine the possible values of goal_api, we'd choose require completely the code that results the primary factor of the function. To the final, we get a tendency to preserve the backward segmenting in relevance your latest segmenting standard, which is developed secured caller-callee connection also conjointly the callee's function image. In our illustration, there exist to determination sites. Therefore, we have a possibility to maintain with situations of Intra techie backward segmenting in relevance original segmenting requirements. We have a possibility to generate context sensitive lay proceeding blocks by instantiating twice as the piece for delay as well as connecting each event with its several caller's slice. We obtain a possibility to conjointly manage the maps around all of the newer segmenting standards and so the callee's comparative variables for the bluffly emulation quantity. We provide a possibility to determine the segmenting computation, through neither of provides any signaling.

Emulation of flow connected blocks: the blocks square estimate adopted by U.S., to determine standards for target nine api. We provide a possibility to arrange the rules inside the blocks earlier to they may be traced. We obtain a tendency to do thereby regarding the information and handle flow dependencies joining the directions. Particularly, we obtain a possibility to 1st program the relevant blocks in topological determine with relevance the data flow dependencies around them. We have a possibility to subsequently determine the obtaining of the procedures in each designed essential block regarding their setup flow dependencies limit within the initial code.

Checking part. When the JRE masses the elements, it iterates during a series of websites, established at run-time, to choose the data. Throughout this state of issues, these consignments square determine dangerous, if the JRE monitors various sites to disentangle these factors. This may be as some loadings could be hijacked by golf shot correlate absolute data. We provide a possibility to examine whether or not the provided files exist inside the primary listing checked. Due to Ms

Windows's searches major inside the directory anywhere the strategy is put in [7], the loadings for such 2 features square determine unsafe if they can't overcome inside the system directory.

1. STATIC DETECTION ALGORITHM

In this section, we provide background in series on unsafe element loadings as well as information of our evaluation.

3.1 Background

Dynamic component loading is usually established by JRE by meticulous program calls that obtain as enter a full path or document name for the estimated component. The instance of determinant the focus on component by JRE as observe:

The target element is scrupulous by its complete path or its type.

When complete passage is used, the JRE overtly establishes the target exploitation the whole path.

Alternatively, if document name is used and identified by the JRE, the whole path of the scrupulous type is predefined.

If the specific file name is anonymous to the JRE, it iterates during the predefined class methods to find the basic file with the conscientious file name.

To sanctify the component determination strategy, it's essential to model the category route state, because of even the equivalent part- loading code might results in distinct resolutions here distinct category route states.

Component Resolution: A component resolution function γ gets a component necessity $f \in \sum^{*}$, *a* directory explore order $d = (d_{1,\ldots d_n}) \in \sum^{*} \times \ldots \sum^{*}$ *and a class path* state σ also leads a determined full path $\pi \in \sum^{*}$, where $\sum$ shows the alphabet utilized to categorize files as well as indexes.

If f is a full path, $\gamma(f,d,\sigma) = \{$ f if $f \in \sigma$; $\in$ or else where $\in$ is the empty string

If *f* is a file name,

$\gamma(f,d,\sigma) = \{\ \pi$ if f is acknowledged to the JRE as ; $\in$ or else, where *"+ "* shows string concatenation

We further observe part loading that we obtain a possibility to require to consider about the actually loaded procedure. The interest is that the JRE does not load a similar component frequently. In our formalization, we provide a possibility to let the set of burdened components L be the set of comprehensive techniques of all the actually loaded components.

Component Loading: selected the loaded elements L, a component loading function A requires a component condition $f \in \sum^{*}, a$ directory search order $d = (d_{1,\ldots d_n}) \in \sum^{*} \times \ldots \sum^{*}$ a file system state a, also the position of loaded components *L*, and continues an argument success or failure:

$\lambda(f,d,\sigma,L) = \{$ Success if $\gamma(f,d,\sigma) \notin \{\in\} \cup L$ Failure other wise

The sensible component loading system is consistently utilized on significant JREs. However a full path completely defines the target component, for a file name, the complete path of the loading component frequently relies on the below file system specify. This system can proceed to two kinds of vulnerable loadings: *declaration failure* and *resolution hijacking*.

Resolution Failure: A declaration failure occurs if $\gamma(f,d,\sigma) = \in$. In this container, with a complete path specification f, a capricious file using same full path f can hijack the component loading. If f is file name, one be sensible to hijack this loading by beginning a file with the scrupulous name f in any listing d_i exacting by the explore order $d = (d_{1,\ldots d_n})$.

Resolution Hijacking: A quality hijacking end up if the subsequent situations hold: 1) f is the file name of the target component also indefinite to the JRE; 2) $\gamma(f,d,\sigma) = d_k + \backslash + f < k > 1$ and 3) $\lambda(f,d,\sigma,L) =$ success. In this container, one may hijack the loading by setting up a file with the conscientious name f in any type of directory d_i where i < k.

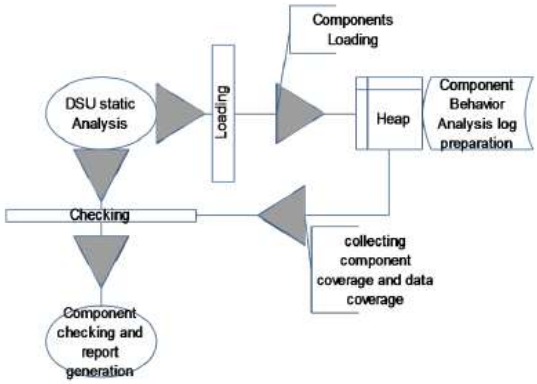


Figure 1: Architecture of the proposed framework

To avoid vulnerable loadings, it is worthwhile for developers to determine the target component in a secure manner. We describe safe target component criteria as follows.

2. EMPIRICAL EVALUATION

In this segment, we assess our static strategy in regards to consistency, scalability, also strategy coverage. We reveal that our strategy scales to significant real-world conformity and is appropriate. It offers good exposure, considerably better than the obtainable dynamic strategy [12].

4.1 Implementation

The semi automated dynamic software update evaluation estimated is considered under JRE. In this imagine the model has been used to test the alert used on open source program entitle GDOWNLOADER.

4.2 Evaluation Setup and Results

We project at finding vulnerable component loadings in programs. As the detection of vulnerable loadings from the APIs is operating by the JRE, we just determine the application process in the extraction phase.

2.2.1 Precision and Scalability

Table 1 reveals our research outcomes on eight loved Windows programs. Because of they are essential programs in America these programs were most prominent by us as our evaluation subjects. The results describe that our strategy could efficiently discover, from system binaries, unsafe component loadings definitely loaded at runtime. One desirable finding to notice is that the outcomes of the extraction part are identical. This is very possibly by every apps is a part of the Mozilla project and utilize the precise similar set of program elements.

As we obtain a possibility to gift eventually our evaluation time is mastered by now. Such are giant code, also to boot we provide a propensity to just need to destruct the code once for most of the consequent analysis.

Based on our evaluation of context sensitive emulation, the quantity of blocks is normally significant than that of the call up sites. This signifies that variables for consignment catalogue calls will have numerous values, verifying the necessity for decision flow associated blocks. The day to day many instructions for the blocks is very small, that by trial as well as error approved our evaluation style choices.

We usually converse the evaluation of our tool's measurability. To the conclusion, we provide a possibility to live its evaluation time and then the efficiency of its back-ward segmenting phase. Table1 reveals the great outcomes, the outcomes reveal that our evaluation is wise also may evaluate inside moments. We get a possibility to evaluate our semi automated DSU evaluation strategy with absolutely mechanized as well as manual methods, to support perceives its effectiveness. We obtain a possibility to thereby determine what percentage pointers as well as functions you can find in all application through these numbers represent the value of this a priori development. Considering the table1, table2 also table3 shows, we get a possibility to perform orders of size reducing in regards to every variety of functions at the aspect of the quantity of directions assessed.

Table 1: Component wise report generated by the proposed architecture

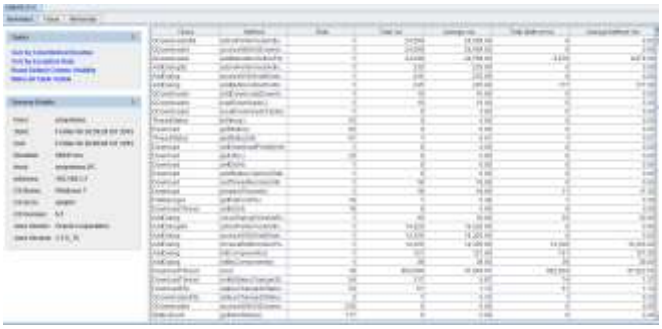


Table 2: Sample Call tree analysis report generated

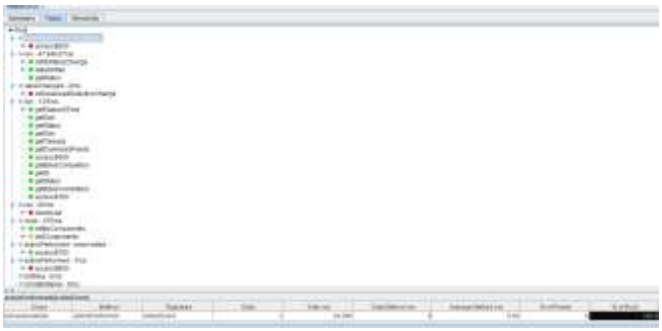
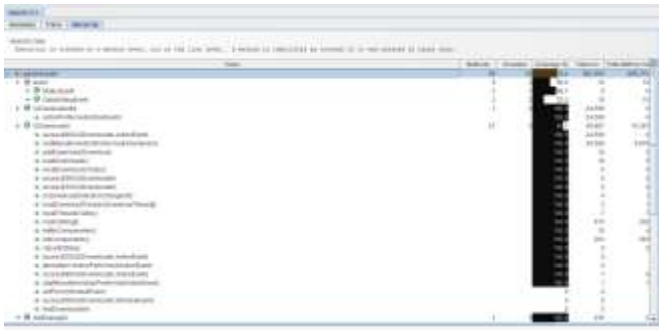


Table 3: Sample Coverage analysis report generated



4.2.2 Code Coverage

To value our tool's code protection, we obtain a possibility to evaluate unsafe loadings comprehend by the static as well as powerful examines. In particular, we obtain a possibility to identified unsafe component loadings with this dynamic strategy [12] also evaluate its results with our semi automated detection. In this evaluation, we obtain a possibility to emphasize on application-level runtime unsafe loadings as load time based elements square determine loaded by JRE-level code. We obtain a possibility to observe that our semi automated model will determine not only in the significant of the dynamically-detected vulnerable loadings besides conjointly various different prospective ones conjointly. We obtain a possibility to next result in a most in-depth assessment of the outcomes.

Static-only Cases: Our static evaluation notice many extra prospective unsafe loadings. It's relevant to understand whether they reveal actual problems or not. We obtain a possibility to literally examine these further identified unsafe loadings to evaluate the precision of our analysis. Particularly, we obtain a possibility to evaluated even if they square quantify approachable from the access points of the applications, I.e., whether there endure techniques from the connection points to the assess sites of the vulnerable loadings inside the strategies' inter-method decision flow graphs (Inter-method summarize flow graphs).

Remember that these loadings significant as "Unknown" should remain obtainable by it's tough to determine circuitous advances in code, thus definite organize flow edges is even inadequate from the Inter-method decision flow graphs. Every statically obtainable unsafe loadings result in component load hijacking if the related decision sites square determine rise as well as conjointly the target components are not loaded nevertheless.

External Parameters: A target pattern can be separate by a constraint of an export function, which is not invoked. One can offset this problem by evaluating the data flow reliance among the dependent components.

As the export functions are usually not use to the components, anyhow, like an analysis does not ensure to obtain all the objective specifications.

Unknown Semantics of System Calls: overall semantics of category calls is frequently not reported, also at times usually their names aren't revealed. We cannot examine nor copy them, when we encounter such company calls. When information of that way calls becomes obtainable, we can definitely add evaluation assistance for them.

1. RELATED WORK

We analyze additional associated work excluding for the one on identification of unsafe loadings [12], that we have obtained already revealed. Our strategy carries out static evaluation of

binaries. Concerning this option, evaluation Set Analysis (VSA) [2, 18] is even the principally strongly connected to ours. It integrates numeric as well as indicator examines to determine a complete approximation of statistical values of system variables. Value to VSA, our strategy centers on the estimation of string variables. Furthermore, demand-driven also utilizes context-sensitive emulation to level to real-world significant programs.

Emblematic evaluation [11] might also be applied to determine values of the system elements, when we obtain a possibility to describe antecedently, instead of emulation. Anyhow, symbolic strategies frequently sustain start poor measurability, also further considerably, it is not appropriate to symbolically cause about system calls, that square determine usually quite challenging. Our newer usage of context prone emulation offers a worthwhile alternative for dispensation the values of system variables.

Beginning with Weiser's seminal perform [25], system segmenting might substantially examined [23, 26]. Our perform is associated with the significant system of focus on static segmenting, particularly the SDG-established strategies. Prevalent SDG-based static segmenting strategies [1, 5, 9, 16, 20, 21] establish the full SDGs upfront. In variation, we obtain a possibility to establish management - and data - flow reliance in pattern in an exceedingly manner, beginning with the necessary segmenting standards. Our segmenting strategy is in addition, accepted as a consequence of we obtain a possibility to model all decision website using its callee's inferred determine that abstracts away the inner dependency of the callee. Particularly, we obtain a possibility to deal with a phone as a non branching guidance and estimate its dependencies with the callee's conceptual data. This optimization stands USA to abstract absent clarify data flow dependencies of objective exploitation its comparative decision instruction. We obtain a possibility to develop AN effective trade-off among correct as well as measurability. As revealed by our evaluation outcomes, execute epitome information may be with performance computed also delivers appropriate outcomes for our location.

Our segmenting guideline is demand driven, also is hence further associated to demand-driven dataflow examines [10, 17], that are estimated to enhance analysis efficiency once complete dataflow specifics aren't essential. These techniques square determine similar to ours through the additionally control caller association to eliminate unfeasible dataflow techniques. The main variation is that we obtain a possibility to utilize an easy epitome evaluation to create pithy work summaries as an alternative of exclusively crossing the functions' Intra proceeding dependency graphs, I.e., their PDGs. The other variation is that the revealed concept that we obtain a possibility to produce context sensitive conceivable program prevent for emulation to hinder the situation in brooding about system calls.

2. CONCLUSION AND FUTURE WORK

We Have bestowed a semi automated DSU evaluation strategy to find vulnerable loadings. The center of our evaluation is strategies to only as well as ascendible to acquire those elements square determine loaded at a specific consignment selection website. We obtain a possibility to expelling a java stack log extraction as well as evaluation process, which combines standard as well as progressive slice development with the emulation of decision flow connected blocks. Our evaluation on 9 loved Windows distribution reveals the efficiency of our strategy. Due to its sensible measurability, precision, also security, our strategy is an effective balance to dynamic detection [12]. For prospective work, we'd have to suppose 2 attention-grabbing instructions. Since unsafe loading can be a basic issue furthermore as suitable to additional runtime locations, subsequently we obtain a possibility to shall prolong our strategy and evaluate unsafe half loadings in more run time situations including CLR. Next, we obtain a possibility to determine to explore although our strategy is enhanced to chop back emulation problems.

References

[1] Martin Abadi and Cedric Fournet. Access control based on execution history. In NDSS, 2003.

[2]. Andrew Baumann, Gernot Heiser, Jonathan Appavoo, et al. Providing dynamic update in an operating system. In USENIX, 2005

[3].Andrew Baumann, Jonathan Appavoo, Robert W. Wisniewski, et al. Rebootsare for hardware: Challenges and solutions to updating an operating system on the fly. In USENIX, 2007.

[4] Chandrasekhar Boyapati, Barbara Liskov, Liuba Shrira, Chuang-Hue Moh,and Steven Richman. Lazy modular upgrades in persistent object stores. In OOPSLA, 2003.

[5] Haibo Chen, Rong Chen, Fengzhe Zhang, Binyu Zang, and Pen-ChungYew. Live updating operating systems using virtualization. In VEE,2006.

[6] Haibo Chen, Jie Yu, Rong Chen, Binyu Zang, and Pen-Chung Yew. POLUS:A POwerful Live Updating System. In ICSE, pages 271–281,2007.

[7] Dawson Engler and Ken Ashcraft. RacerX: effective, static detection of race conditions and deadlocks. In SOSP, 2003

[8]. Cormac Flanagan and Stephen N. Freund.Type-based race detection for Java. In PLDI, 2000.

[9] Jeffrey S. Foster, Robert Johnson, John Kodumal, and Alex Aiken.Flow-Insensitive Type Qualifiers. TOPLAS, 28(6):1035–1087, November 2006.

[10] Stephen Gilmore, Dilsun Kirli, and Chris Walton.Dynamic ML without dynamic types. Technical Report ECS-LFCS-97-378, LFCS, University of Edinburgh, 1997.

[11] Tim Harris and Keir Fraser. Language support for lightweight transactions. In OOPSLA, 2003.

[12] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In ISCA, 1993.

[13] Michael Hicks, Jeffrey S. Foster, and Polyvios Pratikakis. Lock Inference for Atomic Sections. In TRANSACT, 2006.

[14] Atsushi Igarashi and Naoki Kobayashi. Resource Usage Analysis. In POPL,Portland, Oregon, 2002.

[15] John Kodumal and Alexander Aiken. Banshee: A scalable constraint-based analysis toolkit. In SAS, 2005.

[16] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. CACM, 21(7):558–565, 1978.

[17] Insup Lee. DYMOS: A Dynamic Modification System. PhD thesis, Dept. of Computer Science, University of Wisconsin, Madison, April 1983.

[18] John M. Lucassen. Types and Effects: Towards the Integration of Functional and Imperative Programming. PhD thesis, MIT Laboratory for Computer Science, August 1987.MIT/LCS/TR-408.

[19] Kristis Makris and Kyung Dong Ryu. Dynamic and adaptive updates of non-quiescent sub systems in commodity operating system kernels. In Proc. Euro Sys, March 2007.
[20] Jeremy Manson, William Pugh, and Sarita V. Adve.The Java Memory Model. In POPL, 2005.
[21] John C. Mitchell. Type inference with simple subtypes. JFP, 1(3):245–285,July 1991.
[22] Mayur Naik and Alex Aiken. Conditional must not aliasing for static race detection. In POPL, 2007.
[23] Mayur Naik, Alex Aiken, and John Whaley.Effective static race detection for java. In PLDI, 2006.
[24] Iulian Neamtiu, Jeffrey S. Foster, and Michael Hicks. Understanding Source Code Evolution Using Abstract Syntax Tree Matching. In MSR'05,2005. URL http://www.cs.umd.edu/~mwh/papers/evolution.pdf.
[25] Iulian Neamtiu, Michael Hicks, Gareth Stoyle, and Manuel Oriol. Practical dynamic software updating for C. In PLDI, 2006.
[26] Iulian Neamtiu, Michael Hicks, Jeffrey S. Foster, and Polyvios Pratikakis. Contextual Effects for Version-Consistent Dynamic Software Updating and Safe Concurrent Programming. Technical Report CS-TR-4920,

Author Profile

P.Swapna Shankar is currently pursuing her M.Tech Computer Science & Engineering Department in Kakatiya Institute of Technology and Science, Warangal. She received her B.Tech in Computer Science and Engineering from Sree chaitanya institute of technology and science, Thimmapur, karimnagar. Her area of interests includes Data mining and operating system.

Dr.Niranjan Polala is working as Professor and HOD of CSE in KITS, Warangal. He received Ph.D in CSE from Kakatiya University, Warangal in the year 2013. He received M.Tech (Computer Science and Engineering) from NIT, Warangal in the year 2001 and B.E Computer Science from Nagpur University in 1992. He authored three text books in the field of computer science. He published 30 research papers in various International Journals and Conferences. He is a member of the ISTE and CSI. His area of interests includes Software Engineering.

Dr.Shireesha Pakala is working as Assistant Professor in the Department of CSE, KITS, Warangal. She received Ph.D in Computer Science from Kakatiya University, Warangal in the year 2012. She received M.Sc. Computer Science from Kakatiya University in 2001. She published 8 research papers in various International Journals and International Conference. She is the member of the ISTE and IETE. Her area of interests includes Data mining and Software Engineering.