Challenges in Deploying Static Analysis Tools

Piyush Jain, Infosys Technologies Ltd

DTV Ramakrishna Rao, Infosys Technologies Ltd

Sathyanand Balan, Infosys Technologies Ltd

Abstract. For higher quality software and competitive products, many projects are feverishly deploying static analysis tools. Unfortunately, it turns out that many of the deployments are failures. Some have discontinued static analysis tools altogether. Some continue to use them, but find that the results are not as effective as they hoped.

There are many challenges facing static analysis tool deployments. Although static analysis tools have some weaknesses, the main challenge stems from people. Whether the tool deployment succeeds or fails depends on the people behind it. What are the challenges facing static analysis tool deployments and how can those challenges be overcome? This paper tries to answer that question based on our own deployment of the tools, consultancies with other organizations, and others' experiences.

Introduction

Various studies (e.g., [1]) suggest that around 40% of all software defects could have been detected using Automated Static Analysis (ASA)[1] tools. ASAs are also supposed to help in reducing field failures and time to market. Accordingly, many defense and non-defense projects are increasingly deploying [2] [3] ASAs such as Polyspace [4] and Prevent [5].

Are the deployments successful? Unfortunately, it turns out that many of the deployments are failures. Some projects discontinued ASA altogether. Some continue to use them, but find that the results are not as effective as they hoped. We consider both situations as failures of ASA deployment. These failures stem from the challenges facing ASA deployments. The first situation is more of a "hard failure" where because of the challenges facing ASA deployments, some were discontinued ASA altogether. In the case of the second situation, the failure is a "soft failure" and they continue to use ASA, but because of unmet challenges, the results from the deployments are not as effective as anticipated.

It is important to study the reasons for these failures and the challenges facing ASA deployments. It will help to learn from the mistakes of others in deploying ASA. Second, many are deploying ASA for competitive advantage. Hence, it is important to avoid failure of the deployment.

ASAs have some weaknesses [6] [7], but the leading cause of the failures is ill adoption of the tool by the people in the project. As we will see, just as people are the cause of failure, they are also the solution to make ASA deployments succeed.

This paper is organized in terms of various stages of introducing ASA into a typical project. In each stage, after pointing out how ASA is integrated with that stage, we discuss the challenges faced and how to overcome them. The first four sections –software defense application (Section 1), motivating the stakeholders (Section 2), training (Section 3), and integration into the process (Section 4)–are a prelude to actual use of ASA. The next two sections describe the actual use of ASA by individual developers (Section 5) and at build-time (Section 6). Section 7 adds a feedback stage to tune ASA. Section 8 concludes.

Section 1: Software Defense Application

The defense industry–as shown by projects such as Software Assurance Metrics and Tool Evaluation–is paying significant attention to static analysis tools [3]. This paper helps DoD decision-makers, project managers, and developers in deploying static analysis tools to meet their quality requirements.

Section 2: Motivating the Stakeholders

It is not sufficient to procure the ASA and hope that it will be used. All the stakeholders–developers, middle management, and higher management–need to be motivated. Wide acceptance of ASA by the stakeholders is a prerequisite to get the best possible benefits [8].

In reality, some projects do a poor job in motivating the stakeholders. Lichter et al. [8] found that their ASA deployment was not as effective as expected because they did not spend enough time motivating all of the people involved with ASA.

It is difficult to introduce ASA top down, i.e., only driven by management decision [8]. To get the stakeholders support, they need to be convinced that ASA is important and they benefit from it. For example, higher management demands return on investment analysis, which is a challenge.

Quality Consciousness

ASA is primarily targeted at improving product quality. ASA deployment suffers if the project is not quality conscious.

Do you have quality goals? Do you reward developers for meeting deadlines at the expense of quality? Do developers compete on the basis of fewest defects? Do teams compete with each other for the fewest defects?

We all know the famous saying that goes, "What gets measured gets done." If the project is lacking in quality consciousness, the first step would be to institute some metrics aimed at quality. Here are some metrics worth considering:

- Time to reach system test phase
- Number of defects discovered in the system test phase and their distribution across teams
- Time to reach delivery of system to customers
- Number of defects discovered in the field and their distribution across teams

Once the quality metrics are instituted, the next step would be to suggest ASA as a mechanism to improve the team's performance on these metrics.

Demonstrating the Benefits of ASA

Not everyone in the team will be convinced that ASA will lead to improvement of the team's performance on its quality goals [8]. Demonstrating the benefits of ASA using a case study would convince most.

The case study should show that ASA will be effective at detecting bugs typically noticed in system test and field. It must also show that ASA can detect those bugs with less cost and time compared to the current approaches used in the project. An example of such a case study conducted in industry is reported by Baca et al [9]. We recommend that case study as a model worth emulating for others.

Need for a Champion

Many ASA deployments wither away over time because of a lack of developer and management support. There is a need for a champion who takes the responsibility to include ASA in the development process, and who makes sure that it is sustained. For example, while deploying an ASA [10], Microsoft put this strategy in practice and benefited from it.

Section 3: Training

From their experience in deploying ASA, Lichter et al. [8] conclude, "You need sufficient theoretical and technical know-how to apply ASA systematically." Interactions with ASA require expertise and defect consciousness [11] on the part of developers. Expertise is especially required in (1) configuring ASA, (2) triaging, (3) extending ASA, (4) underlying technology of ASA, like data flow analysis and control flow analysis, (5) how to write code so that it is easily analyzable by ASA, and (6) the internal algorithms used by ASA. Developers need to be trained to gain the expertise.

Although some projects do provide training on the use of ASA, it is found to be incomplete. Some managers are not even aware of the extensive training that is required. The challenge here is that often managers find it difficult to arrange for training in these areas either because of a lack of trainers or because of a lack of information. The training challenges may be met through a Center of Excellence (CoE).

The organization should establish a CoE focusing on ASAs. The objective is to have a single point offering the knowledge that is required to use ASA. For example, the software dependability design group at Nortel [12] works with development teams to train and to include ASA in their development process. A CoE will also be useful for evangelizing ASA.

Section 4: Integration Into the Process

As the old proverb goes, "Failing to plan is planning to fail." Many projects simply drop the ASA into the project and hope that it will show its benefits. The single most likely reason why many ASA deployments fail is that the ASA is not properly integrated into the development process [9].

Introducing a new breed of tool into the development process is easier said than done.

"To be successful, the new tool must fit smoothly into the existing process–it has to make a difference but not cause such a disruption that it is perceived as a source of busy work rather than the solution to a thorny set of problems" [13].

We have observed that management tends to underestimate the effort and cost (direct and indirect) of integrating ASA into the development process–when the reality dawns, ASA suffers or gets neglected.

Approaches to Integration

ASA is a form of Quality Assurance (QA). Studies (e.g., [14] [15]) reveal that ASA complements and does not replace existing QA activities like reviews and testing. Integrating ASA into a development process and combining all the QA techniques to get the best of them in the least time possible remains a challenge. Software development projects have tried two approaches and their variations [16]:

1. Running ASA by a dedicated team: Although tried by many projects, a major challenge with this approach is scalability (as, for example, found at Google [16]).
2. Running ASA by individual developers.

Section 5: Running ASA by Individual Developers

In this approach, developers apply the ASA as part of their regular work on a feature or a defect. ASA will be more effective when applied this way. However, it majorly affects many steps of the development process.

After finishing implementation, the developer runs the ASA, weeds out false positives, fixes the real defects, unit tests the changes, submits the changes for peer review, and checks-in the changes. This section considers how ASA is integrated with these steps of the development process, and what challenges await the project management.

5.1 Estimations

To apply ASA, significant time and effort is required by developers. Do your schedules take that into account? Many do not. It is a difficult problem to update estimation models to take ASA into account. Projects should collect metrics to evolve estimation models (see Section 7.1).

If sufficient time is not set aside for ASA application[2], ASA deployment suffers because it conflicts with the deadlines imposed on developers for their work.

5.2 Triaging and Fixing

After ASA runs, it produces a set of defect reports. The developer has to go through each report to separate true defects from false positives–a process called triaging. Once false positives are weeded out, the developer needs to fix the true defects.

Projects vary in how developers triage and fix. In some projects, developers need not fix immediately, but can open a new defect report containing the issues reported by ASA for future triaging or fixing [17]. Often this is done because estimates do not set aside time for ASA or because of too many false positives. We believe this is a wrong strategy and sets a seed for failure of ASA.

On the other hand, some projects mandate that developers must handle all the ASA-reported issues before check-in. If such a mandate is given–especially without setting aside time–it leads to a different problem. Developers may label real issues as false positives or opt for quick fixes [18] just to keep ASA quiet.

For example, developers are known to simply add a NULL check if the ASA reports a NULL dereference issue, although that is not always the appropriate fix.

5.3 Peer Reviews

After unit testing, developers submit the code for peer review. Not all projects have peer review in their development process. It plays an important role in ASA adoption.

For ASA, peer review serves two purposes: (1) Deal with the challenge of making sure that developers applied and handled ASA reports correctly; (2) Detect those defects missed by ASA.

The review package from developer to reviewers should include among others: How was ASA configured? What were the results? How were the ASA defect reports handled? From this package, the reviewers should verify that false positives are indeed false positives, identified defects are fixed properly, and whether ASA should be altered to find more defects, etc.

Based on that review, the reviewers need to focus on detecting those defects missed by ASA. To do that, they need to have a good idea of the strengths and limitations of ASA–which is often where they are lacking.

5.4 Human Factors

As Gerald Weinberg says, "No matter how it looks at first, it is always a people problem."

For most projects, project sociology is more important than technology [19]. This applies to ASA use also. ASA is considered a pure technology, where as it has numerous sociological angles. Managers tend to focus on technology and not on the sociology side [20] with the resultant failure of ASA deployments.

In this section, we consider the role developers and reviewers play in the success or failure of ASA and the challenges they present to management.

5.4.1 Changing Habits is Hard

The main barrier to adoption of ASA lies in the ability of developers to impose the discipline required to make ASA a routine part of their work [17].

Introducing ASA is a change for most developers. Changing habits is hard [13]. Some regard ASA as a nuisance.

Some ASAs are initially difficult to use, but over time developers may start appreciating them. This has been confirmed experimentally [4]: The experimenters created three versions of code with different errors. For model checking, false positives decreased across all three versions because of developers creating better abstractions as the experiment proceeded.

In practice, many developers give up before they come to a stage where they appreciate ASA. Process methodology and project management need to ensure that developers persist with ASA. Management should convince developers that ASA ameliorates the frustrating debugging associated with field failures.

5.4.2 Lack of Motivation

Some projects deploy ASAs ad hoc–they depend upon developer's motivation in using ASA [9]. Some deploy ASA in a planned fashion, but they still leave it to the developer to decide what warnings are important and what are not, etc. Again, a lot of responsibility is with the individual developer.

When so much responsibility is with a developer, whether ASA works or not depends on the developer's motivation. A developer's motivation depends on many factors. Do you feel bad about defects in your code? Developers differ. Where developers feel proud of their work, the possibility that they successfully deploy ASA is high, and vice versa. To ensure that ASA is used by developers, development process should include a rule like no code can be checked in until ASA results pass a set criterion. The criterion could be no defects, no severe defects, or fewer than five minor defects, etc.

Team organization and project policies also have a role in the developer's motivation of using ASA effectively. A specific situation worth considering is the separation sometimes seen between two groups of development: feature development and sustenance. The development group introduces new features and sustenance group fixes defects. For effective ASA, feature developers need to spend time using ASA. But often the policies and goals of development group are such that they are indirectly discouraged to use ASA. Some organizations [21] have the policy where the developer of a feature is responsible for fixing the defects in the feature. If a feature developer is responsible for fixing the defects, then the developer will have to spend time to debug and resolve the testing-reported defects. It will hinder the developer in moving to work on new features. Since ASA-reported defects can be fixed sooner than testing-reported defects [22], developers would prefer to use ASA to shorten their debugging cycles for resolving testing-reported defects.

5.4.3 Reducing Discipline Because of ASA

Because ASA is there to detect defects, will it lead to sloppy coding by developers and less effective review by reviewers? All too often a pre-tested module does not get inspected properly, "Well, that [module] works OK [23]. Why waste time inspecting it?" The situation is analogous with ASA. There is anecdotal evidence that ASA presence leads to some loss of discipline in developers and reviewers (based on our interaction with some project managers). Private self-assessment (see Section 7.2) would help in mitigating this challenge.

Section 6: Build-time Running of the Tool

To complement developers running the tool, periodical (e.g., weekly) ASA is run on the entire codebase. Not all projects have this important step of build-time running of the tool. In this step, the ASA is configured to do deeper analysis compared to developers running the tool. A central team triages the defect reports and opens defects in defect tracker for later fixing by developers.

The results of build-time running ASA are also useful for generating reports and metrics. Although ASAs provide various types of reports, a major challenge is providing reports management can understand and relate to. The reports should show trends of number of defects and defect types across builds. Wherever possible, the reports should correlate ASA-reported defects to defects that managers and users are aware of. For example, maybe an ASA-reported defect could explain a field failure of the system. Such reports enhance the support of both management

and developers to ASA adoption and helps sustain it.

The major challenge in this stage is not fixing the ASA-reported defects. As mentioned earlier (Section 5.2), other stages in the development process also contribute to this challenge.

Not Fixing the ASA-reported Defects

You will get the real benefit from ASA only if ASA-reported defects (after pruning false positives) are fixed [24]. In many projects, the emphasis on fixing ASA-reported defects is minimal; hence they tend not to get fixed [25]. It negates the whole purpose of deploying ASA. We observed that some of the unfixed ASA-reported defects actually turned up in field.

Here are some reasons that contribute to non-fixing of ASA-reported defects [7]:

Delayed detection of ASA defects: The later an ASA defect is detected, the lesser the chance that it will be fixed. That is why ASA defects detected at build-time or by a dedicated team have a lesser chance of getting fixed compared to defects detected by an individual developer.

Allowing ASA defects to accumulate: In some projects, developers do not have to triage or fix the ASA-reported defects prior to check in (see Section 5.2). This leads to accumulation of ASA defects and reduces the probability of fixing them.

Severity and priority of ASA defects not clear: Testing reported defects normally shows that something is broken, and from that it can be ascertained how urgent the fix is (priority) and the consequences of not fixing it (severity). ASA defects normally do not show that.

This challenge can be dealt with in the following ways. First, developers should be encouraged and facilitated to fix the ASA-reported defects before check in rather than deferring them. Second, management should strive for nightly ASA builds rather than weekly builds, although it might mean infrastructure needs to be heavily upgraded for a faster ASA run. Management and developers should determine the right frequency. Third, when the ASA finds a defect in the nightly build, it should be integrated with source code management system to find which recent check in is responsible for this issue. Then it can immediately inform the associated developer for faster and easier fixing.

Section 7: Retrospectives

Many–but not all–projects have retrospectives, where they look back on the project. Retrospectives are important for effective ASA. They help in tuning of ASA and the process.

7.1 Tuning ASA and the Process

Based on the retrospective, ASA should be tuned for detecting new types of defects and for higher performance. The development process needs to be tuned. For example, estimation models need to be tuned to set aside time to use ASA by developers in various activities like triaging and fixing. To do the tuning, the development process should collect related metrics.

7.2 Appraisals

Appraisal of developers is common in software organizations. ASA can provide additional means to appraise. If not used properly, it can backfire. The balancing act is a challenge.

ASA provides many metrics and reports (see Section 6). Many of them relate to defects. The temptation to use them for measuring developer's programming capabilities is high [8]. That is not a good idea [19]. It can easily upset the whole program of deploying ASA and will be counterproductive [8]. Developers either find ways to bypass ASA or apply a coding style that only leads to acceptable results of ASA but not result in overall product quality [8].

Defect counts and their categorization are especially useful for developers. While they should not be used to measure developers' capabilities, it is important to deploy such metrics for private self-assessment [19] and self-learning only. Some projects do not provide such metrics for private use. Instead, they tend to deploy metrics for management use only.

Section 8: Conclusion

ASA is an important tool in the quest towards higher quality. But its deployment is not easy. The challenges include:

- Motivation (e.g., not motivating the stakeholders on why the tool is important and how they are going to benefit from it);
- Training (e.g., inadequate training on the technology that underlies the tool and not raising the defect consciousness of developers);
- Development process (e.g., difficulties in integrating the tool into development process);
- Developers (e.g., developers resenting the additional over head of the tool);
- Project management (e.g., difficulties in scheduling to fix the tool reported defects);
- Performance appraisals (e.g., team dissatisfaction because of appraising engineers using metrics generated by the tool).

Forewarned is forearmed. By being aware of the challenges one may face in deploying ASA beforehand, one can be prepared to deal with them.

Acknowledgments:

We thank Ilan Kumaran, Thomas George, Sujay Gupta, Ravi Kumar B, and Srikanth S for sharing their experiences of ASA deployments and their comments on earlier versions of this paper. The paper particularly benefited from the insightful and detailed comments from the CrossTalk Editorial Board.

Disclaimer:

The statements and opinions expressed in the paper are authors' own and not their employer's.

REFERENCES

1. L Hatton. Software failures-follies and fallacies. IEE Review, 43(2):49–52, 1997.
2. A. German. Software static code analysis lessons learned. CrossTalk: The Journal of Defense Software Engineering, November 2003.
3. Yannick Moy. Static Analysis Is Not Just for Finding Bugs. CrossTalk: The Journal of Defense Software Engineering, September 2010.
4. G. Brat et al. Experimental evaluation of verification and validation tools on Martian Rover software. Formal Methods in System Design, 25(2):167–198, 2004.
5. A. Bessey et al. A few billion lines of code later: using static analysis to find bugs in the real world. Comm. of the ACM, 53(2):66–75, 2010.
6. Paul Black. Static Analyzers in Software Engineering. CrossTalk: The Journal of Defense Software Engineering, March 2009.
7. D T V Ramakrishna Rao, Piyush Jain, and Sathyanand Balan. Why Static Analysis Tool Deploy ments Fail? (And How...). In Embedded Systems Conference (ESC), Bangalore, July 2010.
8. H. Lichter and G. Riedinger. Improving software quality by static program analysis. Software Process: Improvement and Practice, 3(4):235– 241, 1998.
9. Dejan Baca. Automated static code analysis – A tool for early vulnerability detection. PhD thesis, Bleking Institute of Technology, Sweden, 2009.
10. T. Ball et al. SLAM and Static Driver Verifier: Technology transfer of formal methods inside Microsoft. LNCS, 2999:1–20, 2004.
11. D T V Ramakrishna Rao. Defect Detection By Developers. CrossTalk: The Journal of Defense Software Engineering, pages 8–11, March 2009.
12. J. Zheng et al. On the value of static analysis for fault detection in software. IEEE Transactions on Software Engineering, 32(4):240–253, 2006.
13. P. Chandra et al. Putting the tools to work: How to succeed with source code analysis. IEEE security & privacy, 4(3):80–83, 2006.
14. S. Wagner et al. Comparing Bug Finding Tools with Reviews and Tests. In Proc. 17th Int'l Conf. on Testing of Communicating Systems (TestCom05), pages 40–55. Springer, 2005.
15. J. Nazario. Source code scanners for better code. Linux Journal, January 2002.
16. N. Ayewah et al. Using static analysis to find bugs. IEEE software, 25(5):22–29, 2008.
17. Melissa Webster. The Software Quality Imperative. Technical report, IDC, 2005.
18. Russel King. "Do not misuse Coverity please".
19. Tom Demarco. Why does software cost so much? Dorset House Publishing, 1995.
20. T. DeMarco and T. Lister. Peopleware: productive projects and teams. Dorset House Publishing Company, Incorporated, 2nd edition, 1999.
21. S. Maguire. Debugging the development process. Microsoft Press, 1994.
22. Brian Chess and Gary McGraw. Static analysis for security. IEEE Security and Privacy, 2(6):76–79, 2004.
23. J.G. Ganssle. The art of designing embedded systems. Newnes, 2000.
24. Matthew Hayward. The Truth Behind Static Analysis Pitfalls. EE Times, <http://www.eetimes.com/design/embedded/4008210/TheTruth-Behind-Static-Analysis-Pitfalls> January 2009.
25. William Pugh. Making Static Analysis Part of Your Build Process. Presentation to the Silicon Valley Java Users Group, 2009.

NOTES

1. In this paper, we use the terms "The Tool" and "ASA" interchangeably.
2. Even if developers take additional time because of ASA, overall product release time still tends to improve because of reduction in testing and debugging cycles.

ABOUT THE AUTHORS

Piyush Jain, PMP, is a Delivery Manager in Product Engineering Unit of Infosys Technologies Ltd, Bangalore, India. He has more than 17 years of experience in software development. In his role of delivery manager he is responsible for new business development and end-to-end delivery management of projects for global customers. He has provided consulting services to leading networking firms on how to improve engineering efficiencies and effectiveness of tools deployment in product development and testing. Prior to taking up management role, he has had extensive experience in embedded development on networking products with primary focus on system engineering and L3 and L4 protocol development. He is PMP certified and has published papers in PMI India conference and other conferences/forums. He holds a BE in Computer Engineering from National Institute of Technology (NIT), Surat.

Infosys Technologies Ltd.
Electronics City,
Hosur Road,
Bangalore – 560100
Phone: +918041166289
Fax: +918028521695
Mobile: +919880596947
E-mail: Piyushj@infosys.com

D.T.V. Ramakrishna Rao is a Senior Technology Architect in the Product Engineering Division of Infosys Technologies Ltd, Bangalore, India. He has 14 years of experience in software development with primary focus on building high-end networking systems. He has presented and published 15 papers on project management, static analysis, networking, and defect management in international journals and conferences including CrossTalk, IEEE, IETF, and PMI. He created two new static analysis tools to detect interfacing bugs and endian bugs. He has evaluated, deployed, and worked with various static analysis tools including Coverity's Prevent, cppcheck, and Sparse. He recently published the paper "Defect Detection by Developers" in Mar/Apr 2009 edition of CrossTalk. He holds a B.Tech in Computer Science from National Institute of Technology (NIT), Warangal, and an M.Tech in Computer Science from Indian Institute of Technology, Kanpur.

Mobile: +919845554451
E-mail: ramakrishnadtv@infosys.com

Sathyanand Balan is a Senior Technology Architect in the Product Engineering Division of Infosys Technologies Ltd, Bangalore, India. He has 11 years of experience in software design and embedded software development for networking systems. He is a lead architect in a large engineering program for a global customer. He was part of the tools group that rolled out a static analysis tool (Coverity Prevent) for a leading networking OEM in their development process. He has conducted various training programs on board bring up, Network processors (Wintegra), embedded system design and development. He holds a B.Tech in Electronics and Communication from NIT, Calicut.

Mobile: +919845392221
E-mail: sathyanandb@infosys.com