



Corporate Background

CONTACT:

Dave Park
Vice President, Business Development
Coverity, Inc.
3705 Haven Avenue
Menlo Park, CA 94025
650-368-4952
dave@coverity.com

Kathy Keenan
Keenan Strategic Communications
15946 Linda Avenue
Los Gatos, CA 95032
408-358-7358
kathy@keenancomm.com

Background on Coverity, Inc.

Coverity Has the Answer to Software Testing

Software reliability is a serious problem in today's high tech society. Much of the modern world runs on software. Our transportation systems, communications, factories, finance systems and business operations depend on software to function. Software failures can be calamitous. These failures are most often measured by their negative economic impact. In today's society, though, they are often also measured by the number of lives lost. The increased reliance of businesses, industries and government organizations on computerization has resulted in a demand for increased functionality: programs that once were measured in thousands of lines of code are now measured in multi-millions of lines of code.

Software testing tools have not kept pace with the growing complexity of software programs and their operating environment. The National Institute of Standards & Technology (NIST) issued a report in May 2002 entitled, "The Economic Impacts of Inadequate Infrastructure for Software Testing," stating that "[Software] testing tools are still fairly primitive. The lack of quality metrics leads most companies to simply count the number of defects that emerge when testing occurs."

Using studies conducted in two industries (transportation/aerospace and financial services), NIST estimated the financial impact of inadequate software testing on the U.S. economy at \$59.5 *billion annually* (not even including catastrophic failures). The NIST report estimates that "feasible" improvements to testing infrastructures could save \$22.2 billion out of the \$59.5 billion. According to the report, the feasible improvements are (1) the detection of defects earlier in the development cycle and (2) more precise error diagnoses.

It requires little imagination to see that there is a critical and growing need for software testing tools and infrastructure that can address these improvements. Coverity, Inc. meets that need with **Deduce**, a toolset that detects defects in the source code as soon as they are introduced by the software developer and pinpoints the root cause and location of each detected error.

Deduce

Deduce is unique amongst software testing tools because of the following critical features:

- Catches defects at compile time, during the development phase (not at run-time, during integration testing, or in production)
- Requires no manual testing, code modifications or changes to existing processes
- Detects a wide range of defects, including several classes of bugs that no other tool can detect
- Scales to millions of lines of code within a small fraction of the build time
- Examines every path through the code instead of just a few representative paths thereby testing 100% of the code regardless of size or complexity
- Reports fewer false error reports than any other compile-time tool
- Automatically customizes and retargets to each new code base

- Can be easily extended with custom checks for company-specific defects
- Finds complex defects using interprocedural dataflow analysis, automatic abstraction, and statistical inference techniques

These benefits make **Deduce** unique in the software testing market. Its performance and results are exponentially beyond the abilities of any other competitive tools.

Static, Compile Time Tool: The earlier defects are caught and corrected, the less expensive they are. **Deduce** catches defects at the coding stage. As the table below suggests, after the coding phase, the cost of each defect increases exponentially:

Relative Cost to Repair Defects When Found at Different Stages of Software Development
(Example only. X is a normalized unit of cost and can be expressed in terms of person-hours, dollars, etc.)¹

Requirements Gathering and Analysis/Architectural Design	Coding/Unit Test	Integration and Component/RAISE System Test	Early Customer Feedback/Beta Test Programs	Post-product Release
1X	5X	10X	15X	30X

Requires no manual testing, code modifications or changes to existing processes: Because **Deduce** analyzes the source code directly, software developers are not required to perform laborious manual tests to make the tool effective. In addition, because the tool is self-customizing, comprehensive out of the box, and integrates seamlessly with existing development processes, software developers do not need to change the way they work to accommodate the tool. **Deduce** adapts automatically to each source base by inferring many of the rules and standards under which each piece of code was developed. Thus, **Deduce** does not need to be programmed or configured to differentiate legitimate code sequences from defects.

Detects a wider range of defects: **Deduce** detects these classes of defects:

- Memory leaks
- Resource leaks
- Memory corruption
- Array/buffer overrun
- Illegal pointer access
- API misuse
- Deadlocks
- Race conditions
- Security vulnerabilities
- Dead code
- Unused variables
- Useless update operations
- Uninitialized variables

¹ “The Economic Impacts of Inadequate Infrastructure for Software Testing,” National Institute of Standards & Technology, May 2002, Table 5-1, p. 5-4

- 64-bit compliance
- String manipulation errors

Scalability: Deduce is the only software testing tool that can scale to millions of lines of code while maintaining 100% path coverage. Other tools can only test a small proportion of today's complex programs.

Examines every path through the code: Today's software provides complex functionality and, thus, requires complex source code. The other tools in the market can either only examine those few representative paths exercised during testing or only maintain full path coverage for small programs. The result is that the vast majority of the pathways through the code are untested.

Results in fewer false positives: Deduce yields only one false positive for every five real bugs. Other source code analysis tools may report as many as 50 or 100 false positives per real bug. This high ratio of false positives renders these tools frustrating, time-consuming, and unusable to software developers.

Extensible and customizable: Every software project is designed and implemented using different techniques and different idioms. The result is that any effective source code analysis tool must be able to adapt to the features of each customer's source code. Deduce meets this challenge on three levels: it can be configured to recognize these idioms, it can statistically infer the properties of these idioms, and it can also be extended with new types of checks to detect specific defects that arise from these idioms. Most often, these added checks detect defects that are similar to those that have hit in production and are identified as likely sources of future failures. This extensibility allows customers to reap continuous returns over the product's lifespan.

The advantages to the developer and its customers are clear:

- Faster time to market
- Reduced development and test costs
- Reduced post-sales support and service costs
- Increased customer satisfaction
- Reduced cost of ownership

The Technology

Deduce is based on patent pending source code analysis technology developed by a team of researchers headed by Dr. Dawson Engler, an assistant professor of computer science at Stanford University. An early research prototype of Deduce detected over 2000 defects in Linux, including several hundred security vulnerabilities. Dr. Engler and his students have produced over twelve publications on the research underlying Coverity's technology.

Deduce runs on all major platforms. At present, Deduce is available for testing C code. It will soon be available for C++ and Java testing. The product uses patent pending statistical

inference technology that allows it to recognize the rules by which a piece of code was built, thus eliminating the need to manually program the product for each new customer.

Deduce runs either as a companion to the nightly build or on every developer's desktop. Bugs detected by the toolset are stored in a persistent database that allows for easy diagnosis with an intuitive, web-based user interface that integrates seamlessly with most bug tracking and source code control systems.

The technology behind **Deduce** is described in further detail in Coverity's "Deduce: A Technical Background."

The Market

Coverity's initial product offering is aimed at the C code marketplace, which includes both embedded software and large legacy software projects. While most new software is developed in other languages, there is both a huge reservoir of legacy C code in active use and a continuing demand for development in C in the embedded marketplace. As companies upgrade their products or systems, this legacy code is modified, requiring additional testing. Coverity estimates that the C code test market will eventually account for \$30-\$50 million in annual revenue. The C++ test market is expected to add another \$30-\$50 million of annual revenue.

In 2004, Coverity plans to introduce C++ and Java versions of its product. Coverity estimates that the large and rapidly growing Java test market will eventually yield over \$100 million of annual revenue.

The Competition

While no other software testing products can match the capabilities of **Deduce**, there are several other software testing tools on the market. The comparison matrix below shows how the other tools compare with **Deduce**:

<p>Gimpel Software Lint (FlexLint, PCLint) Description: Finds software defects at compile time.</p> <ul style="list-style-type: none"> • High rate of false positives (typically 50-100 FP's to one real defect). • Need code annotations to find deeper, interprocedural errors. • Works only with vanilla code that uses standard libraries and interfaces out of the box. 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Low rate of false positives (typically 4 real bugs to 1 FP). • No code annotations or modifications necessary to find complex bugs. • Automatically retargets to different code bases with custom interfaces and conventions.
<p>Rational IBM Purify Description: Finds memory problems at runtime.</p> <ul style="list-style-type: none"> • Works at runtime when the entire program is executable. • Requires test cases and manual testing. • Only monitors code paths that are executed (typically 5-10% coverage). • Limited to finding memory problems. • Cannot diagnose the cause of an error in many cases. • Not extensible. 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Runs at compile time before the program is even run. • No test cases or manual testing required. • 100% path coverage without compromising performance. • Detects a wide range of defect types. • Shows root cause and precise location of all reported errors. • Easily extensible with custom checks.
<p>Parasoft Insure++ Description: Similar to Purify. See comparisons for Purify.</p>	<p>Coverity Deduce</p>
<p>CodeWizard Description: Runs at compile time and enforces general coding guidelines.</p> <ul style="list-style-type: none"> • Does not find bugs. Just enforces some coding guidelines and clean coding practices such as “do not use the ?: operator.” 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Detects problems that cause real problems such as crashes, performance degradation, or security vulnerabilities.
<p>Reasoning Illuma Description: Static analysis auditing service.</p> <ul style="list-style-type: none"> • Not a tool; clients ship code to Reasoning. • Range of defects detected limited. • Not customizable or extensible. • Requires 1-2 weeks turnaround to get the results. 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Tool that integrates into build environment. • Detects a wide range of defect types. • Customizable to clients' needs. • Results are virtually instantaneous.
<p>Polyspace PolySpace Verifier Description: A compile time simulator that detects memory and arithmetic defects.</p>	<p>Coverity Deduce</p>

<ul style="list-style-type: none"> • Range of defects detected limited. • Prohibitively slow; does not scale to large code bases in the 100k+ lines of code. • Not extensible. • Produces hundreds of false positives per bug. • 	<ul style="list-style-type: none"> • Detects a wide range of defect types. • Fast: scales to millions of lines and runs in fraction of the build time. • Customizable and extensible. • Produces 5 bugs for each false positive.
<p>Programming Research QAC</p> <p>Description: A compile time tool that detects dangerous coding practices, enforces guidelines, and checks common coding standards. A cross between Lint and CodeWizard (see comparisons for both).</p> <ul style="list-style-type: none"> • Mainly a guideline, standards checker. 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Detects deeper errors that affect reliability and security. Deduce can be extended to check any similar guidelines or standards.
<p>Secure Software Rats</p> <p>Description: A code auditing service (like Reasoning) specifically for security problems.</p> <ul style="list-style-type: none"> • Downloadable tool that runs superficial checks that look for dangerous calls to strcpy() and other known, insecure interfaces. • For good results, Rats requires professional services like Reasoning; clients must ship code. 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Detects deeper errors based on dataflow analysis that tracks values through the program logic. • Integrates into build; no services required.
<p>Klocwork InForce</p> <p>Description: Detects software defects through automated source code analysis.</p> <ul style="list-style-type: none"> • Limited to mainly memory problems. • For security problems, runs superficial checks that mainly warn against suggested guidelines. 	<p>Coverity Deduce</p> <ul style="list-style-type: none"> • Detects a wide range of defect types. • Detects deeper errors based on dataflow analysis that tracks values through the program logic.

The Founders

Dr. Dawson Engler headed the Stanford Research Laboratory team that developed the technology underlying **Deduce**. Dr. Engler is an assistant professor in Stanford's Department of Computer Science. His work on operating systems and compilers is well-known, due to his many published articles and presented papers on these topics. Dr. Engler holds a Ph.D. in Electrical Engineering and Computer Science from MIT, where he won the George Sprowls Best PhD Thesis Award for "The Design and Implementation of a Prototype Exokernal Operating System." Dr. Engler's involvement in Coverity is consultative rather than operational.

Benjamin Chelf was a founding member of the Stanford Research Laboratory team that architected and developed the technology for **Deduce**. In addition to co-authoring numerous research publications, he also wrote for *Linux Magazine* to discuss various programming techniques for Linux programmers and contributed to the gcc open source project. He holds an M.S. and B.S. in computer science from Stanford University and is currently on leave from the Ph.D. program at Stanford to head up the engineering efforts at Coverity.

Dr. Andy Chou was a member of the Stanford Research Laboratory team that developed the technology for **Deduce**. Author of a number of publications on error detection and checking systems, Dr. Chou received his Ph.D. in Computer Science from Stanford University and his B.S. in Electrical Engineering from UC Berkeley. A recipient of a National Merit Scholarship, he is a member of the Eta Kappa Nu EECS Honor Society.

Seth Hallem is Coverity's CEO. Working with the Stanford Laboratory Research team under Dr. Engler, he was one of the principal architects and implementers of Coverity's source code analysis product. Graduating with a B.S. in Computer Science with Phi Beta Kappa honors from Stanford, Mr. Hallem is currently on leave from the Ph.D. program at Stanford while he helps to build Coverity's customer base.

David Y. W. Park is Vice President of Business Development. Mr. Park, also a Ph.D. candidate at Stanford in Computer Science, has worked with a number of high tech start-ups, such as Virtualscape, Inc., FogDog.com, Silvan Networks and Sanera Systems. Author of several publications on software verification and model checking, Mr. Park holds a B.S. in Computer Science, with distinction, and is the recipient of several awards, including Phi Beta Kappa, Stanford President's scholar, President's Award for Academic Excellence, Tau Beta Phi, the Terman Engineering Award and a National Science Foundation Fellowship. Mr. Park is an award-winning pianist and has soloed with numerous symphony orchestras in the U.S. and in Europe.