CALTECH/MIT
VOTING TECHNOLOGY PROJECT

A multi-disciplinary, collaborative project of
the California Institute of Technology – Pasadena, California 91125 and
the Massachusetts Institute of Technology – Cambridge, Massachusetts 02139

THE SAVE SYSTEM: SECURE ARCHITECTURE FOR VOTING ELECTRONICALLY
Existing Technology, with Built-in Redundancy, Enables Reliability

Ted Selker
MIT

Jonathan Goler
MIT

Key words: ***electronic voting, voting security, voting system architecture***

VTP WORKING PAPER #12
October 22, 2003
Revised January 4, 2004

The SAVE System: Secure Architecture for Voting Electronically

Existing Technology, With Built-in Redundancy, Enables Reliability

Ted Selker, Jonathan Goler
Caltech/MIT Voting Project 10/22/2003, revised 1/4/04
Massachusetts Institute of Technology

Abstract:

Existing technology is capable of yielding secure, reliable, and auditable voting systems. This system outlines an architecture based on redundancy at each stage of the ballot submission process that is resistant to external hacking and internal insertion of malicious code. The proposed architecture addresses all layers of the system beyond the point when a voter commits the ballot. These steps include the verification of eligibility to vote, authentication, and aggregation of the vote. A redundant electronic audit trail keeps track of all of the votes and messages received, rendering a physical paper trail unnecessary. There is no single point of failure in the system, as none of the components at a particular layer relies on any of the others; nor is there a single component that decides what tally is correct. Each system arrives at the result on its own. Programming time for implementation is minimal. The proposed architecture was written in Java in a short time. A second programmer was able to write a module in less than a week. Performance and reliability are incrementally improvable by separate programmers writing new redundant modules.

1.0 Introduction:

Computation systems are designed to be the most reliable systems for tabulation. By their very character, they are not subject to the kinds of mechanical failures that plague traditional voting equipment. Despite the advantages electronic systems offer several papers and well-known authors [8] have raised fears, uncertainties and doubts as to the effectiveness and trustworthiness of electronic voting equipment.

It is possible to create electronic voting systems that, by their very nature, are secure, reliable and trustworthy. An analysis of types of possible attacks, the possible scope of these attacks and the likelihood that they will occur is a place to begin. The architecture should address these vulnerabilities.

This paper will demonstrate an approach for using existing technologies in the form of computers and their networks to effectively and efficiently handle the voting process. Indeed, the proposed approach would solve current problems while improving efficiency.

Specifically this paper will lay out an n-version [12] type of voting system that addresses the issues of:

- Accurate transmission and recording of voter intent, resulting from an architecture that performs fault detection and correction.

- Prevention of outside tampering or hacking, especially involving the threat of changing votes.

- Prevention of malicious internal fraud involving changing or specifically developing malicious voting system components.

- Interception of vote transmission or falsifying the contents of messages between system components.

Voting is a complex procedure. This particular paper will not address the important difficulties of registration, local administration of voting, the process of voting itself or other important problems. In subsequent papers we will address the voting process as a whole. It is of note that an increase in voter turnout has coincided with the introduction of new electronic voting equipment.

The current and past approaches to voting depended on administrators who assembled paper votes by hand to be counted by computers or people. In all voting to date polling-place operations and processes have required careful administration and observation to be reliable. The typical approach to security in voting is to have bipartisan human oversight of each critical step of an election. Such human observation can be fairly effective, but a combination of the complexity and the scale of the process can lead to errors that result in inaccurate vote counts.

2.0 Approach:

This approach for designing a secure and reliable system relies on an analysis of attacks, accidental failures and intentionally malicious code. While we believe that voting systems to date have not been tampered with maliciously, they rely on the same processes that other voting systems rely on for security, oversight and care. Up until now it has not been easy for would-be hackers to obtain access to voting systems. However, it is possible to obtain the code illegally in a variety of ways, so this method is insecure against the most likely sophisticated hackers.

As the world becomes more reliant on computers in voting systems, it will become more important to secure them from new internal and external attacks. Showing systems to work and testing them under many circumstances are important and useful steps. In many cases testing voting systems ahead of time has found problems that could be resolved. Still, in the case of all current voting systems, there is really no way of determining if a particular voting system does exactly what the manufacturers claim it does.

One concern about the Internet is that electronic transmissions can be held up or slowed down for one reason or another. A system that communicates electronically can batch the communication for later transmission, use land telephone lines to communicate the information, or use cell phones or satellite phones as alternate communications modes to make communication reliable. Secure transmission can be achieved with a variety of communication media.

The dangers of power outages have successfully been addressed in Brazil where the computer-based voting system relies on batteries that last 14 hours[11]. The question of messages being intercepted is one of simple encryption; the issue of changed messages would be dealt with using redundancy, cryptography and message authentication codes (MACs) to ensure integrity. Unfortunately, votes recorded on paper have no protocols to deal with loss or alteration. Thus, paper balloting is still vulnerable to interception, alteration and deletion. The architecture for this voting system is designed to demonstrate that Internet voting can be safer and more reliable than voting has ever been.

Traditional schemes for making processes robust include following best practices of software design and testing: solid, simple systems that can be analyzed and overseen either by being available for anyone to view open-sourced or by being made by experts and kept inaccessible. The open-source approach is the opposite of the secrecy approach, but, given varying threat models, both have been used as valid and useful methods.

2.1 Threat Model

We assume a variety of attacks are possible, by individuals involved in the production, distribution and use of the election systems, as well as by outside hackers.

The Evil Development Company: The danger of losing contracts due to faulty equipment has been a constant concern of election technology companies. They have small close-knit development organizations and review their work together. These are all safeguards for their systems. Still, there is concern that either as an individual or organization, the author of a voting system might insert malicious code. This code could change votes, delay or drop votes, or produce intentionally incorrect tallies. In addition, the code could flood the rest of the system with invalid messages, damaging the performance of the system. Included with this type of threat are the distributors of the code, as well as the hardware providers.

External Hackers: To date, external hackers have not had enough time and access to voting systems to hack them. Systems such as Diebold's, which was found on an open FTP server in source code form, fail to hold up to scrutiny [13]. With experience with the protocols and enough time-- if a system is communicating over open lines--outside hackers could modify, delete and/or record messages between system components. If the system is not over an open network, this threat is of far less concern.

Malicious Voters: A voter gaining access to the system could try to vote more than once, or as another person, or try to steal the votes of other individuals. While to date care has been taken to limit access to smart cards or other methods to opening a poll, it is

possible and important to improve access control to the voting act. Coercion is always a danger; technology can be used to allow or to reduce coercion as well.

2.2 Security

Typical methods of implementing voting security focus on: (1) Isolating the process so that no one can see or change a vote; (2) Building in review. Typical governmental applications have relied on isolation and confidentiality as a security approach, labeled by some in industry as "security by obscurity." The most modern conversations about security describe the value in oversight either by expert review, redundancy or open-source methods .

In the first case, confidentiality as a security approach has worked well. Potential hackers have not had access to the software and have not known what to do when they have access to it. Software systems for voting are relatively new. People attempting to compromise security in elections have not been sophisticated hackers yet. The approach of prohibiting access to things that should remain secure has been very successful. A final key point is that voting conditions change with time, many ballots being finalized within a day of the election. The concerns that would alter a specific ballot tend to be more local and time dependent than concerns about trying to bring down a country or economy.

Certainly secrecy itself is a key method of preventing the wrong people from gaining access to sensitive data. However, secret and closed systems present the serious problems of Easter eggs and backdoor approaches. While these problems may seem far-fetched, they have to be taken seriously, because it is possible that a set of voting machines in a particular precinct could be turned into zombies by setting them to a testing mode.

Such tampering, of course, would be easily uncovered due to the discrepancy with the number of registered voters casting ballots. However, if, for instance, 4 to 7 officials at a balloting place agreed to work together, they could cast ballots after the voters left.

While these methods seem to have worked for a long time, there have been breaches of security in many elections. Most have been isolated incidents and have had little impact on the national level. Thus they have merited little national scrutiny. This changed in 2002.

However, with the prospect of large-scale, undetectable fraud by using a single system, it becomes more important to have an n-version system, with full auditing along the process.

3.0 Architecture overview:

Designing secure systems requires attention to many levels. Our approach begins by ensuring that there is no single point of failure after the ballot leaves the eyes of the voter. The security starts with the general system concept and goes down to specific ways that the code is written to avoid introducing reliability problems at any stage. The key advantage of this n-version architecture is that structurally there is no way the whole system can be compromised without compromising a very significant number of the parts.

The principle of redundancy is central. It enables the system to continue to work even if there is a failure somewhere along the line. Having multiple programs that process each stage of the ballot casting can establish improved reliability; regardless of how they are written, regardless of who has written them, and regardless of whether they are the same code. Because these versions can be transmitting over different networks, the system is more reliable. Because these are different programs, subverting one of them would not affect the others and still would ultimately enable an accurate vote to be cast. More importantly, if different people and organizations write these modules, intentional tampering of one module (discussed as the 'Evil Equipment developer'), such as putting in an Easter egg (a secret module of code that invokes undocumented functionality), would not affect the integrity of other modules.

N-versioning on the actual code can only take security so far, while they can avoid common flaws amongst the modules, common flaws in the underlying hardware and operating system might still compromise the code for all modules. Thus it is essential to place different modules on different pieces of hardware, preferably with small, real time operating systems, not large complicated systems such as Windows.

Certainly, to be sure these new measures are effective, the system will have to be tested beforehand. By forcing each module to comply with the abstraction-function behavior that we specify, the architecture will be uniformly black-box testable to ensure compliance with the protocol. Clearly, any certification of the system must include a thorough formal review of the code. In addition, there must be no difference between a test vote and a real vote, as far as the software is concerned.

In our system we separate the aspect of user interface from the rest of the voting system. The intent is to allow user-interface designers to build the best possible user interface for every type of user. A user who is blind might use a different interface than a user who has little motor control. However, we have dealt with a remedy to the possibility of a person entering an unintended vote. In an n-version user interface system could be a series of digital cameras would snap digitally signed pictures of the ballot on the screen as a way of providing an additional audit trail independent of the rest of the system. Alternatively, different device drivers that read the screen could record their observation to provide the independent audit trails.

Once the user has filled out the ballot, the next step is to authenticate the voter. A back-end system checks the person's name against a database of registered voters. The registration server signs the vote, along with various electronic witnesses which see only hashed data. The "witnesses" sign the vote to indicate that a valid voter as assessed by the registration server cast it. At this point the signed, blinded ballots are then sent to a variety of aggregation servers to be counted.

4.0 A Demonstration Implementation

The redundancy system involves an n-version modular voting architecture designed for implementation in a variety of environments, ranging from a single geographically and electronically isolated location to a globally distributed system.

The architecture is composed of four principal layers: A **User Interface** connection to ensure capture of votes, the **Registration** to assure that the user is valid, the **Witnesses** layer to create an auditable and secure record, and **Aggregators** to establish an actual outcome. Additionally, feedback layers give the voter proof that the vote was established and recorded, as well as another layer between the registration systems and the aggregators, known as a mix-network [14], which can perform random secure shuffles of ballots to further guarantee anonymity in the final count.

XML is a protocol that is available on all modern computer platforms. It is human readable and allows for definitions of modules by virtually all programmers [4.6]. Communication between the components is provided by an XML messaging protocol Each level of the architecture logs the incoming and outgoing messages to aid in auditing the system. The modules are split up into small modules so that each of them contains less than 1000 lines of program code. This separation will enable a faster and more thorough review process, while limiting the number of bugs that can be introduced.

4.1: The User-Interface

Perhaps the most vital component of any voting architecture is the user interface. This architecture allows for the user-interface modules to be developed independently of the rest of the architecture. This flexibility permits faster progress incorporating human factors' research in improving the voting experience. Other work [5] establishes user-interface quality assessment. This architecture recommends that all available effort be put into building a user interface that is extremely effective for efficient and accurate voting.

The user interface takes two inputs: the interface definition and the blank ballot. Both of these components are XML documents. The interface definition describes the way in which the UI is to render a ballot.

The user interface collects the votes of the user, as well as the registration data. It then encrypts the ballot using keys from the aggregators. The registration information is added to the encrypted ballots, and the resulting packages are then transmitted to the registration system.

When the user approves the ballot, there will be an n-version type system of digital cameras mounted to the DRE that can take a picture of the ballot, or redundant device drivers that observe the actual ballot on the screen and record the contents. To prevent the production of an actual receipt, the picture can only include the ballot itself, and no other features, so that either the ballot is showing entirely or the ballot is obfuscated, so a user cannot put his or her face in the way, or put a piece of paper saying "Alice Bobster" in the way. Nevertheless, since the digital photograph backup is not used as the primary counting mechanism, this problem is of little concern for coercion and vote buying.

4.2: The Registration System

The registration system is the center of this voting architecture. The registration server has access to the roster of all registered voters. When the registration receives a ballot package containing registration information and an encrypted ballot, it looks at the database, checks to see if the user is valid, and then makes an entry in the database checking off the user as having sent a vote to the aggregator.

Each registration module extracts the encrypted ballot, signs it, and then sends it to the witness modules (4.3) for their signatures. Once the witnesses return their signatures, the signatures can be appended to the encrypted ballot. Then the whole ballot package (without individual identifying information) is shipped off to the aggregators.

4.3: The Witness Module:

The witnesses are the simplest of the modules. They take as input an encrypted ballot and produce a signature. Signatures are produced using MD5/RSA. [9] The ballot is digested, and a hash is produced, which when combined with the witness's private key, produces a number that, as far as we know, can only be produced by the holder of the private key. Witnesses do not maintain a record of the ballots coming through them, as they are meant to be lightweight implementations, preferably using separate databases or smart cards so they can be handled easily. Witness modules are to be provided by independent organizations (political parties, watchdog organizations).

4.4: The Aggregator Module:

The aggregator module takes encrypted ballot packages as input. The packages contain the encrypted ballot and a series of signatures produced by the registration system and witnesses. The aggregator parses the signatures and uses the witness public keys to verify the signatures. The aggregator then determines that a set threshold of signatures verify and then decrypts the ballot. Once the ballot is in plain text, the selections are parsed and recorded. Both the encrypted and plain text versions of the ballot will also be stored in a repository.

4.5: Messaging Protocol:

The messaging protocol is based on XML. The Document Type Definitions (DTDs) are included as Appendix A. Communication between modules is simple. The listening

module waits for connections; the signaling module then initiates a socket connection, opens an output stream, then an input stream, and writes a string containing the command to the listening module. The module then does its processing and writes a string of commands indicating its response. The output stream is closed first, and then the input stream is closed.

Standard sockets are used to connect between various components. The prototype implementation uses the Java Socket and ServerSocket classes that are conveniently provided by JDK 1.4.

5.0 Security and Reliability via Architecture

The architecture of this system uses modularity and threshold agreement for fault and hack tolerance. Redundant audit trails enforce certain security and reliability. Modularity is an important cornerstone of any system that can be scrutinized. Each component we have developed is a few hundred lines at the most. And most of that is simply placing the data into a database. The tightness of the software code allows it to be quickly and easily certified to do what it is intended to do, for its compliance with the protocol that demands plug-and-play interaction with the rest of the architecture, as well as code that is easily viewed by outside agencies to determine its accuracy and correctness. Additionally, the separation into interoperable modules creates a voting system that could be modified in one aspect without affecting the certification of another aspect or component. This modularization dramatically lowers the cost in time and money for certification as systems are created and improved. The most important part of modularity, however, is that by separating the modules by steps we can analyze security in each stage.

Each module keeps track of the other modules it is supposed to send and receive information from, as well as the public keys of those servers. Modules are defined by a contract that indicates what they are to send, receive, and process. By creating a standard contract, anyone can write to the standard and plug a module into the working environment. The architecture itself enforces security and reliability while improving maintainability.

In the previous stage there are n systems, each of which provide a piece of data. We cryptographically verify the data for each of them, checking their keys and signatures and obtain k verified answers. Using the following rule, we can ensure that most systems are behaving:

> For some integer t, where $n > t > n/2$, consider the result
> to be valid if $k > t$.

This is an implementation of an n-version threshold system, which only accepts something as being true if a threshold of "yeas" is achieved. By setting n and t to reasonable values (we believe that if each layer has seven components, it will require the compromise of four of the units at each layer to compromise the election), a few cheaters

will never compromise the end result. But, if there are too many mischievous systems, the system will identify the danger of compromised results.

The thresholding characteristic provides improved reliability and verifiability while enforcing security and preventing unscrupulous development companies from writing poorly behaving systems. Still, unscrupulous or careless developers could write some of these systems. By having several different implementations of each module, we guard against a few of these systems being compromised from the inside intentionally or from outside attackers.

5.1 Redundancy in Eliminates Need for Voter Verifiable Paper Trail

Much of the controversy lately has revolved around whether or not to include a voter verifiable paper trail. The simple answer is that there should not be a voter verifiable paper trail. Paper is far easier to lose track of than an audit trail kept on different computers. In the 2000 election, there were many reports of entire precincts worth of paper ballots, as well as ballots that were found weeks later in the backs of trucks.

Adding paper to electronic voting system undermines the public confidence in electronic voting systems. Rather than spending research dollars on attempting to make electronic systems more reliable and trustworthy, money is wasted on paper, which has only contradictory proof of effectiveness.

If the failure rate of two systems is independent, then there is very little likelihood of four out of seven systems failing. Of course, we must verify that systems to not contain common vulnerabilities and flaws. To ensure the elimination of common vulnerabilities, the source code for each system will be passed through a commonality checker such as PLAG or SMAT [18-21]. This system tests for similarities between code, and is commonly used to detect cheating in assignments. In addition to the source code, it is prudent to examine the compiler used and in fact, varying the actual compilers used aid in preventing the external introduction of common vulnerabilities.

5.2 Cryptographic Security

A number of cryptographic protocols aid in security. This system uses the following cryptographic schemes to achieve improved security and verifiability:

1. All modules are issued their own private keys.

2. Modules digitally (RSA) sign all of their transmissions, so that their data transmissions are protected against wiretapping or man-in-the-middle attacks.

3. All transmissions are also maintained with Secure Sockets Layer (SSL), the most reliable approach for sending information today, protecting the data itself from being read.

In addition to these fairly standard ways of protecting data, we employ a more specific set of schemes for protecting voting data. When sending a ballot to a registration system, the architecture must assure that the voter is valid. The registration system, on the other hand, should have no knowledge of how the vote was cast. The filled ballot needs to be separated from the access to vote. Encryption additionally prevents others from seeing the voter's vote through the registration system. For governments which keep ballot data together (for eliminating walk-around fraud with absentee ballots), the following procedure is necessary: take a ballot, encrypt the vote, send the vote along with the registration data to the registration server, and have the registration server return the same encrypted ballot, but with a signature attached. That signature is known as a blind signature. What is known as the "blinding factor" is an additional layer of encryption that the voter can decrypt [10]. This is known as a blind-signature scheme, and it permits the system to obtain a signature for a plain text ballot even if the signer does not know the plain text. It is analogous to putting a piece of carbon paper in a sealed envelope and having someone sign the outside. The signature will appear on the inner piece of paper. Blind signatures allow a system to maintain the privacy and security of the ballots.

In addition to the registration server signing and validating the ballot, a number of other modules must sign the ballot as redundant verification. These are known as witness modules, and various watchdog organizations as well as the political parties could provide them. They could be smart cards or pieces of software, and they all would provide a blind signature to ballots that are considered valid. In that way, when ballots are recorded, they are recorded with the signatures of all of these witnesses. The witness scheme provides enhanced verifiability and trustworthiness.

Aggregators simply decrypt and store ballot information. Care must be taken to make sure that they properly validate signatures on the ballots and that they are properly placing the data into the repositories that they should be in. Having multiple aggregators allows us to "recount" on the fly. Aggregators provide redundancy of data and verify the entire process up to that point.

5.2 Distributing the responsibility and fallibility

Voting administrators and voting equipment makers sometimes say, “We don't care who wins as long as it isn't a close race.” If we are resolved to count each and every vote, this is not an appropriate outlook. (See Voting what was and what could be [1] for discussions of how votes get lost.) Votes obviously have been lost on occasion, sometimes several boxes full. In paper systems, we do not immediately make 5 or 9 or 11 copies of a ballot to ensure proper handling. In our electronic system, the overwhelming redundancy provides many more ways than paper for checking a vote tally.

From time to time, voting administrators and voting companies get into trouble for various violations of election law. Even in 2002, a salesman was indicted for giving

kickbacks to elections officials. The charges were dropped as he turned in the officials he bribed.

Those who make and operate voting machinery must be somewhat trustworthy. On whom can governmental election agencies depend? An excellent model can be found in Brazil were the trust in elections was reestablished by involving two research institutions that helped establish equipment reverence platforms for the voting equipment manufacturers. The choice of equipment was then made in full view of the public by voting officials. In this country this might involve politically separate research institutions, such as Caltech, MIT, or Stanford, or governmental research organizations, such as NIST, which have no political ties nor are funded by any particular constituency. However, we must accept the possibility that some voting systems will be maliciously manipulated, and we must guard against that possibility by using our redundant model.

While the Brazil system eliminates certain potential abuses, the voting system architecture outlined in this paper takes the distributed approach to security to another level. Instead of relying on a single company to provide a system in a region, we are relying on the distribution of people to avoid fraud. The architecture becomes more secure when more people are involved. In some cases, too many people involved produce sloppy and buggy code, however, by the very architecture involved, this system becomes more secure and reliable as additional modules are added. Even if some of the organizations have their own political agenda (and act on it), the architecture will maintain the integrity of the system.

Multiple groups create versions of the same part of the architecture. It must be easy for an election administrator to pick n of these systems, and run them seamlessly. Thus, there should be a common registry of these modules, and an effective means of

6.0 Conclusion

This voting architecture provides a means to vote over open networks in a way that is reliable, secure, and private. Due to its modularity and common specifications, it is easy to implement, improve and it is inexpensive. The system uses COTS equipment for the all of the back end systems, reducing the likelihood of fraud with the system components as well as keeping the cost down. These innovations make it particularly attractive for implementation as state budgets are increasingly tightened.

7.0 Future Work

N-Verision Programming can be a powerful tool for improving electronic voting security. The next steps to creating an N-Version Programming voting system are using it to secure the user interface and using it to secure backend vote tabulation and storing.

Much work remains to be done in the voting architecture field. Our group is working on developing effective user interfaces and improved registration systems. We are also examining ways of providing verifiable feedback to users, but in a way that does not compromise the confidentiality and receipt-freeness requirements of voting. To address

the need for clear, balanced ballot forms, we are developing an artificial-intelligence based system to help inform ballot designers.

References

[1]Caltech/MIT Voting Technology Project. *Voting: What Is, What Could Be,* July, 2001.

[2]McLachlin, M. *How many Buchanan supporters are there?*, The Palm Beach Post, November 12, 2000.

[3]Brady, H. *Report on Voting and Ballot Form in Palm Beach County*, November 2000.

[4]United States General Accounting Office. *Elections: Perspectives on Activities and Challenges Across the Nation*, October, 2001.

[5] Selker, T. "User Interface and Ballot Design as Part of an Improved Voting System," May, 2001.

6 Fischer, E. *Voting Technologies in the United States: Overview and Issues for Congress,* March, 2001.

7 Foote, E., and Smith, J. *Revitalizing Democracy in Florida: The Governor's Select Task Force on Election Procedures, Standards and Technology,* March, 2001.

8. R. Rivest. Security in Voting Technology, House Testimony May, 2001. (Text: http://theory.lcs.mit.edu/~rivest/rivest-may-24-01-testimony.txt)
9. R. Rivest. IETF RFC 1321: The MD5 Message-Digest Algorithm. 1992.
10. Fujioka, Okamoto, Ohta. A Practical Secret Voting Scheme for Large Scale Elections. AUSCRYPT 92. pp. 244-251.
11. R. Saltman. Adopting Computerized Voting in Developing Countries: Comparisons with the US Experience. CPSR Newsletter, Vol.16, No.1, Winter 1998, pp. 13-16, Computer Professionals for Social Responsibility.
12. A. Avizienis, "The Methodology of **N**-Version **Programming**," Chapter 2 of Software Fault Tolerance, M. R. Lyu (ed.), Wiley, 23-46, 1995. http://citeseer.nj.nec.com/avizienis95methodology.html
13. T. Kohno, A. Stubblefield, A. Rubin and D. Wallach. "Analysis of an Electronic Voting System" July 2003.
14. D. **Chaum**, "*Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*", Communications of the ACM (24)2, pp. 84-88, 1981

15. J. C. Knight, **N**. G. Leveson and L. D. S. Jean, "*A Large Scale Experiment in **N**-Version Programming*", in 15th Int. Symp. on Fault Tolerant Computing (FTCS-15), Ann Arbor, Michigan, USA, pp.135139, IEEE Computer Society Press, 1985.

16. Jie Xu, Brian Randell, Alexander B. Romanovsky: A Generic Approach to Structuring and Implementing Complex Fault-Tolerant Software. Symposium on Object-Oriented Real-Time Distributed Computing 2002: 207-214

17. A. Zorzo, J. Xu, and B. Randell, "Experimental Evaluation of Fault Tolerant Mechanisms for Object-Oriented Software", in Proc. 23rd Brazilian Software and Hardware Seminars, (Recife), pp.457-68, 1996. http://citeseer.nj.nec.com/zorzo96experimental.html

18: Finding Plagiarisms among a Set of Programs with JPlag - Prechelt, Malpohl, Philippsen (2000)

19: JPlag: Finding plagiarisms among a set of programs - Prechelt, Malpohl, Phlippsen (2000)

20: Identifying Similar Code with Program Dependence Graphs - Krinke (2001)
21. Measuring Similarity of Large Software Systems Based on Source Code Correspondence. Yamamoto Et Al. IEEE Transactions, March 2002.