

LATEST SOFTWARE SOLUTIONS IN DEVISING NEW APPLICATIONS

Tanvir Ahmed

Department of Computer Science & Software Engineering, Grand Asian University
Sialkot, 7 KM Pasrur Road, Sialkot Punjab, Pakistan
Email: drtawaraich@gmail.com

Rai Samee Ullah

Department of Computer Science & Software Engineering, Grand Asian University
Sialkot, 7 KM Pasrur Road, Sialkot Punjab, Pakistan
Email: sameetatla@gmail.com

Ishfaq Ali*

Department of Computer Science, Faculty of Sciences, National textile university
main campus Faisalabad Pakistan Email: ishfaqalimalik@gmail.com

ABSTRACT

From past few years there has been a dramatic increase in the research and development of Software Engineering applications. With more number of users and professionals in industry, business, games, education and in administrative fields, more complex and large scaled applications are emerging in the market. Due to the growth of new fields of computer applications, complex and immense development methods are used to meet the demands of huge market. The software plays a major role to meet the future challenges of exploded population where space management will be also a great issue. The development history of Software Engineering applications is becoming more sophisticated day by day. Today's software consists of a set of different components reused from a library already developed or partially developed in-house. Each component provides a different functionality as a single unit. In software applications development, such components are often deployed as per architecture derived for distributed and heterogeneous environments. This paper provides a review of developing such software applications which are becoming the catalyst of change in the quality of human life. It is envisaged that current problems of software solutions, variety of software applications and future challenges in Software Engineering will be an area of users' interest. The research in this paper focuses upon the variety of applications to predict the kind of future society and richness of Software building blocks in form of components, rules, techniques, expertise and knowledge.

Keywords: It Trends, Software Refactoring, Knowledge Carrying Software, Software Evolution, Software Quality, Developer Productivity, Software Evolution, Empirical Studies

Introduction

There are distinct challenges in software practical problems. It includes successful design, bug free production or flow and maintenance of useful software system. Therefore, we face ever new requirements to gain advanced features every day passing. The limitation of software quality rests upon poor design, poor performance, instability and mismatching requirements of the end product. The source of problem of difficulties is distributed that is actually easy to identify yet difficult to cure. Our main problems arise from the change of scale in alphabetic proportions. We don't know how to reduce them. We need to move towards goals without knowing the base of additional requirements. We need to organize the tools of productions efficiently that will help to permit the lower work ratio and full cost of growth. We should have knowledge of developing special purpose system and how to integrate with a larger one [32] .

What are the new applications in software engineering? To plan the future we should understand first past and present work in software applications with respect to software engineering. By this way we will be able to indicate where we were and where will we be, in future timeline. We can predict what we will do in future. The objective of software development research is to use the history of a software system to examine its present status and predict its future development [11, 5]. Such an examination needs a lot of information about a system to give exact approaches about its history. Usually researchers derive their data from versioning systems, as their warehouse holds the objects already developed or used history. In the past producing high quality software was not the primary objectives for most developers. They were generally focused upon generating functionality and automation through language dependency, distribution and integration capabilities. We require presenting more incremental and frivolous approach that developers can use in their development processes. So for finding the answer of the above mentioned questions we will go through the history of software and software engineering till now current situation and will dig out current trends, there impacts on software production.

Today's software applications are moving towards new types of software architecture development like object management systems and service oriented architectures. They required some new paradigms and security features. To secure and develop these service oriented systems, there is a lot of work to do. Thus, we will move by curiosity-driven research method. The main focus will be about the past and future upcoming challenges in software engineering.

HISTORY OF SOFTWARE INVITATION

After a survey of software engineering history from 1956 to date, three main eras emerge in the surface for producing software applications. Each era is differentiated by a different set of objectives, practices, methods and tools, lessons learned and problems known. The periods may be related and are less different from each other than the three eras. Any effort to explain a 40-year era on a few pages means selecting highlights and comprehensive details. It is like a sketch of the situation. [34, 35, 36]

Era I - Mastering the Machine

The era of mastering machine was the first era from 1956 to 1967. It is the era where the term “software engineering” had not been discovered yet. This time is considered powerful era due to ownership by manufacturers. In other words, it was the forming era of the computer industry.

The Batch Period

Batch –oriented systems were the first computers that achieved important use in the software industry for scientific and commercial applications. This form of the process resulted from the two classical Input Output and storage media that is punched cards and magnetic tape. The major objective of software development was to develop the limited hardware resources (like storage and processing power) in the most advantageous way. Processing power was measured in hours that make total job infeasible. Consequently, lots of effort was expended on manually tuning code, mostly written in Assembler. The initial compilers are the example of manually tuning code that (the7090 FORTRAN compiler developed by Backus and his team). Since compiling times for logically sized applications might also last hours, most alterations were applied to the object code. Recompilations just occurred in particular time periods, generally at night. Engineers learned one key lesson during this period. For most applications high level languages of the FORTRAN or COBOL type could produce sufficiently performing code. Engineers also identified the extensive compiler times.[]

The Interactive Period

Period in the mid of the sixties, development tools were the man focused area of engineers and the research community. The major objective was to decrease the clerical aspects of coding and to remove the need for modifying object code. The non-volatile hardware environment was supported this goal. During this period, incremental compilation, automatic test data generation techniques, source level control and debugging were used in software technology development. These technologies were supported by the theory of on-line source code management and version control. The tools developed throughout this period are called lower CASE tools, was a simple typewriter (or teletype device) is an example of online development.

The accomplishment of this period was to maintain programs in source code, rather than object code level. The problems were described in terms of “code and fix” approach and “Spaghetti Code”. Furthermore, people started to identify software development is more than to just coding. In fact, people realized that pre-coding and pre-testing risks are high risks in development life cycle. After This motivation a new era started up in software development.

Era II - Mastering the Process

During this era software engineering was established as a field the duration of this era was 1968 to 1982. Most of the ideas of this era are still relevant today. But we called them historic ideas because at that time they were more dominant. Software pricing was also introduced by hardware sellers in this era and an independent software

industry started.

The Process Period

This period mostly known or recognition of the “software crisis”. Although original crisis still there, others have recognized a series of crisis, they are following the one another. First time in industry due to software crisis, people realize the development process is more important as compare to. Later on, the subject was also discussed greatly in academic circles and proper curricula, conferences and journals were established. The major objective followed during this period was to minimize development risks and to improve quality and efficiency.

The major directions of this era were to identify the software project failures, estimating cost data gathering per activity and analysis of software error data. The waterfall model and many improved versions of it were introduced. Check and balance principals were applied where sufficient. At the resultant independent test organizations came which later, which later may have developed into a quality assurance function. A landmark event of this period was code inspections that were more effective then testing. Test case and design documentation trends also introduced in this era The first self-evaluation procedures were introduced. The key lesson learned during this period was that the entire development process is most important instead of final product the quality of a product cannot be assured by only looking at the final product.

The Formal Period

The man focused of this era was to adopt formal methods to increase the quality of software’s and improve productivity by achieving automation. This makes clear why practitioners emphasize on formal methods. Formal methods are applicable in software specification, transformation and verification. It is a less demanding solution in the case of verification. Formal methods and verification in particular have had their successes mostly for a little security or safety critical programs.

For large projects, it usually uses for expressing the requirements in the mathematical model. Different relevant expressions and non-functional requirements belong to requirements specified. Cross domain data interrelationships, the usability, performance and compatibility requirements are the example of requirement specification. Formal methods are not capable of addressing requirement analysis or any communication between professionals and non-professional users. In first era highest risks was laid in the pre-coding phases, in second era pre-design activity risks were highlighted. So above mentioned statement justified to view the following periods as a new era.

Era III - Mastering Complexity

Since 1983 this era has considered as a milestone event of the first two periods. The first periods can also consider as history or contemporary periods. Traditional control of hardware over software was ended in this era. Hardware has become a sub servant to software, also demonstrated by the stumbling of some companies that were previously strong as hardware vendor’s e.g (IBM, DEC).

The Structured Period

This era was the original era of structured methodologies, CRT displays and all-points-addressable devices in this category are the breakthrough of this era. Many graphic designs were introduced in this era. Finally, software designers wanted to draw rather than writing programs with graphical design notations. Although graphical notation considered as a good means to communicate with non-programmers and they are formal in nature (as shown by Harel. The appearance of CRTs was the new software entity, specifically the Graphical User Interface (GUI) tools. Unexpectedly, software engineering gives the impression to feed its own industry. It was the starting point of the CASE elation. Initially, only individual small tools like craftsman were accessible, now the whole things are integrated into a worldwide workbench. The planned opportunity presented by the CASE technology by these programs maintained at the design level rather than at the source code level. A CASE tool and all kinds of non-related material soon became too costly, the learning and maintenance costs were too high. The structured methods also showed up one more serious problem. The main problems addressed in this era code reuse and the reengineering of existing systems.

The Object-oriented Period

In early 1968 (McIlroy) pointed out the potential of software reuse. But it was not seriously addressed until eighties. In fact object-oriented and software reuse also the current era problems. If previously software is unplanned then for reuse software implementation should be different differently then. If object-oriented concepts are applied it will be modularized, encapsulated and portable. Object oriented programming purely based on the concepts of inheritance, polymorphism and encapsulation are available.

“Ad hoc polymorphism and parametric polymorphism were originally described in Fundamental Concepts in Programming Languages, a set of lecture notes written in 1967 by British computer scientist Christopher Strachey. In a 1985 paper, Peter Wegner and Luca Cardelli introduced the term inclusion polymorphism to model subtypes and inheritance. However, implementations of subtyping and inheritance predate the term 'inclusion polymorphism', having appeared with Simula in 1967.”[1].” In object-oriented programming (OOP), **inheritance** is when an object or class is based on another object or class, using the same implementation (inheriting from a class) specifying implementation to maintain the same behaviour (realizing an interface; inheriting behaviour). It is a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a hierarchy. Inheritance was invented in 1967 for Simula”[2]. ”In programming languages, **encapsulation** is used to refer to one of two related but distinct notions, and sometimes to the combination thereof:

A language mechanism for restricting access to some of the object's components

A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.”[3, 4, 5, 6].

In the same way as in the case of structured methods, the object-oriented methods were first useful for coding, then to design and later to requirements analysis.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

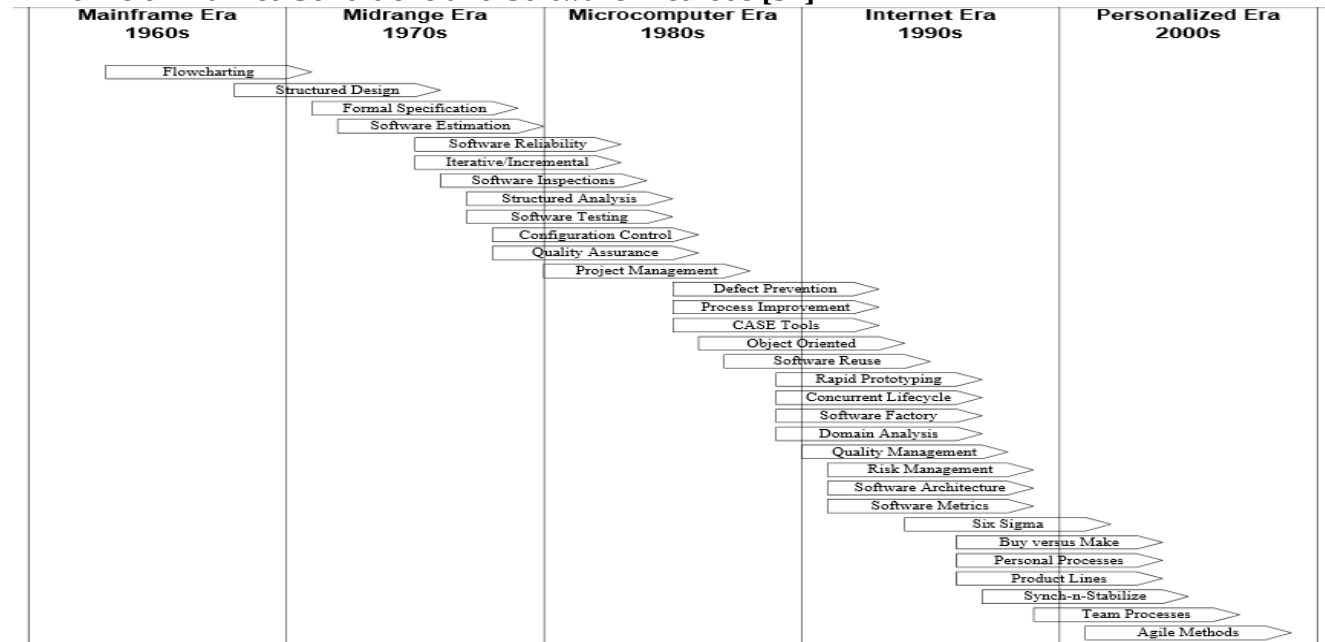
Opposing to the structured methods, there is a flawless transition possible between analysis, design and implementation. We can achieve Code reuse by development of class libraries or (mainly in the case of GUI intensive programs) through application frameworks; design reuse is being discussed generally under the thought of design patterns.

Engineering is the process of converting a non-object-oriented system to an object-oriented structure, thus building a new technological base for reuse. Object-orientation has the likely to construct software systems that should be decomposed. In this era problems identified that development methods are generally related to the increased demands on testing and software understanding or realization. Cycle time was also main important issue, particularly still expanding because creative people see opportunities for modern products.

In short words, we can say last four decades of software development are surely only the beginning of our field and represent a short period if calculated in historic terms. However, it is supportive to try to recognize some threads that could form a road map into the future. As historians typically argue, to study history only purpose to learn about the future.

History identifies 32 major classes of software methods that have emerged over the last 50 years [37]. In next page a short history sketched in some tabular format.

Timeline of Market Conditions and Software Methods [37]



Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

Market Conditions

- Mainframe computer systems stabilize
- Mainframe operating systems born
- High level computer programming languages born
- Large scale computer programming becomes problematic and software engineering is born
- Transistor is commercialized
- Government and commercial bureaucracies dominate market

Market Conditions

- Computer systems becoming smaller
- Midrange computer systems born
- Software crisis for computer programming emerges
- High level computer programming languages become too numerous
- U.S. DoD standardizes high level computer programming languages
- Relational databases emerge
- Semiconductors are commercialized
- Microprocessors 'computers on a chip' are born
- UNIX and C are born
- Telecommunications industry deregulated

Market Conditions

- Engineering workstations emerge
- Cathode ray tubes emerge
- Disk drives become smaller and faster
- Personal computers emerge
- DOS emerges
- Apple Macintosh emerges
- Graphical user interfaces emerge
- Spreadsheets become killerapp
- Videogames become killerapp
- Microprocessors become killerapps
- Mainframes become obsolete
- High tech trade deficit reaches epic proportions

Market Conditions

- Microsoft Windows becomes killerapp
- Shrinkwrapped software bundles become killerapp
- World wide web emerges
- Netscape becomes killerapp
- HTML and Java emerge
- Midrange computer systems and engineering workstations become obsolete
- Broadband Internet service emerges
- Liquid crystal displays and wireless networking emerge
- Personal digital assistants and digital cameras emerge
- Bill Gates and Marc Andreessen become Wall Street wonderboys
- Internet gives much needed boost to U.S. economy

Market Conditions

- Cathode ray tubes become obsolete and laptops are the killerapp that never was
- Personal digital assistants and digital cameras become killerapps
- Everything is miniaturized
- Global industry slowdown in lieu of 9/11
- Michael Dell is new Wall Street wonderboy
- Enron executives cook the books and defraud shareholders
- Broadband internet service and wireless products become the norm
- High tech outsourcing to India comes to fruition after 20 years of prophecy
- Outsourcing threatens U.S. economy

TIMELINE OF MARKET CONDITIONS AND SOFTWARE METHODS [37]

Summary of relationships between market conditions, software methods, and business objectives

Era	Market Conditions	Problem	Method	Goal	Savings
Mainframe	• Bureaucracies use computers • Early programs in binary • Programmers are Math PhDs	Unreadable programs	Flowcharting	Document programs	Maintenance
		Complex programs	Structured Design	Program using subroutines	Development
Midrange	• Telecoms, engineering firms, and academia use computers • Number of projects explode • Productivity, cost, and quality are pervasive issues • Large projects are broken into smaller pieces • Most computer programs inoperable • Software maintenance costs are astronomical • Programmers are engineers and physicists	Error prone programs	Formal Specification	Verify programs early	Maintenance
		Costly programs	Software Estimation	Control costs	Development
		Failure prone programs	Software Reliability	Test important programs	Maintenance
		Long schedules	Iterative/Incremental	Program using subprojects	Development
		Poor code reviews	Software Inspections	Remove most defects	Maintenance
		Unwritten requirements	Structured Analysis	Document requirements	Development
		Inoperable programs	Software Testing	Test all programs	Maintenance
		Lost programs	Configuration Control	Inventory programs	Development
		Unmet requirements	Quality Assurance	Evaluate requirements	Maintenance

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

Microcomputer	• Medium and small-sized businesses use computers • Fast computers and graphical user interfaces ideal for automated software tools to document customer requirements and designs • Automated scheduling tools abound but discipline of project management doesn't take hold • Firms try and solve productivity and quality problems with reusable computer programs • Programmers are computer scientists	• Overrun schedules • Expensive testing • Unpredictable projects • Manual documentation • Numerous subroutines • Redundant subroutines • Undefined requirements • Long phases or stages • Manual lifecycles • Slow learning curves	• Project Management • Defect prevention • Process Improvement • CASE Tools • Object Oriented • Software Reuse • Rapid Prototyping • Concurrent Lifecycle • Software Factory • Domain Analysis	• Control schedules • Reduce appraisal cost • Standardize management • Automate documentation • Group subroutines • Standardize subroutines • Solicit early feedback • Use overlapping stages • Automate lifecycles • Use program specialists	• Development • Maintenance • Development • Development • Development • Development • Development • Development • Development • Development
Internet	• Computers are fast, cheap, easy-to-use and reliable, and one or more computers per household are the norm • Internet technologies like Netscape and HTML obsolete all prior computer languages • Productivity, quality, and automation take a back seat • Project management attempts to make a last stand, though it's rejected as too inflexible • Most software can be purchased now and doesn't have to be built anew • Millions of websites are created by anyone with a computer and some curiosity	• Late verification • Numerous problems • Redundant designs • Inconsistent measures • Inconsistent quality • Redundant applications • Poor management • Redundant products • Late programming • Poor team management	• Quality Management • Risk Management • Software Architecture • Software Metrics • Six Sigma • Buy versus Make • Personal Processes • Product Lines • Synch-n-Stabilize • Team Processes	• Verify requirements early • Reduce problems early • Standardize designs • Standardize measures • Achieve high quality • Buy commercial programs • Teach management skills • Standardize products • Begin programming early • Teach team management	• Maintenance • Development • Development • Maintenance • Maintenance • Development • Development • Development • Development • Development
Personalized	• Computers small and wireless	• Inflexible programs	• Agile Methods	• Create flexible programs	• Development

SUMMARY OF RELATIONSHIPS BETWEEN MARKET CONDITIONS, SOFTWARE METHODS, AND BUSINESS OBJECTIVES [37]

HALLENGES OF THE SOFTWARE INDUSTRY

Software has an important role in many fields of life. With more number of users and professionals in industry, business, games, education and in administration fields, the development of software engineering become more complex and structured. Due to the complexity and immensity of software development we need to adopt some professional and improved ways instead of old or traditional approaches of software engineering. However, the traditional software engineering development was able to address the challenges of increasing complexity and dependence in the system. for the copepod These software development cope with complexity and adapt quickly to changes. We cannot achieve this goal if each time we develop a product from scratch. The solution of this problem according to [2], is reusability and component-based development (CBD), which seems to be the best approach to software engineering development. Daramola and Bamigbola [17] defined component-base development as

the conceptualization of software systems as a composition of individual reusable parts (components), that are integrated for coherent interaction in order to achieve the specified requirements of a system. They maintained that component-based development is a software componentization that involves the dividing software systems into components for the purpose of easy development, deployment and maintenance. Crnkovic, Stanford, and Larsson (2000) identified a software component as a modular, deployable and replaceable part of a system that encapsulates and exposes a set of interfaces. On the other hand, an interface according to Daramola and Bamigbola – 2006, [17] specifies the set of services provided by a component. However, it is a approach that encourage higher level abstraction that is similar to industrial component-based assembly line procedures. In practice, software systems are built in component-based development by assembling components (parts) already developed (e.g. COTS) and prepared for integration. **Rainer (2008)**, identified and summarized software development challenges as that of communications, collaboration and human issues. However, some of the challenges in component-base development are listed below

Problem of compatibility, components maintenance and upgrading: The main principle of compatibility is to replace components easily with new or old parts of components thus component should be reusable they should be compatibility between various versions of the components. For system maintenance and up gladness it's also applicable. It becomes more difficult to upgrade or maintain components for various system evaluations if compatibility does not exist but we can reduce these challenges by selecting the right people with skill, experience and right communication ability

Configuration management challenge:

Configuration management is a biggest challenge. Due to the heterogeneous nature of these systems comprises from multiple components with different behaviours. The configuration management is not only of technology but of business also. The solution to this challenges to gathering candidate of proven managerial / technical experience and tools. Finding the components which may be used in the system requires high level of partnership among the candidate and this is a times complex.

Quality control and profitability issues:

Component-based development is a collection of different outsourced components. These components would have come from different providers so quality control becomes more difficult. Correspondingly, copyrights, maintenance and upgrade of components become more complex. This complexity and outsourcing of several components reduces the net profit.

Component generic, Efforts and Lead-Times:

Develop a generic component is very difficult because its cover different aspects of reusability. It's requiring more efforts and lead time on the other hand demand and components interface varied. While some components require a high level of abstraction, others may require the interface to be on a more detailed level. These types of different requirements produce different challenges in development .some

other challenges also there like Legal issues, Standardization and certification issues, Software metrics, Availability of procedures for verification, testing and Checking of component-based systems, Migration between different platforms, Development environment tools e.t.c.

However, the solution to the above challenges is selection of skilled experienced people (candidates), use systematic approach to component-based development at the process and technology levels. Develop on-line communication among team members. Another challenge is the problem of evolutionary changes such as creation of new departments, splitting of existing ones etc. that may have occurred after the development of the system has commenced. These kinds of challenges will surely cause disruption of the initial design of some of the components and hence resulting to emergence of new interfaces and methods. This type of situation therefore calls for need for integration and interoperability among components.

FUTURE CHALLENGES

Now a day's software is revolutionary developable. In this position software is careful to provide a degree of flexibility which is wonderful particularly for software developers, for example new version releases of Amazon. If this fact is a pointer of the general way software development will look like in the future, it shows good processes for these particular systems of this field. So the major challenge is to maintain rapidity in general, software development with different versions. Some of future challenges are generic and some are domain specific.[37]

Domain Specific Challenges

Scalability

Interconnected systems, distributed systems, ultra-large scale systems are the main application domains that face these challenges. Different users can access data from different locations so maintain scalability is a major problem of future applications. Currently available solutions do not cope up the need of scalability.

Challenges:

Required new techniques that support multi-language systems

Cope with massive amounts of data (e.g. metadata, programs, processes, models, tests, languages, tools, documentation)

Merge dissimilar data sources (e.g. version repositories, databases, file systems, bug tracking systems, mailing lists, developer fora)

Merge dissimilar technologies and paradigms

Maintain co-evolution of interconnected systems.

Software migration / re-engineering

Any domain where "legacy" systems need to be upgraded to newer technologies

Legacy systems are too expensive and required new technologies

Existing legacy systems are using "Wrapping" as short-term solution for workaround that does not have long-term benefits

Upcoming technologies will be tomorrow's legacy!

Challenge -

How to migrate/re-engineer legacy systems (and their data) with cost-effective, resource-limited way?

How to make sure that the resulting system has the desired value and functionality?

How to migrate to new technologies and paradigms?

How to use software transformation techniques to automate the migration process?

Required good software process models for migration

Upgrading software frameworks

ERP systems (e.g. SAP, Microsoft, Oracle, ...), CMS systems, or an customization software or framework are the application domains.

For vendors, up gradation and customization is a major problem and creates conflicts

Customers bear from vendor lock-in, which threatens develop of their systems

Challenges:

Required techniques to address the upgrade problem

Guarantee data reliability and protection after an upgrade

Runtime evolution and dynamic updating

Telecommunication, distributed systems, finance, internet applications any domain that requires some degree of high availability is a Application domain for upgrading one system some time it becomes more essential to shut them other systems

Challenge

How to securely update/modify a software system throughout its execution?

How to construct in a control system to choose when and how to modify?

How to achieve dynamic reconfiguration for component-based, service oriented and distributed architectures?

Generic Challenges

Software quality improvement and quality assurance

Software quality mean software should be bug free and should fulfil the user requirements without any trouble

Challenge

Non-functional requirements

Quantifiable and observable quality problems from managers and users side

Effort estimation and change impact analysis

It is very difficult to analyse the effect when requirements changed in term of cost men power etc

Challenge

Provide non-intrusive instrument that fulfil these questions answers

When should an accepted change request be planned?

What is the predictable (local and system-wide) impact of a exacting change?

How much time and resources does it take to make a particular change?

When and where should one reorganize the software to make a planned change easier?

Integrate evolution support in software development processes and tools

Current software development tools do not provide satisfactory support for software development

Challenge

Provide better languages, formal methods, and tools that provide more precise support for incremental change

Address maintenance and evolution issues during preliminary development to produce longer-lasting and durable software systems

Support co-evolution of requirements, design, code, tests

Focus on non-functional attribute and quality Need for integrated methodologies to calculate, control and develop these aspects of software

Empirical research in software evolution

Problem Empirical research in software evolution is inflexible due to various aspects like lack of subjects, pressure to validity and lack of industrial data

Challenge

How to get industrial data to achieve empirical studies?

How to obtain statistically important results?

How to set up a trial in such a way the other researchers can replicable?

How to discover the key problems maintainers face?

RECOMMENDATION & FUTURE EXPECTED BENEFITS FROM NEW APPLICATIONS

In this section we will review some benefits of the future development in new application of software engineering. These benefits encompass, though often implicit, themes Oriented Software Development, these are the 'cross cutting concerns' of the Search Based Software Engineering world; advantages that can be derived from almost all applications at various points in their use.

Generality and Applicability

One of the main features of future applications is search based optimization there are few areas of Software Engineering activity to which search based optimization remains unemployed. There are two primary tasks that should be applied in search based approach:

Representation of the problem

Fitness function that can capture the optimization objectives

If these two tasks are accomplished together, then we can get fruitful results in search based optimization techniques. Software has no physical manifestation it's based on intangible material such as logic information and processes so the intangibility makes the optimization difficult. It's also difficult to find out the putative fitness function, but authors and software engineers are continuing to find ways to bring new Software Engineering subareas within the remit of Search Based Software Engineering.

Scalability

Scalability is another challenge in software engineering in term of results there are so many approaches that are attractive and well-designed in the laboratory, applicable due to lack of scalability. One of the attractions of the search based model of optimization is that it is naturally parallelizable Genetic algorithms, being population based, are also naturally parallel; the fitness of each individual can be computed in parallel, with minimal overheads. Search algorithms in general and SBSE in particular, therefore, offer a 'killer application' for the developing model of ubiquitous user-level parallel computing.

Robustness

Robustness is an important solution of some software engineering applications. Search based Software Engineering, focusing on the production of the fittest possible results. So, many application areas need solutions in a search. Research communities have attention about it due to its natural.

Feedback and Insight

In Software engineering false perception is often the cause of major errors, misunderstood Specifications, poor communication of requirements and hidden assumptions in designs. Search Base Software Engineering can address this problem. Distinct human-based search, programmed search techniques carry with them no prejudice. They automatically scour the search space for the solutions that best fit the (stated) human assumptions in the fitness function. This is one of the central strengths of the search based approach. It has been widely observed that search techniques are good at producing unexpected answers.

CONCLUSIONS

At the start of this article it was suggested that our future is in our hands. The sense of that statement should now be clearer. This paper has tried to make a case for the importance of curiosity-driven research, as a complement to the problem-driven research that has been predominant in the past. But it is the software engineering community that must make a decision upon the significance of this type of research, and must support it in solid ways, such as tolerant its authority in its featured publication venues. Such acceptance is definitely not assured, however. It is clear, for example, that there will be an increasing flow of problems that originate in practice, and there is no doubt that there will be a steadily increasing need for our community to grapple with them. Correspondingly, there are a steadily increasing number of venues for the presentation of such papers, and the publication of archival research results originating from the domain of practice. If these publication venues choose to insist upon the publication only of results of problem-driven research, and if these venues decide to deny publication to curiosity-driven research, then the direction of our community will have been decided. Fortunately, these venues are all controlled by our own research community, however, and thus these decisions are not mandated upon us, but can be chosen by us. A pessimistic assessment of the current situation is that curiosity-driven research papers are rarely accepted by these publication venues,

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

and that there seems to be a decided preference for papers driven by the needs of practice. There is nothing wrong with this, if the decision to favor such papers, essentially to the exclusion of papers of the other kind, is the result of careful consideration of what is best for the health of the community. On the other hand, there seems to have not been any debate about this, leading to the perpetuation of a research direction and route that were begun decades ago. It would be a great shame if the future of research in software engineering were decided by default and inertia, rather than by consideration of what we as a community think is best for us. A debate on that subject is needed, and now seems to be an appropriate time for such a debate. The solicitation and appearance of this paper for this venue is a very encouraging sign. One can only hope that it signals the beginning of consideration of some fundamental questions that could form the core of an enduring scientific discipline of software engineering.

REFERENCES

- C. Strachey - Fundamental Concepts in Programming Languages
<http://www.itu.dk/courses/BPRD/E2009/fundamental-1967.pdf>
- Mike Mintz, Robert Ekendahl (2006). Hardware Verification with C++: A Practitioner's Handbook. United States of America: Springer. p. 22. ISBN 0-387-25543-5.
- Michael Lee Scott, Programming language pragmatics, Edition 2, Morgan Kaufmann, 2006, ISBN 0-12-633951-1, p. 481: "Encapsulation mechanisms enable the programmer to group data and the subroutines that operate on them together in one place, and to hide irrelevant details from the users of an abstraction."
- Nell B. Dale, Chip Weems, Programming and problem solving with Java, Edition 2, Jones & Bartlett Publishers, 2007, ISBN 0-7637-3402-0, p. 396
- John C. Mitchell, Concepts in programming languages, Cambridge University Press, 2003, ISBN 0-521-78098-5, p.522
- Pierce, Benjamin (2002). Types and Programming Languages. MIT Press. ISBN 0-262-16209-1. p. 266
- Wm. Paul Rogers, Encapsulation is not information hiding, JavaWorld.com, 05/18/01
- Thomas M. Connolly, Carolyn E. Begg, Database systems: a practical approach to design, implementation, and management, Edition 4, Pearson Education, 2005, ISBN 0-
- Crnkovic, I. 2001:2,4: Component-Based Software Engineering-New Challenges In Software Development, Software Focus, vol. 2(14), Sweden.
- Crnkovic, I., Stafford, J., and Larsson, S. 2000: Composing Systems From Components; IEE Conference And Workshop On Engineering Of Computer-Based Systems, Lund University, Lund.
- Daramola, J. O. and Bamigbola, M. O. 2006:17,19: The Attraction And Challenges Of Software.
- Componentization: An Experience Of Auto Budget, The Journal Of Computer Science And Its Applications, Vol. 12, No. 1, Olaejinson Ventures, Lagos.
- Gartner Group 2001: Workshop on Component-Based Software Engineering. Sourced: <http://www.gartner.com> Larsson, M and Crnkovic, I. 2002:61:

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

- Challenges Of Component-Based Development’ Journal Of Systems And Software vol. 3.
- Mrva, M. 1997: Reuse Factors In Embedded Systems Design, High Level Design Techniques Dept. Siemens A. G.,Munich, Germany. Rainer, Otterbach, 2008: Trends, Challenges and Solutions For Automotive Software Development, User Conference, North America.
- Szyperski, C. 1998. Component Software-Beyond Object-Oriented Programming, Addison Wesley, London.
- C. Alexander, Notes on the Synthesis of Form, Harvard University Press, Cambridge, MA, 1964.
- J. Estublier, D. Leblang, A. van der Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber, “The Impact of Software Engineering Research on the Practice of Software Configuration Management”, ACM Transactions on Software Engineering Methodology, 14, 4, Oct. 2005, pp. 383-430.
- J. Goguen, J. Thatcher, and E. Wagner, “An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types,” in Current Trends in Programming Methodology, V. 4, Data Structuring, R. Yeh (ed.), Prentice-Hall, Englewood Cliffs, NJ, 1978, pp. 80–149.
- B.H. Liskov and S.N. Zilles. “Specification Techniques for Data Abstractions” IEEE Transactions on Software Engineering, v. 1, #1, 1975, pp. 7-19.
- Mary Shaw. Prospects for an Engineering Discipline of Software. IEEE Software, November 1990.
- Evertz L, Epple U (2013) Laying a basis for service systems in process control. In: Proceedings of the 18th international conference on emerging technologies & factory automation
- K. W. Hamlen, G. Morrisett, and F. B. Schneider, “Computability classes for enforcement mechanisms,” ACM Transactions on Programming Languages and Systems, vol. 28, no. 1, pp. 175–205, January 2006.
- K. W. Hamlen, G. Morrisett, and F. B. Schneider, “Certified in-lined reference monitoring on .NET,” in Proc. ACM Workshop on Programming Languages and Analysis for Security, June 2006, pp. 7–16.
- F. B. Schneider, “Enforceable security policies,” ACM Transactions on Information and System Security, vol. 3, no. 1, pp. 30–50, February 2000.
- M. Jones and K.W. Hamlen, “Disambiguating aspect-oriented security policies,” in Proc. 9th International Conference on Aspect-Oriented Software Development, March 2010, pp. 193–204.
- K. W. Hamlen and M. Jones, “Aspect-oriented in-lined reference monitors,” in Proc. ACM Workshop on Programming Languages and Analysis for Security, June 2008, pp. 11–20.
- M. Sridhar and K. W. Hamlen, “Model-checking in-lined reference monitors,” in Proc. 11th International Conference on Verification, Model Checking, and Abstract Interpretation, January 2010, pp. 312–327.
- T. Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design. New Jersey: Prentice Hall, 2005.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

- E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 65–104, 1999.
- B. Thuraisingham, S. Baber, Robert L.: *Software Reflected: The Socially Responsible Programming of Our Computers*, North-Holland, Amsterdam, 1982.
- C. Alexander, *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, MA, 1964.
- 'Formal Methods and the Sociology of Proof', in J.M. Morris and R.C. Shaw, eds, *Proceedings of the Fourth Refinement Workshop* (London: Springer, 1991), 115-124.
- [Commentary] 'The Fangs of the VIPER', *Nature*, 352 (8 August 1991), 467-68.
- 'Computers, Formal Proof and the Law Courts', *Notices of the American Mathematical Society*, 39 (1992), 1066-69.
- A. Pretschner, M. Broy, I. Krüger, T. Stauner: "Software Engineering for Automotive Systems: A Roadmap", in *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007.
- David F. Rico. "Short history of software methods." Retrieved March. 09, 2015, from <http://www.davidfrico.com/rico04e.pdf>