Software Engineering: Reflections on an Evolving Discipline

Ingeniería de software: Reflexiones sobre una disciplina en evolución

David Garlan[1]

[1] Professor of Computer Science and Director of Software Engineering Professional Programs. Institute for Software Research, School of Computer Science, Carnegie Mellon University. Pittsburgh, USA.

garlan@cs.cmu.edu

ABSTRACT. This paper analyzes Software Architecture, defining it and describing the evolution of this field and its role in software engineering. In addition, it covers key concepts of a software architecture course, steps to pursue an architectural thinking, the elements of organizational architecture maturity and emerging trends and issues such as: Architecture evolution, Architecture conformance, Frameworks, platforms, and ecologies, and Self-Adaptive Systems.
Further we examine how software engineering has matured over the past two decades (and the role that software architecture has played in this process), the requirements of architectural thinking (at both technical and organizational levels), the importance for an organization to have mature architectural practices and the existence of important new trends that are reshaping the way software architecture is practiced.

KEYWORDS: Software Engineering, Challenges of Software Engineering, Software Architecture, Evolution of Software Engineering, Architectural thinking, Organizational architecture maturity, Emerging trends and issues.

Garlan, D. (2014). Software Engineering: Reflections on an Evolving Discipline. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 70-77. Consultado el [dd/mm/aaaa] en www.ijisebc.com

1. Introduction

The big problem in the Software Engineering is 'How to bridge the gap between requirements and solutions?' Software Architecture attempts to address problem by providing a layer of abstraction that supports:

- A high level of system design
- System-level abstractions and qualities
- Design reuse design through common architectural idioms.

Over the past two decades the field of software engineering has made remarkable progress, but many challenges remain.

These significant innovations include:

- Processes for making software development predictable and repeatable
- Better understanding of how to balance technical and business concerns
- Tools to improve the quality of our systems
- Techniques to master the complexity of software systems

But, in this field 'The Challenge' is to:

- Turn Software Architecture into an engineering discipline
  - o from ad hoc definition to codified principles
- Develop systems "architecturally"
  - o build systems compositionally from parts
  - o assure that the system conforms to the architecture and has the desired properties
  - o use standard integration architectures
  - o reuse codified architectural design expertise
  - o reduce costs through product lines

2. Software Architecture: past and present

2.1. What is software architecture?

There are many definitions in the literature: indeed, CMU's Software Engineering Institute's web site on software architecture lists over 100 of them (Software Engineering Institute - Carnegie Mellon University, 2014). One definition that is often used is that the Software Architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them and properties of both.

From an operational point of view issues addressed by Software Architecture include:

- Gross decomposition of a system into parts
  - o often using rich abstractions for component interaction
  - o often using common patterns/styles
- Emergent system properties
  - o performance, throughput, latencies
  - o reliability, security, fault tolerance, evolvability
- Rationale
  - o justifying architectural decisions
- Allowed change
  - o "load-bearing walls"

2.2. Evolution of the field and its role in software engineering

Figure 1 shows a brief summary of the evolution of software engineering.

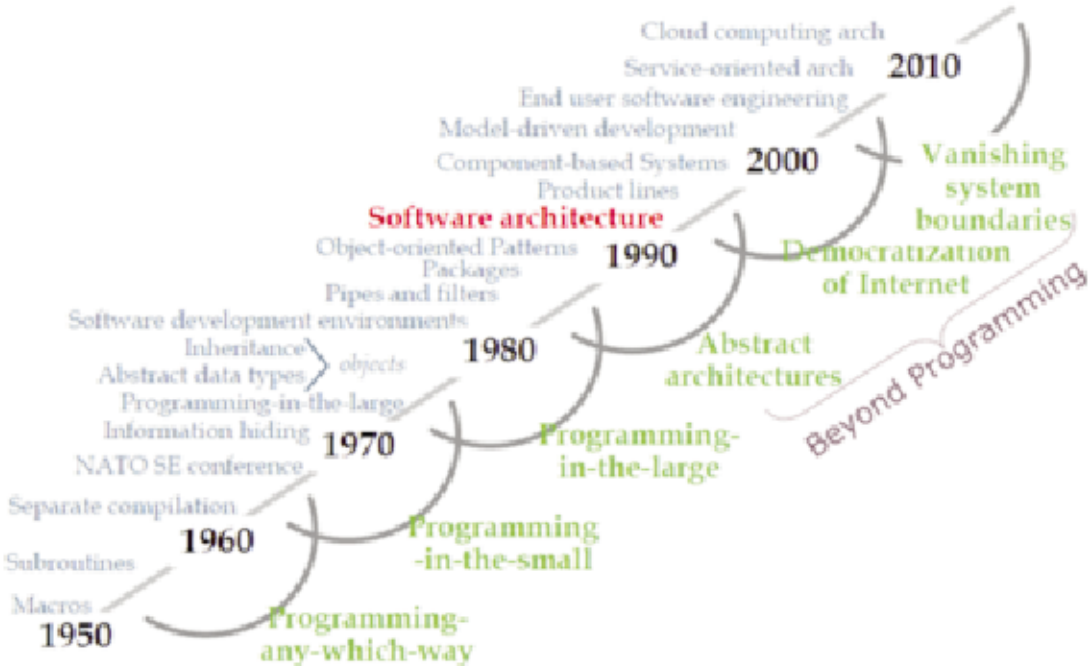

Figure 1. Software Engineering Evolution.

And in this overall evolution of software engineering, Software Architecture has likewise evolved considerably. Some features of this evolution, organized by decade, include:

1980's

- Informal use of box and line diagrams
- Ad hoc application of architectural expertise
- Diverse, uncodified use of architectural patterns and styles
- No identified "architect" on most projects

1990's

- Recognition of the value of architects in software development organizations
- Processes requiring architectural design reviews & explicit architectural documentation
- Use of product lines, commercial architectural standards, component integration frameworks
- Codification of vocabulary, notations & tools for architectural design
- Books/courses on software architecture

2000's

- Incorporation of architectural notions into mainstream design languages and tools (e.g., UML-2)
- Methods based on architectural design and refinement (e.g., Model-Driven Design)
- Some architecture analysis tools

- Architectural standards for Enterprise Systems (e.g., RM-ODP, TOGAF)
- Architectural frameworks (e.g., SOA)

3. What should software engineers know about software architecture?

3.1. Elements of a course on software architecture

Any course on software architecture must contain the following elements:

General Concepts (Definition of software architecture and Basic concepts: views, styles, patterns,…)

Principles of Architecting (Understanding architectural requirements, Architecture styles and tactics, Product lines and integration frameworks, and Techniques to go from architecture to code)

Architecture in Practice [Evaluating architectural designs, Handling architectural problems, Documenting a software architecture, Presenting an architecture to others and Architecture for X (security, usability, reliability, etc.)]

3.2. Architectural thinking

Key to being an effective software architect is a set of mental perspectives and concepts, sometimes referred to as "architectural thinking". These include:

An engineering mindset: Reasoning about qualities such as Scalability, Reliability, Performance, and Cost. In particular, being able to make principled trade-off analysis when it is not possible to achieve all of these simultaneously.

Different issues for architecture & programs

Architecture - (interactions among parts, structural properties, declarative, mostly static, system-level performance, outside module boundary)

Programs – (implementations of parts, computational properties, operational, mostly dynamic, algorithmic performance, inside module boundary)

Styles, Platforms, and Product Lines (Figure 2)

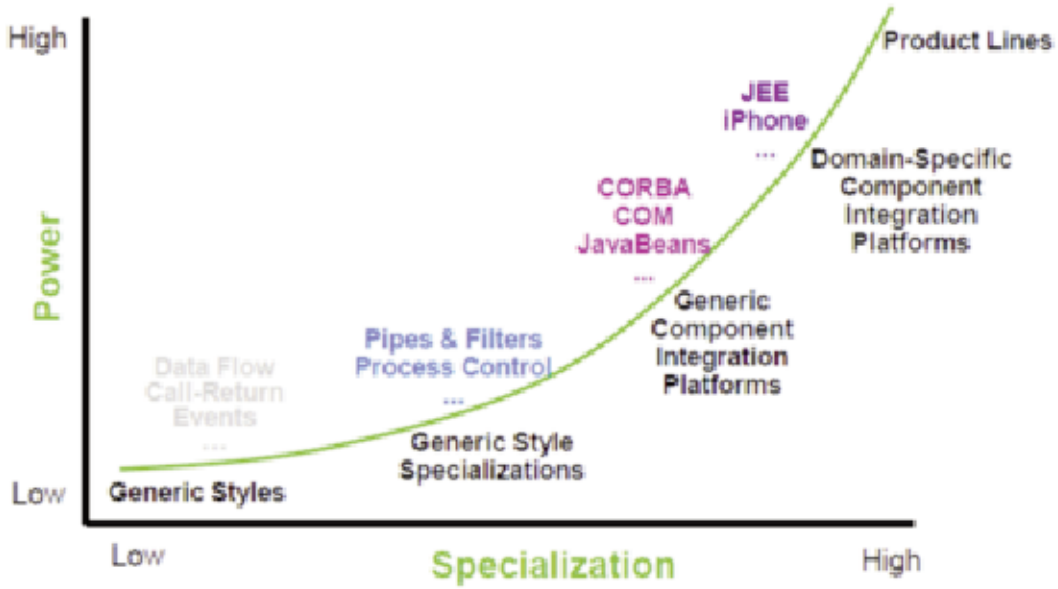

Figure 2. Styles, Platforms, and Product Lines in software architecture.

3.3. Organizational architecture maturity

Organizations differ considerably in their architectural practices. Key features that indicate architectural maturity include:

- Software architects are explicitly recognized and rewarded (Not everyone can become a software architect)
- The company has architecture training (Basic training for all entering software engineers, and advanced courses for designated architects)
- Architecture reviews are a requirement (No project can be approved without a review)
- The company maintains architecture case history, checklists, and guidance documents (Allows corporate knowledge to be maintained)
- The company uses and maintains product lines (Requires both technical and organizational changes)
- Architecture documentation standards are defined and followed (It is not enough to say "We use UML")

4. Emerging trends and issues

4.1. Architecture evolution

Businesses must evolve their architectures from A to C, through a series of incremental architectures. Examples include migrating batch-oriented systems to web-based interactive system; or migrating client-server systems to service-oriented architectures (SOA).

An important issue is how do we approach this problem: Can we leverage past evolution histories? How does this problem link to project planning, cost estimation, work assignments, etc?

4.2. Architecture conformance

We would like to make sure that the implementation conforms to architecture (and vice versa).But the issue

is what does it mean to "conform" and how would we evaluate its satisfaction.

4.3. Frameworks, platforms, and ecologies

We have been building on top of platforms and using software frameworks for most of the history of software engineering. The nature of such platforms has evolved.

But the issue is how to create ecosystems that allow a platform to be sustainable. This one requires a deep understanding of context: economic, legal, organizational, human motivation, user communities,…

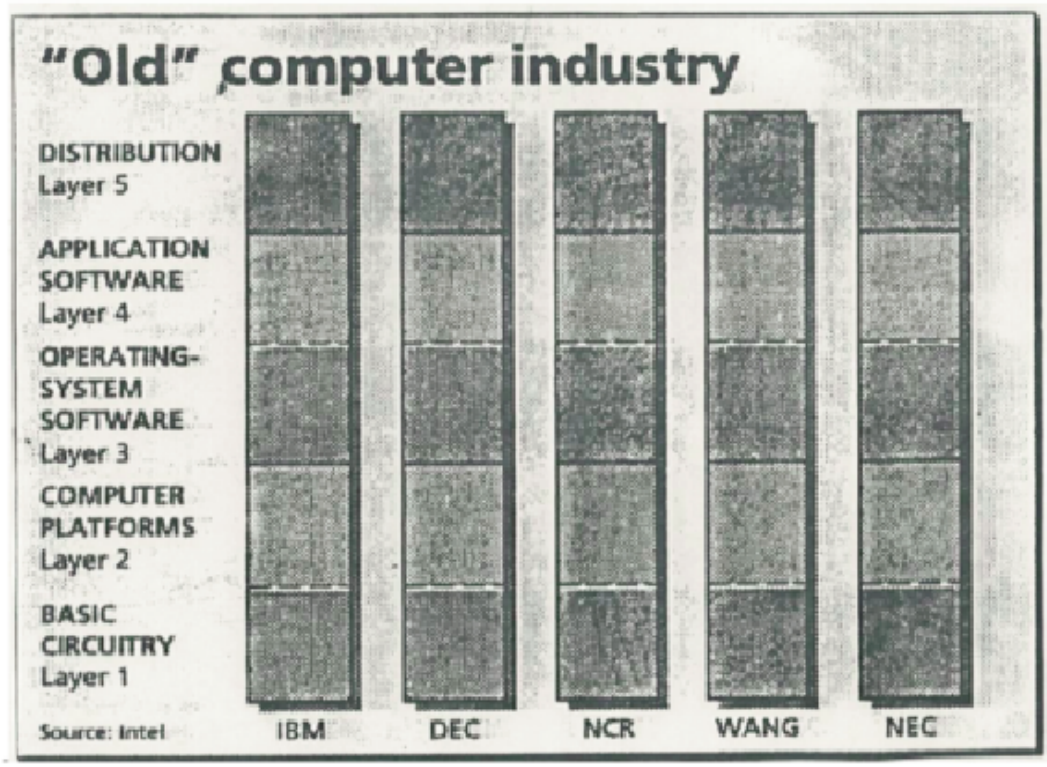

Figure 3. Old structure of the computer industry. Source: Reprinted from The Economist, Feb 27, 1993.

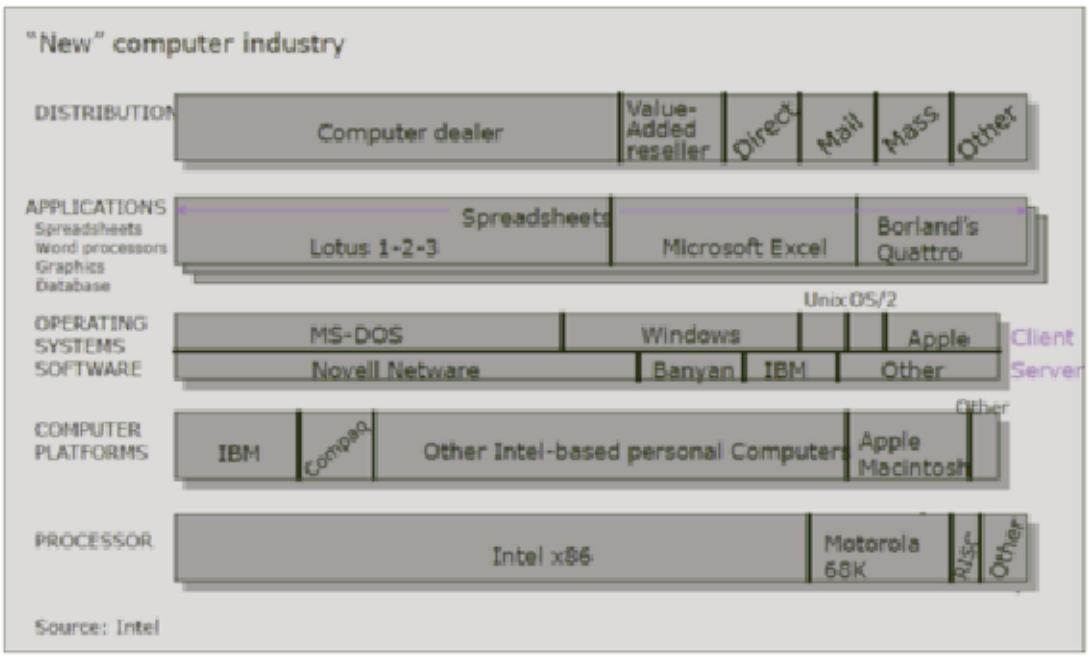

Figure 4. New structure of the computer industry. Source: Reprinted from The Economist, Feb 27, 1993.

Today, there are many new architectural ecosystems:

Architectures enable new ecosystems. Examples: Windows, iPhone, Java EE, Eclipse, Robotics OS, VISTa, SOA

Architectures introduce new roles into the ecosystem. Examples: governance bodies, platform developers and maintainers

Failure to understand how architectures and ecosystems interact can lead to failure. Example: JavaPhone

4.4. Self-Adaptive Systems

Systems must have high availability today and the use of human operators is expensive and error-prone. Hence, systems must automatically adapt to: failures, changing environmental conditions, changing requirements and threats. Examples of adaptation operations: server reboot, reduce fidelity to accommodate high load, and block an intruder. But the challenge is how to control this.

One approach to this problem is exemplified by the Rainbow System. Key features of that system are:

- Uses system models at run time as a basis for self-healing (monitoring; problem detection; repair)
- Supports new reasoning techniques for self-adaptive systems (determining "best" repair strategy and analyzing soundness of repairs)
- Can be used to achieve self-securing systems (both reactive and proactive).

5. Conclusions

In conclusion, the principal lessons to take away are:

- Software engineering has matured over the past two decades
- Software architecture has played a major role in this process
- We now understand what it means for an organization to have mature architectural practices
- This requires "architectural thinking" at both technical and organizational levels
- There are important new trends that are reshaping the way software architecture is practiced

Indeed, Software Architecture is a science that today that has substantial practical impact, but remains the focus of much research andcontinues to evolve as technology drives the field.

Cómo citar este artículo / How to cite this paper

Garlan, D. (2014). Software Engineering: Reflections on an Evolving Discipline. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 70-77. Consultado el [dd/mm/aaaa] en www.ijisebc.com

References

The Economist (1993, Feb 27). Structure of the computer industry. The Economist.

Software Engineering Institute - Carnegie Mellon University (2014). Retrieved November 15, 2014, from http://www.sei.cmu.edu/