Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Future Trends of Software Technology and Applications:

Software Architecture

Paul Clements
Software Engineering Institute
Carnegie Mellon University

Sponsored by the U.S. Department of Defense

Software Engineering Institute

Applied R&D laboratory situated as a college-level unit at Carnegie Mellon University, Pittsburgh, PA, USA

Established in 1984

Technical staff of 335

Offices in Pittsburgh, Pennsylvania, Arlington, Virginia, and Frankfurt, Germany

Purpose: Help others make measured improvements in their software engineering practices

The ascendance of software architecture

The software architecture of a program or software system is the structure or structures of the system, which comprise elements, the externally visible properties of those elements, and the relationships among them.
--- Software Architecture in Practice, 2nd ed., 2003

Software architecture as a field of study has enjoyed a steady ascendance since the 1980s (although its foundational principles go back much farther: e.g., Parnas on Information Hiding ,1972).

- Dedicated conferences: WICSA
- About 20-30 CFPs mentioning it are “active” at any particular time
- *Journal of Systems and Software* (Elsevier) section

The ascendance of software architecture

In practice, software architecture is has been embraced organizationally.

- Well defined practitioner roles and career paths
- Well defined architecture-based development methods
- Codification of duties, skills, and knowledge

A recent search revealed

- 29 university courses with on-line descriptions *
- 22 industrial courses
- 7 certificate/certification programs
- 75 books

* http://www.sei.cmu.edu/architecture/educators.html

The ascendance of software architecture

Architecture is seen as essential for the construction of large, complex software systems

- It is the primary carrier of a system's quality attributes
- It structures the development project as well as the software being developed
- It is a capital investment that can be reused, powering (for example) software product lines
- It is an essential communication vehicle among the system's stakeholders

Architecture’s role to date

In order to predict its future, it helps to understand architecture’s role up to this point.

- When programs became too large and complex to understand (or to engineer) by working only with source code, architecture became indispensable to exert intellectual control over the software.
- It helped us think in terms of problem-domain abstractions unsupported by programming languages of the day.
- Architecture provided large scale conceptual “chunks” of functionality wired together as a solution, chunks that addressed particular parts of the problem.
- Generally useful or commonly recurring chunks led to pre-packaging and pre-wiring of those chunks, which in turn led to languages in which those chunks became primitives.

Examples of “chunks”

Architectures and architectural thinking, driven by principles of modular design, gave rise to
- Clients and servers
- Middleware
- Services
- Whole-enterprise IT solutions

Over time, the “chunks” have become simultaneously more specialized, more general, and more complex.
- 1960s chunk: SQRT subroutine
- Intermediate chunk: database, transaction processing
- 2000s chunk: shopping cart, on-line auction, B2B solutions.

Here’s an idea

Programs are *very* complex and they require programmers to think too much in terms of the underlying computational platform, and not the problem space.

Let’s give programmers a way to express themselves in the language of the problem space.

Let’s introduce software to automatically translate from platform-independent expressions into platform-dependent solutions.

Wouldn’t that be great?

We could call this concept…

FORTRAN.

John Backus

"FORTRAN was designed for mathematicians and scientists, and remains the preeminent programming language in these areas today. It allows people to work with their computers without having to understand how the machines actually work, and without having to learn the machine's assembly language."

-- The History of Computing Project
http://www.thocp.net/biographies/backus_john.htm

FORTRAN

FORTRAN was not thought of by its creators as a programming language.

It was designed to be the input to a translator.

People initially threw out their FORTRAN code and pored over the assembly language generated from it.

On the day people stopped looking at the assembly language and started treating the FORTRAN code itself as the artifact of value, FORTRAN became a programming language.

So what?

Axioms:

1. There is nothing inherently different between a specification language and a programming language.

2. There is no stark dividing line between problem space and solution space. Rather, it is a continuum.

So what?

Model-driven architecture is remarkable (to me) in that its descriptions seem to have very little to do with architecture.

I prefer the term “model driven development.”

The fundamental concept is in the translation from models to solutions.

So if this important new paradigm can sidestep architecture, can software architecture be all that important?

So what?

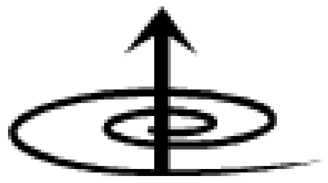
In some IT communities, software architecture seems to be de-emphasized today.

The “chunks” are so large and sophisticated, and go together in pre-wired ways. Creating the software architecture is not a complex task.

Some IT firms say their architectural decisions consist of choosing between two competing vendors’ solutions. Once they do that, everything else follows.

Can software architecture be all that important?

Software architecture's role in software engineering



The history of software engineering can be viewed as a steadily increasing spiral of language expressiveness, into problem spaces and away from platform details.

When a new way to express problems arrives on the scene, programs are initially very simple – their structure is very simple.

But we always learn to express more and more complex solutions using the languages we have.

Final word

As the solutions written using these new mega-programming languages become more and more complex, architecture will be there to help "solution architects" (the current term for those who program in these super-languages) to

- impose order on the chaos
- make sure quality attributes are achieved
- to structure the solution being given to the underlying computing systems.

We will see continued movement towards supporting larger, more complex, and more varied problem-space abstractions, backed up by more sophisticated translation or compiler technology that will allow today's "specification languages" or "modeling languages" to continue to be tomorrow's programming languages.

Specific predictions

Architecture-based practices (e.g., formal evaluations, standardized documentation) will continue to be codified.

Tooling to support architecture design and development will improve greatly.

We'll see true "round trip engineering" in which the current gulf between architectural design and downstream design and implementation languages and representation is bridged.

Carnegie Mellon
Software Engineering Institute

Questions—Now or Later

Paul Clements
Product Line Systems Program

Email: clements@sei.cmu.edu

U.S. Mail:
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

World Wide Web:
http://www.sei.cmu.edu/
architecture

SEI Fax: 412-268-5758