Paradigmatic Software Development

Introduction

Paradigms

What they are and why it is important to understand them.

Paradigm

- A philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated.
  - Merriam-Webster Online Dictionary

- A *methodology* is a specific implementation of a paradigm.

Paradigms – What

- A paradigm is a model or pattern, an archetype, for solving problems. Inherent in a paradigm are basic assumptions about problems. Consequently a paradigm defines or structures how problems are approached.
- Paradigms constrain the way that we think about problems, and guide the way problems are solved.
- A paradigm provides a broad philosophy to be adopted during the development process. It describes the general approach to be used, rather than the specific tools and methods.

Paradigm Examples

- Computational Paradigms (Models of Computation)
  - Function oriented
  - State oriented
- Programming Paradigms
  - Procedural
  - Object Oriented
  - Logic
  - Functional
- Development and Process Paradigms
  - Waterfall Model
  - Iterative/Incremental Development
  - Agile Processes
  - The Quality Paradigm

Paradigms – Why

Distinguishing between paradigms is valuable for solving problems, for several reasons:

- Understanding a paradigm involves identifying the important entities and strategies in problem solving.
- Knowing which paradigm is being applied reveals assumptions being made in modeling a problem.
- Understanding paradigms helps the problem solver avoid biases.
- Being able to state the advantages and drawbacks of various paradigms permits a decision between paradigms based on the problem under consideration.

Software Development

The *process* of developing
software *products*

Software Development

- Cost estimation, budgeting, scheduling.
- Requirements analysis.
- Design.
- Implementation.
- Testing.
- Deployment.
- Measuring products and processes.
- Quality improvement.
- Maintenance.
- And so on …

Product Viewpoint

- What functionality is provided?
- What performance characteristics are required for the functions provided?
- What is the user interface?
- What are the intended uses of the product?
- What are the customer's expectations regarding the product?
- How will the product be maintained over time?

Product Viewpoint (2)

- Does the system use resources wisely?
- How many defects are present in the product?
- How serious are the defects?
- What quality attributes are important for the product?
- How will the important quality attributes be assessed?

Process Viewpoint

- Who is involved? What roles do individuals have?
- How many individuals are involved?
- What are their individual abilities and characteristics?
- What are the tasks they perform? How are tasks allocated to individuals?
- Is there a match between individual's characteristics and their assigned tasks?

Process Viewpoint (2)

- How do you know when a task is complete?
- How long does each task take?
- Are there dependencies such that some tasks must be complete before other tasks are begun?
- Can some tasks be performed concurrently?
- How is the process coordinated, managed, and controlled?
- What communications support is required?
- What work products does the process produce?

History of Software Development

History – Famous Predictions

- "I think there is a world market for maybe five computers."
  - Thomas Watson, chairman of IBM, 1943
- "There is no reason anyone would want a computer in their home."
  - Ken Olson, president, chairman and founder of Digital Equipment Corp., 1977

History – Computing Practice (1)

Era	Problems	Style
Early Years	Small batch programs Custom software Limited distribution	Seat-of-the-pants Developer is the user
mid 1960s to late 1970s	Algorithms Real-time, multi-user Databases Product software	Software houses Programming in the small
mid-1970's to late 1980s	User interfaces Distributed Systems Management Systems Consumer software	Programming in the large
mid-1980's to 2000	OO Systems Parallel Computing Desktop systems	Mega-programming
2000 to ???	Multimedia systems Web-based IT systems	

History – Computing Practice (2)

Era	Data Issues	Control Issues
Early Years	Basic data representation	Elementary understanding
mid 1960s to late 1970s	Data structures and data types	Programs execute once and terminate
mid-1970's to late 1980s	Databases	Programs execute continuously
mid-1980's to 2000	Multi-media databases	Concurrency and distribution
2000 to ???	Streaming multi-media	

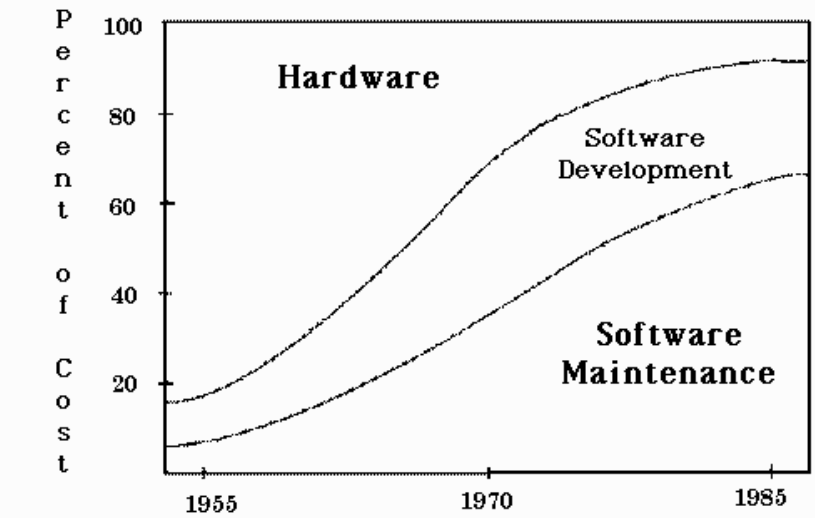
History – Computing Practice (3)

Era	Specification Issues	State Space
Early Years	Mnemonics Natural language prose	Not well understood
mid 1960s to late 1970s	Simple I/O specifications	Small, simple state space
mid-1970's to late 1980s	Complex specifications	Large structured state space
mid-1980's to 2000	Safety critical systems	Enormous
2000 to ???		

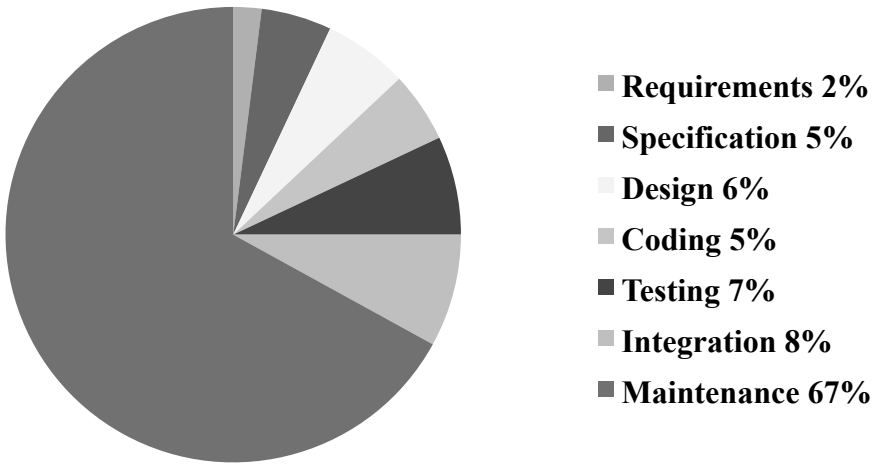
History – Computing Practice (4)

Era	Management Focus	Tools and Methodologies
Early Years	None	Assemblers Core dumps
mid 1960s to late 1970s	Individual effort	Compilers Debuggers
mid-1970's to late 1980s	Team effort	Integrated development environments, Structured methodologies
mid-1980's to 2000	Process quality improvement	OO Methodologies Code reuse
2000 to ???		Design patterns Component based design

History – Cost Trends

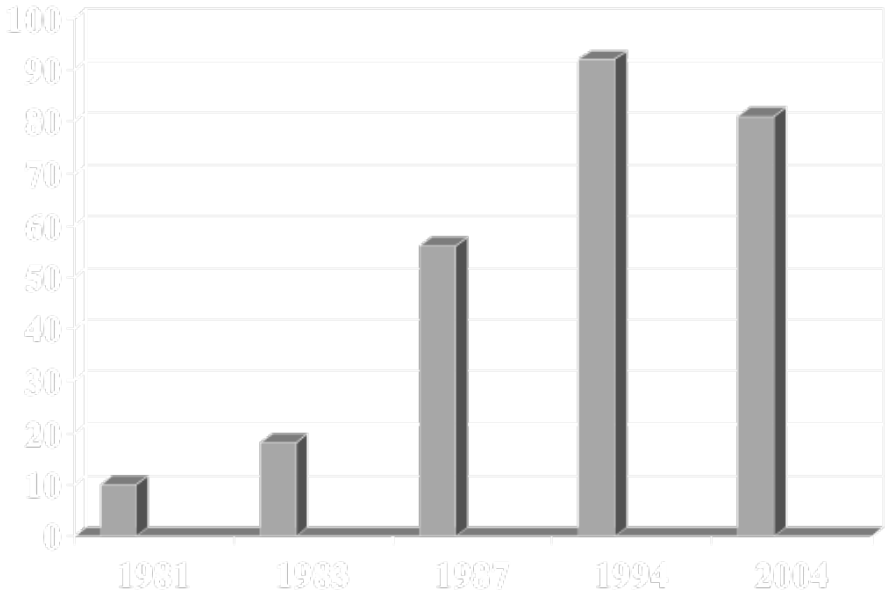


Relative Costs of SW Lifecycle



History – Market Size

Worldwide software market in billions of dollars



Current Process Concerns

- Programming in the large
  - Multi-person teams work for several years to specify and implement a system.
  - Collaboration and communication may be required between individuals who are separated by space and time.
  - Systems may comprise several programs, possibly written in different languages and running on different platforms.
- The efficiency of the process is as important as the efficiency of the product.
- Solving the right problem!

Current Product Concerns

- Hardware continues to advance faster than our ability to write software to take advantage of it.
- Demand for software outpaces production.
- Existing software is difficult to maintain.
- Industry continues to produce poor quality software.
- All segments of society are increasingly dependent on software.

Software Product Liability

Suppose software was subject to the same kinds of product liability lawsuits as other products:

- What would be the impact on software quality?
- What would be the impact on software innovation?
- Would consumers benefit from the change?
- How would the change affect the profitability of software companies?
- How would the change affect the job market for software developers?
- What other issues should be considered?
- What are your conclusions? Should software be subject to product liability? If so, should there be limits?

The Nature of Software

Software Complexity

- Software is more complicated than other artifacts constructed by human beings.
- The number of possible states for a software product is enormous. Additionally, the parts of the product can interact.
- Hardware pales in comparison.

Software Conformity

- Software is not usually isolated in the world.
- Software is typically an integral part of more intricate systems. Software, therefore, must interface with existing systems and conform to the way existing systems behave.
- Since software is not physical it is seen as the most flexible.
- If existing hardware and software are incompatible, nobody ever suggests modifying the hardware to conform to the software!

Software Changeability

- There will always be pressure to change software.
- Software is intended to contribute to people's ability to cope and manage their worlds, and their worlds change.
- When the requirements for a software system are defined, they are based largely on the nature of the world before the system is built. When the new system is introduced, the world is immediately changed.

Software Invisibility

- We have yet to achieve an adequate representation for systems.
- We certainly have various graphical and textual representations for systems, but we do not as yet have any way to capture an overview of the product that allows us to reason or communicate about the system.

Software Variety and Novelty

- Various software products often have very little in common. What are the similarities between a word processor, a web server, and an air traffic control system?
- Software companies are constantly asked to create novel products unlike anything seen before.
- Compare this situation to the automobile industry which gradually refines what is essentially the same product over many years.

Software Testing

- On average, professional coders make 100 to 150 errors in every thousand lines of code.
- In almost any phase of tests less than half of the defects are found and fixing defects in a complex product can introduce new defects.
- If a bridge survives a 500-kilogram weight and a 50,000-kilogram weight it is safe to assume that it will bear all the values in between. With software, there is no way to interpolate.

Factors Influencing Difficulty

- Number of functions performed
- Novelty of application
- Concurrency
- Multitasking
- Real-time constraints
- Parallel and distributed processing
- Amount and structure of data
- Criticality
- Security
- Interaction with other systems
- Hardware constraints
- Stability of specification
- User sophistication

The 4 Ps of Software Development

People

Problem

Process

Product