



l e a n
software development

Expanding Agile Horizons

The Five Dimensions of Systems

Plank Road Fever



USA: Late 1840's – mid 1850's

Massive boom in plank road construction

- ✓ High capital investment
 - ✗ Numerous large and small investors
- ✓ To be paid for with tolls

Immediate, positive results

- ✓ Far superior to muddy, rutted roads
- ✓ Dramatic decrease in travel time
- ✓ Expanded rural markets

BUT

- ✓ Roads deteriorated in 4 years
 - ✗ Half the projected lifespan
- ✓ Maintenance costs were very high
 - ✗ Annual costs were 20-30% of initial cost
- ✓ Most plank roads were soon abandoned

Information Cascade

“The first plank roads were a huge success. People looking for a solution to the road problem found one ready-made at hand. As more people built plank roads, their legitimacy became more entrenched and the desire to consider alternate solutions shrank. It was years before the fundamental weakness of plank roads – they didn't last long enough – became obvious.”

James Surowiecki, *The Wisdom of Crowds*

Is Agile a Plank Road?



A History of Plank Roads

1968: NATO Software Engineering Conference *The Software Crisis*

As long as there were no machines, programming was no problem at all;
When we had a few weak computers, programming became a mild problem;
Now we have gigantic computers, programming had become an equally gigantic problem.

– Edsger W. Dijkstra: ACM Turing Award Lecture, 1972.

[Root cause: Wishful thinking] The most deadly thing in software is the concept, which almost universally seems to be followed, that you are going to specify what you are going to do, and then do it. And that is where most of our troubles come from. – D.T. Ross, MIT

My main worry is in fact that somebody in a position of power will recognize this crisis and believe someone who claims to have a breakthrough, an easy solution. The problem will take a lot of hard work to solve. There is no worse word than ‘breakthrough’. – Ross

The Plank Road

High level languages: Cobol; Fortran; Algol; PL/I

BUT: High level languages removed drudgery from programming, making the job more complex and requiring higher caliber people! – Dijkstra



A History of Plank Roads

1972: New York Times Information Bank

Structured Programming

Edsger Dijkstra: [Do not separate design from implementation.] Testing is a very inefficient way of convincing oneself of the correctness of a program.... The quality of the product can never be established afterwards. Whether the correctness of a piece of software can be guaranteed or not depends greatly on the structure.

Dave Parnas: [Information hiding.] Divide program into modules based on their responsibility and the likeliness of future change, not on order of flow.

Top Down Programming

Terry Baker: An evolutionary approach to systems development....integration is completed parallel with, rather than after, unit coding....As a result, acceptance testing and subsequent system testing have been nearly error free.

The seeds of: Designed-in Quality; Information Hiding; Continuous Integration; Technical Leadership



A History of Plank Roads

1972: New York Times Information Bank

Chief Programmer Team

1. *Lead & Backup pair – responsible for designing and programming the system.*
2. *Both deeply involved in design and programming. Review each other's work.*
3. *Lead programmer supervises other programmers and reviews their work.*
4. *Backup capable of taking over for lead. Continually tests the system.*
5. *Library – repository for all code. Common code ownership.*

Results of New York Times Project

One detected fault and 10,000 LOC per person-year (83,000 LOC)

21 faults in acceptance testing; 25 further faults in the first year of operation

BUT

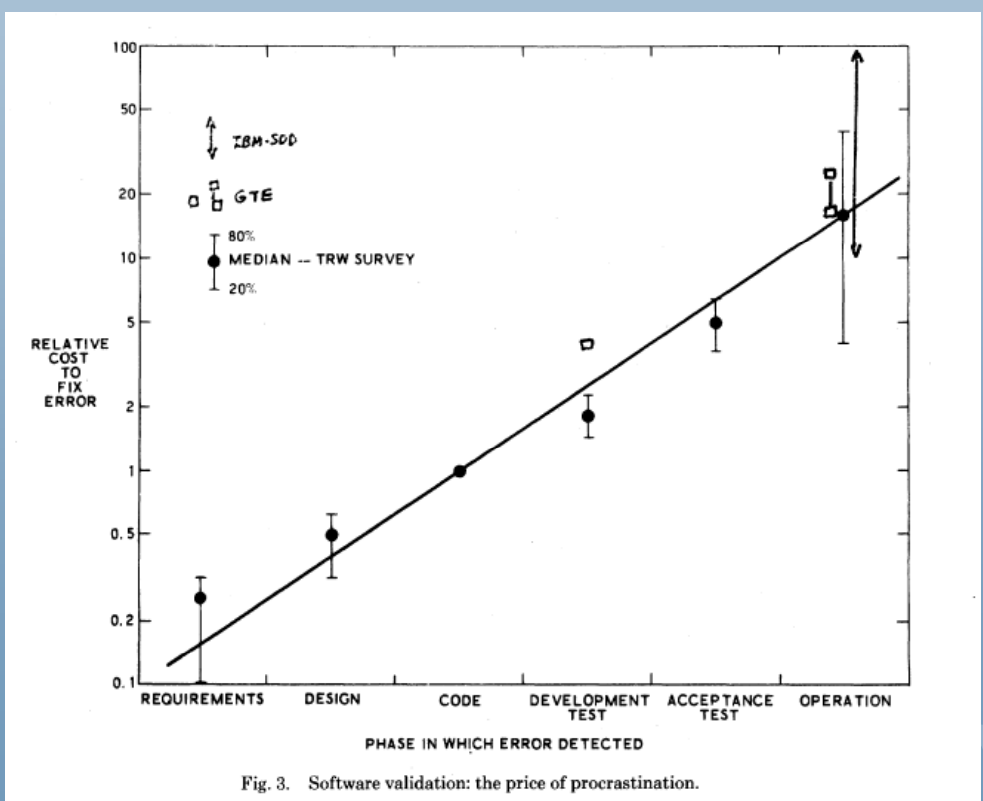
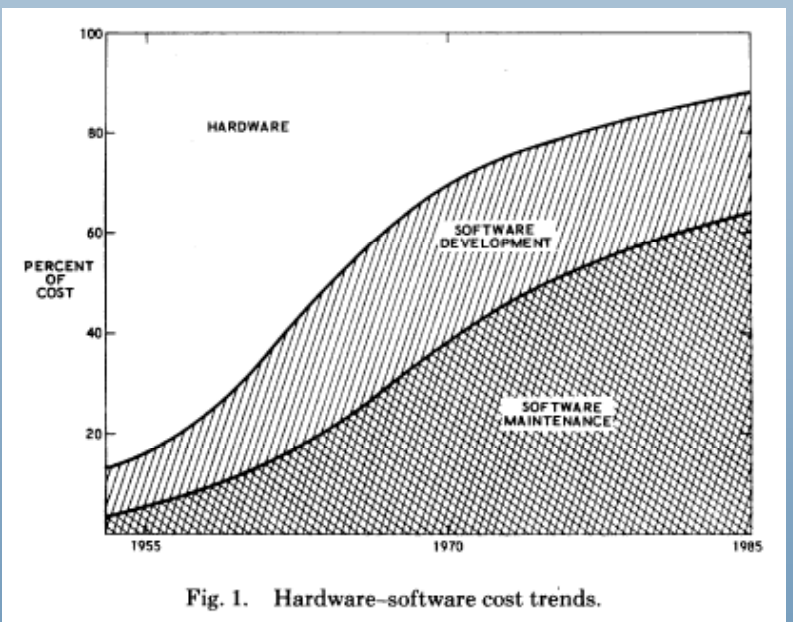
Because we cannot depend on the existence of a super-programmer in any project and because we must transfer our people around in order to apply the right number of resources at the right time to a contract, we have found that it is important to build our systems without dependence upon any particularly strong individual.

– J.D. Aron: 1969 NATO Software Engineering Conference, Rome



A History of Plank Roads

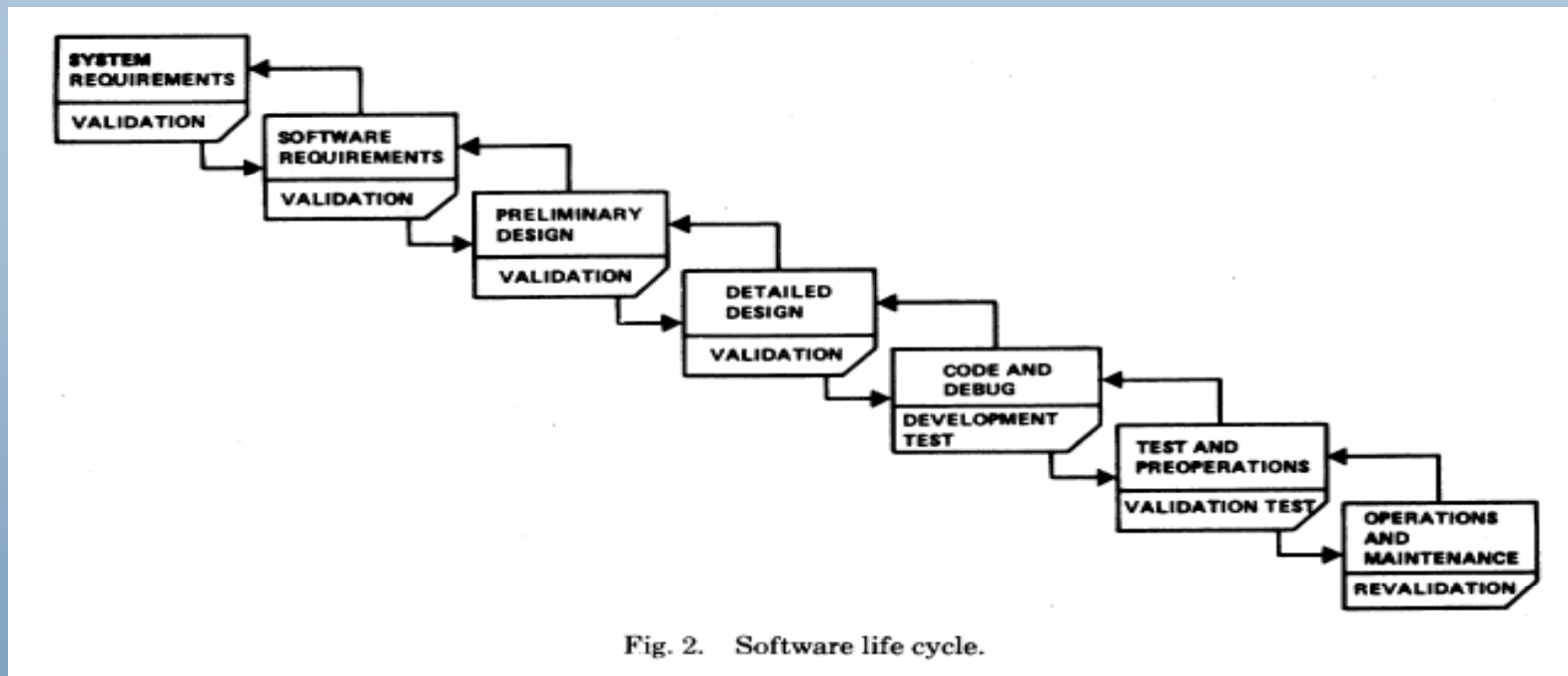
1976: Software Engineering – Barry Boehm *The Problem*





A History of Plank Roads

The Plank Road: Software Life Cycle



BUT

This was aimed at large government systems with unchanging requirements. It was widely applied to business systems with constantly changing requirements.



A History of Plank Roads

1982: Life Cycle Concept Considered Harmful

Daniel McCracken & Michael Jackson

– ACM Software Engineering Notes, April 1982

1. Any form of life cycle is a project management structure imposed on system development. To contend that any life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous.

The elaborate life cycle assumed as the basis for this conference may have seemed to be the only possible approach in the past when managing huge projects with inadequate development tools. (That it seemed to be the only choice, obviously did not prevent many such projects from failing.)

What we understand to be the conventional life cycle approach might be compared with a supermarket at which the customer is forced to provide a complete order to a stock clerk at the door to the store, with no opportunity to roam the aisles--comparing prices, remembering items not on the shopping list, or getting a headache and deciding to go out for dinner. Such restricted shopping is certainly possible and sometimes desirable--it's called mail order--but why should anyone wish to impose that restricted structure on all shopping?

2. The life cycle concept perpetuates our failure so far, as an industry, to build an effective bridge across the communication gap between end-user and systems analyst. In many ways it constrains future thinking to fit the mold created in response to failures of the past. It ignores such factors as the following, all of which are receiving rapidly increasing attention from both researchers and practitioners:

- Heavy end-user involvement in all phases of the application development process--not just requirements specification, but design and implementation also.
- An increasing awareness that systems requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't know them in advance--not even in principle. To assert otherwise is to ignore the fact that the development process itself changes the user's perceptions of what is possible, increases his or her insights into the applications environment, and indeed often changes that environment itself. We suggest an analogy with the Heisenberg Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose. System development methodology must take into account that the user, and his or her needs and environment, change during the process.

3. The life cycle concept rigidifies thinking, and thus serves as poorly as possible the demand that systems be responsive to change. We all know that systems and their requirements inevitably change over time.



A History of Plank Roads

1980: Relational Databases / Query Languages

1. *Edgar F. (Ted) Codd – inventor of relational DB's – wins ACM Turing Award.*
2. *INGRES – an open source relational database – is commercialized.*
3. *SQL introduced by IBM.*

1982: Application Development Without Programmers

– James Martin

The Plank Road: 4th Generation Languages

1. *End users can access and manipulate database information without DP help.*
2. *Applications can be generated directly from business requirements.*
3. *End users can use easy these languages: BASIC, APL, NOMAD, SQL....*

BUT

1. *Inadequate tools*
2. *Un-interested end-users*
3. *Oversimplification of essential complexity*





A History of Plank Roads

1984: Timebox pioneered by Scott Shultz, DuPont

Information Engineering Associates (IEA)

~ 30 days for general analysis and design
90 day timebox to develop the application.

Rapid Iterative Production Prototyping (RIPP):

Applications Factory: fourth-generation language,
screen painters, a database management system, etc.

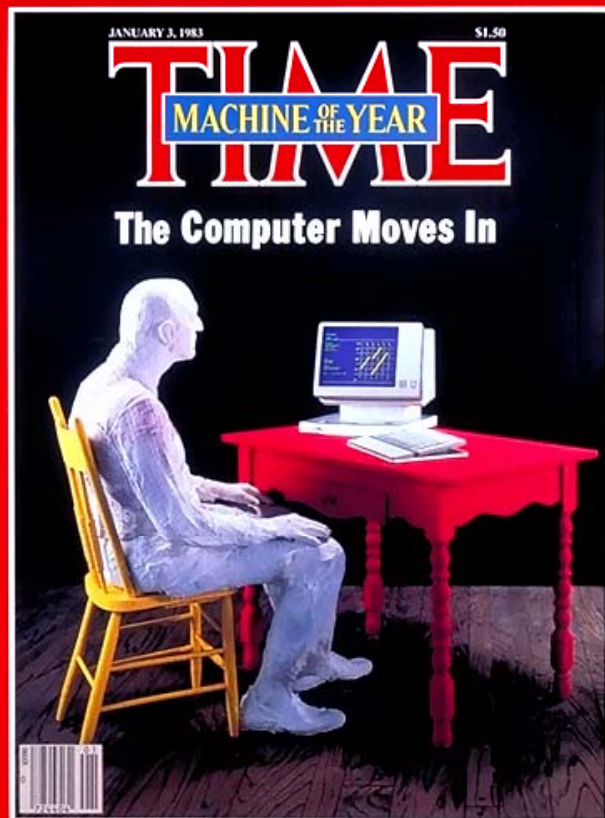
1988 – formed service unit to sell development

Lasted ~2 years.



Interlude

Meanwhile, while no one was paying attention....



1977: Apple II

1978: WordStar

1979: VisiCalc

1980: MTP (=>SMTP)

1981: IBM PC

1982: Lotus 1-2-3

1983: TCP/IP standardized

1984: Post Office Protocol (POP)

1985: PageMaker

1986: LISTSERV

1987: Wintel

1988: First Internet virus

1989: HTTP =>WWW

1990: Internet goes commercial

Lifecycle Revisited

1988

Spiral Model
– Barry Boehm
IEEE Software, September, 1988

Full elaboration of requirements may be delayed.

BUT

This is a project management model, not a systems engineering model.

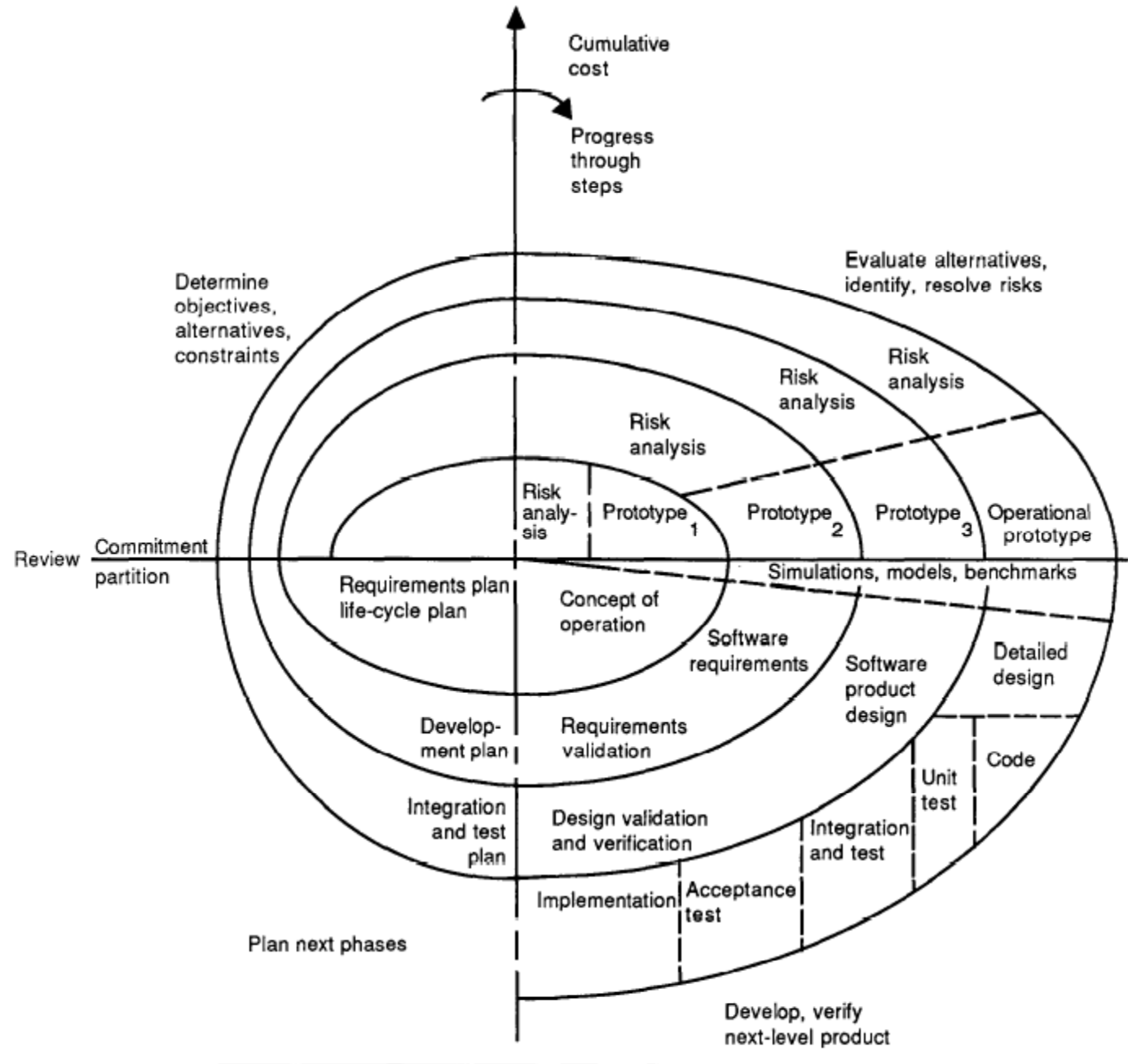


Figure 2. Spiral model of the software process.



A History of Plank Roads

1988: Software Process Maturity Framework

Watts S. Humphrey
Software Engineering Institute

The Problem:

The CMM was designed to help guide improvements in software organizations which miss schedules, overrun budgets, and deliver defective software.

The Plank Road:

Bring Software Development under Statistical Process Control.
Mandate Maturity Assessments.

BUT

“High Maturity” is not the same as good engineering. It focuses on project management practices rather than systems engineering practices.

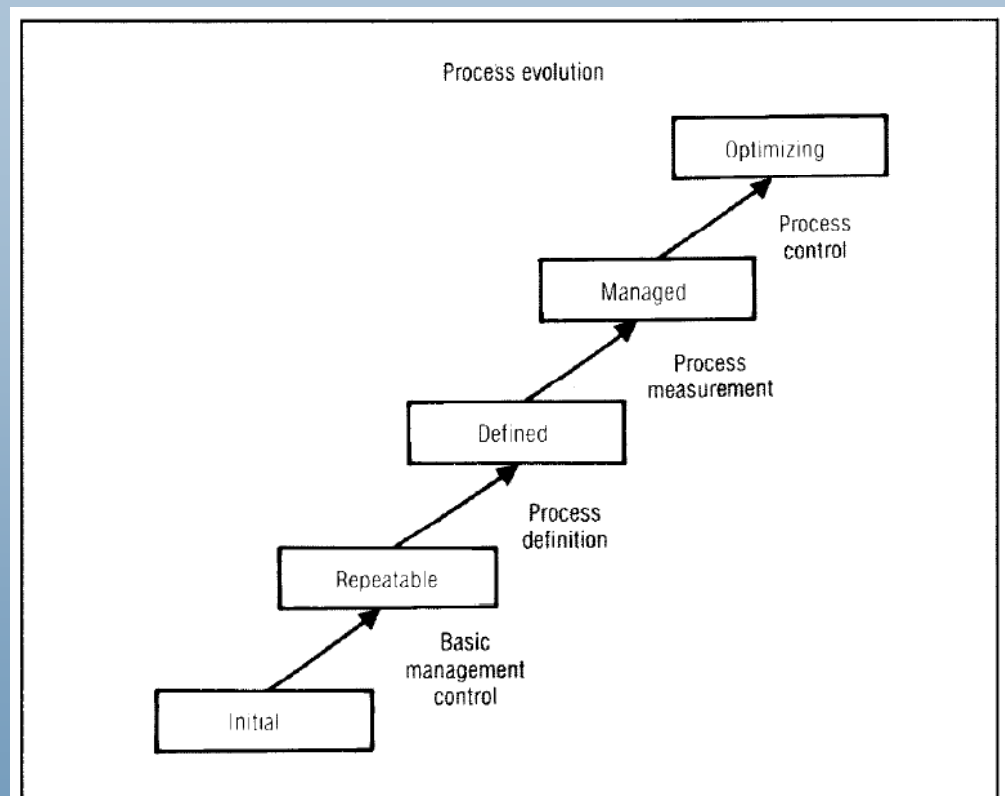


Figure 1. The five levels of process maturity.



A History of Plank Roads

1991: Rapid Application Development (RAD)

– James Martin

The Plank Road: CASE Tools

- 1. Facilitated JAD workshops to engage users in design*
- 2. Incremental [timeboxed] development*
- 3. CASE tools for automatic code generation*
- 4. Small teams of highly skilled and motivated people*

BUT

- 1. JAD sessions were not enough for robust design*
- 2. CASE tools were inadequate for complex problems*
- 3. Teams were not necessarily highly skilled and motivated*
- 4. RAD implementations often produced un-maintainable code*



Does RAD Live Up to the Hype?



Then along comes 1995

Internet

That's not software!



*Completely new system paradigm.
Dramatically different business model.*



Y2K

Black hole.



*No time to pay attention.
People lost to the gold rush.*

The Perfect Storm



What Worked

Internet:

Technical Vision: J.C.R. Licklider

Robustness: Distributed, independent agents

Standards: Emerged through broad-based discussion

Browser: Designed to accommodate constant change

PC's:

Separable Architecture: Small, focused programs

Staged Deployment: Early release, regular updates

Both:

Experimentation: Try lots of stuff and keep what works



Learning from Our History

Let us not confuse:

Systems Engineering

Designed-in Quality

Information Hiding

Continuous Integration

Respect for Complexity

Skilled Technical Leaders

Learning Cycles

No separation of design from implementation.

Project Management

Lifecycle

Requirements

Stages of Testing

Maturity Level

Managed Processes

Timeboxes

Cost, Schedule, and Scope determine success



The Pittsburgh Airport

September 24, 1992 – Tour of Pittsburgh Midfield Terminal

County Engineer – Long ago planned to retire on October 1st

- ✓ The day the terminal was scheduled to open.
- ✓ Completely smooth overnight transfer of operations.
- ✓ Ranked as the best airport in the US by CodeNest.

Showed of many innovative features of his “baby”

- ✓ People-friendly escalators, shops with outside prices
- ✓ No strikes, excellent safety record, on time completion
- ✓ Baggage handling system with manual back-up

Predicted Denver airport fiasco and assigned responsibility

- ✓ “An engineer that does not plan a manual backup for an automatic baggage handling system is asking for trouble.”



Five Dimensions of Systems

Engineering:

The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property.

American Engineers' Council for Professional Development

1. Purpose
2. Structure
3. Integrity
4. Life Span
5. Results



Purpose

- Are technically skilled people deeply in touch with the overall objectives of the system?
- Do technical team members “Go and See” what the real problem is?
- Is there a technical leader who is responsible for achieving the overall system purpose, and who provides technical guidance as necessary?
- Are there built-in learning cycles to discover, verify, and improve suitability for purpose?



Structure

- Does the team have a good overall idea of what they are producing, it's organization, and how their work fits into the overall system?
- Are there learning cycles that engage designers implementation and feedback from actual use?
- Do junior people have the training and oversight to assure that they produce well-factored code which incorporates learning from earlier work?
- Is work reviewed by an appropriate reviewer?



Integrity

- Is the integrity of the system built into the design or is it “tested-in” at the end of development?
- Is the system developed such that minimal time is devoted to any final merge and testing?
- Are failures rare and investigated deeply to discover design patterns that can be used to prevent similar failure modes in the future?
- Is such learning captured, consolidated, and disseminated in an effective, actionable manner?



Life Span

- Is the lifespan and future changes to the system a key consideration in its design and development?
- Are the people who will operate, maintain, and support the system deeply involved in its design?
- Does the development team remain involved with the system over the long term?
- Are rewards structured so that developing a robust system that performs well over time is the most strongly encouraged behavior?



Results

- How well does the system achieve its purpose; how easily and quickly does each stakeholder derive anticipated value from the system?
- Does the system meet its overall economic goals within its valid cost and schedule constraints?
- Does the system perform without failure and is there confidence that it will continue to do so?
- Does the system delight those who operate, use, support, maintain, and derive value from it?



l e a n

software development

Thank You!

More Information: www.poppendieck.com