Rational. software

Staying Rational (In an Irrational World)

Grady Booch
IBM Fellow and Free Radical

IBM Rational Software Development Conference UK 2007

What keeps me **Rational**?

IBM®

How We Got Here

1910s beginning of automation

1920s beginning of expansion

1930s beginning of dependence

1940s beginning of von Neumann machines

1950s rise of the machines

1960s rise of the languages and methods

1970s death of the mainframe

1980s age of the personal computer

1990s age of the Internet and new methods

2000s retrenchment

Where Are We Going

2010s age of transparency

2020s total dependence

2030s rise of the machines

The State of the World - 2031

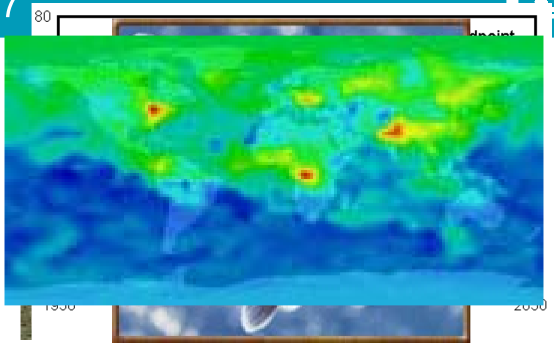
- Population
- Resources
- Politics & Society
- Warfare
- Agriculture
- Business
- Manufacturing
- Transportation
- Consumers
- Entertainment
- Medicine
- Science and technology

www.longbets.org
www.wfs.org
www.chronicle-future.co.uk
www.gwforecast.gwu.edu

Population

- Global population nearing its peak of 8.8 billion
- Global decline in fertility rate
- Population decline in developing nations
- Mortality decline in developing nations
- Continuing shift of population to 100 mile cities -

Resources



- Significant % of the world’s population chronically short of fresh water
- Global oil production in decline (Hubbert Peak)
- Some fisheries have collapsed, some have been saved
- Air pollution plagues a number of cities -

Politics & Society

- Entrenchment of the EU and other trading blocks
- Online representative government common
- Web continues to penetrate national boundaries
- Biometrics commonly used to track the moment of individuals
- New kinds of crime emerging
- Information dark age continues -

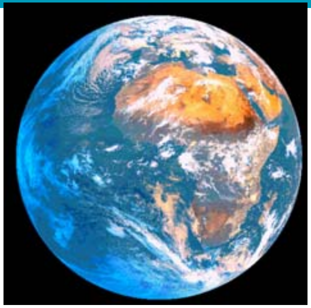
Warfare

- Continuing battle against stateless combatants
- Terrorism and the proliferation of WMD
- Electronic battlefield
- Remotely controlled weapons -

Agriculture

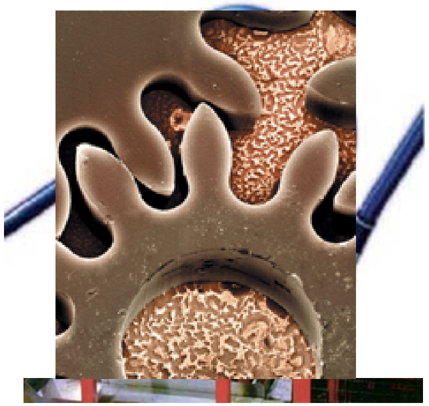
- Genetically modified crops surpass natural crops in acreage planted
- Consolidation of commercial farms
- Continued loss of habitat with localized famines -

Business



- Dominance of transnational companies
- Innovation in some sectors
  - Biotechnology and materials
- Economization in others
  - Communications and media -

Manufacturing

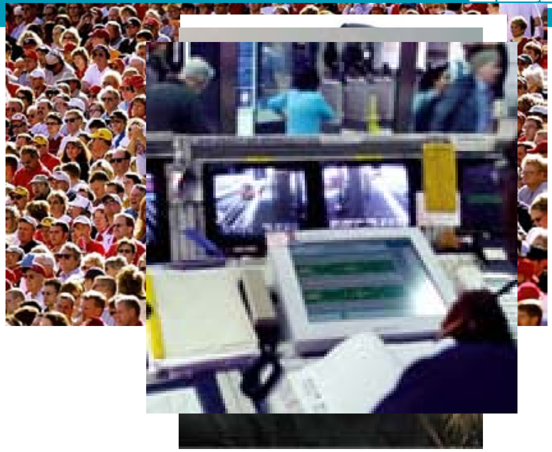


- Increased automation yielding personalized manufacturing
- New materials including ceramics, metallic glass, and nanotubes
- Nanoscale machines -

Transportation

- Hybrid connected cars dominate
- Increased mass transit
- Airline consolidation
- Regular commercial space travel -

Consumers



- Shift from mass to micromarkets
- Convergence of communication
- Pervasive personal assistants
- Increased loss of privacy -

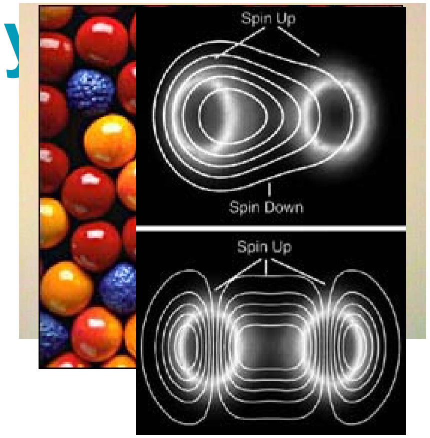
Entertainment

- Books are typically electronic or printed on demand
- Virtually all news and entertainment is delivered digitally across the Internet
- Immersive games and reality adventures dominate the landscape
- Complete photorealism in movies
- Local portable storage persists -

Medicine

- Genetic treatment commonly used to improve the quality and length of life
- Seamless and remote diagnosis and treatment with personal wellness assistants
- AIDS pandemic and growth of drug-resistant strains -

Science and Technology



- Pervasiveness of RFID & GPS
- Coded aperture imaging common
- Practical optical and limited quantum computing
  - Bekenstein’s Bound -

The State of Software - 2031

- Every advance leading to this state of the world in 2031 requires the presence of software yet-unwritten as of 2007
- The typical software-intensive system is
  - Continuously evolving
  - Connected, distributed & concurrent
  - Secure
  - Autonomic -

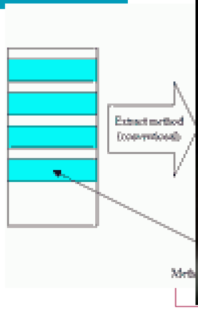
The State of Software - 2031

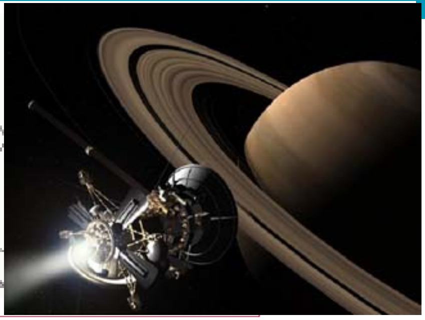
- Platforms
- Languages
- Operating systems & middleware
- Connection
- Security
- Autonomics
- Developer experience -

Platforms

- Moore’s law has died
- The typical personal computer contains multiple processors, a petabyte of main memory, an exabyte of external memory, and untethered terabit connectivity
- Virtual high resolution displays dominate; 3D windows, mice, gestures, and voice are the usual mechanisms for interaction
- Form factors will change such that most personal computers will be wearable or embedded
- Most software is embedded in devices -

Languages



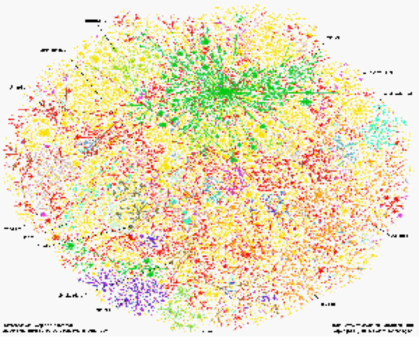


- Most programmers still write algorithmic snippets in the context of a sea of objects
- Legacy XML, Java, C++, UML, scripting languages persist
- Domain-specific frameworks are mainstream
- Some algorithmic breakthroughs have emerged-

Operating Systems

- Operating systems have largely been commoditized
- Middleware that does transaction isolation, load balancing, resource management, and data access still dominates
  - but it too has largely been commoditized -

Connection



- More than ever, the network is the computer
  - Monolithic -> client/server -> Web -> grid
- Network access is a global utility
- Not everything is an enterprise system, but most applications are connected to several -

Security

- New kinds of cybercrime have arisen
  - unlimited piles of money still do not yield secure systems
  - Air gaps are still not enough
- Rolling failures still plague some systems -

Autonomics

- No computer has yet passed the Turing Test (but we have come close)
- Most interesting systems exhibit signs of agency and self-repair
- The singularity <is> near -

Developer Experience

- Most developers have grown up believing that the Internet always existed
- Most programming is now done by domain-specific developers who only incidentally learn how to program
  - Most development occurs along the edge and the seams of systems
- There have been only incremental improvements in programmer productivity and the programming model
  - The developer experience is centered around the collaborative development environment
  - Distributed development is common
- Lawyers are now commonly a part of most development teams
  - Some projects are regulated and some individuals are registered -

The State of Software – 2031

2010s age of transparency

Software burrows itself into the interstitial spaces of society

2020s total dependency

Virtually every human activity touches and hence requires some software

2030s rise of the machines

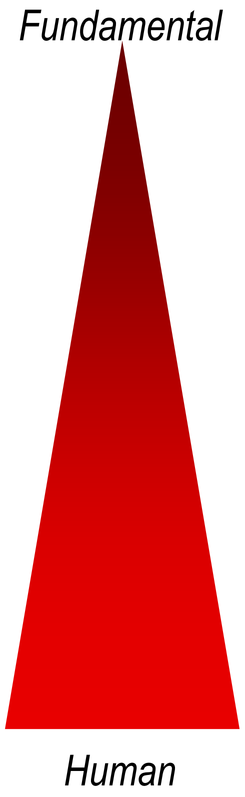
Semiautonomous entities with varying degrees of agency amplify human activity -

The Software Development Paradox

- Not everything we want to build can be built
  - Theoretical/technical/pragmatic limitations
- Not everything we want to build should be built
  - Economic/social/political limitations
- The complexity ceiling

Getting From Here to There

- The limits of software
  - The laws of physics
  - The laws of software
  - The challenge of algorithms
  - The difficulty of distribution
  - The problems of design
  - The importance of organization
  - The impact of economics
  - The influence of politics
  - The limits of human imagination



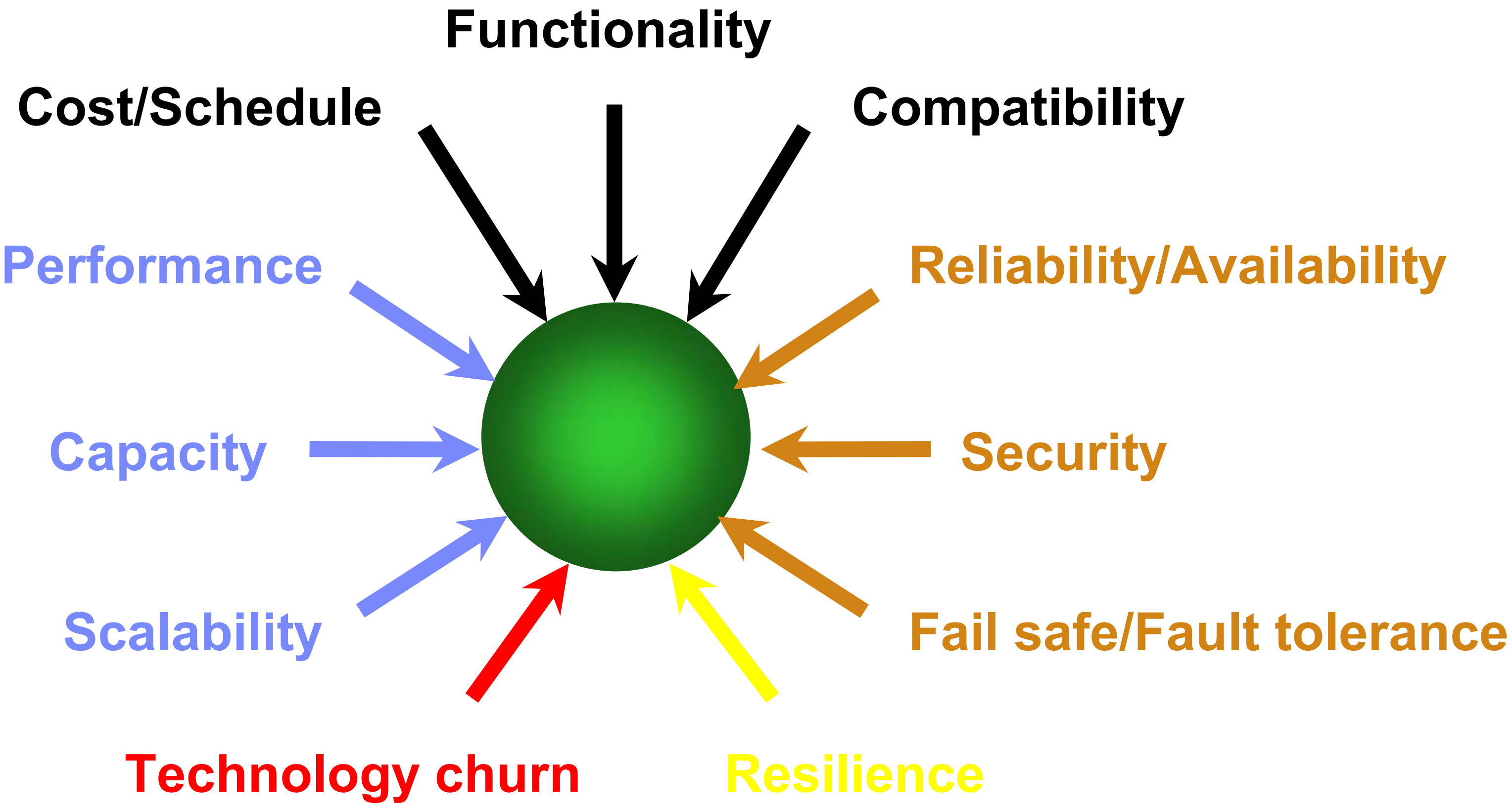
The entire history of software engineering
Is one of rising levels of abstraction

Languages: Assembly -> Fortran/COBOL -> Simula -> C++ -> Java
Platforms: Naked HW -> BIOS -> OS -> Middleware -> Domain-specific
Processes: Waterfall -> Spiral -> Iterative -> Agile
Architecture: Procedural -> Object Oriented -> Service Oriented
Tools: Early tools -> CLE -> IDE -> XDE -> CDE
Enablement: Individual -> Workgroup -> Organization

Getting From Here To There

- Languages for systems
- Mechanisms for interconnection
- Architectural patterns
- Tools for understanding and reasoning about continuously evolving systems
- Tools for collaboration and organization

Forces In Software



Functionality

Cost/Schedule

Compatibility

Performance

Reliability/Availability

Capacity

Security

Scalability

Fail safe/Fault tolerance

Technology churn

Resilience

Points Of Friction

- Start up
- Work product collaboration
- Communication
- Time starvation
- Stakeholder cooperation
- Stuff that doesn’t work

Improving Software Build Economics

Legacy system upgrades

e-business, Web applications

SW Maintenance

New Developments

New Releases

Packaged applications

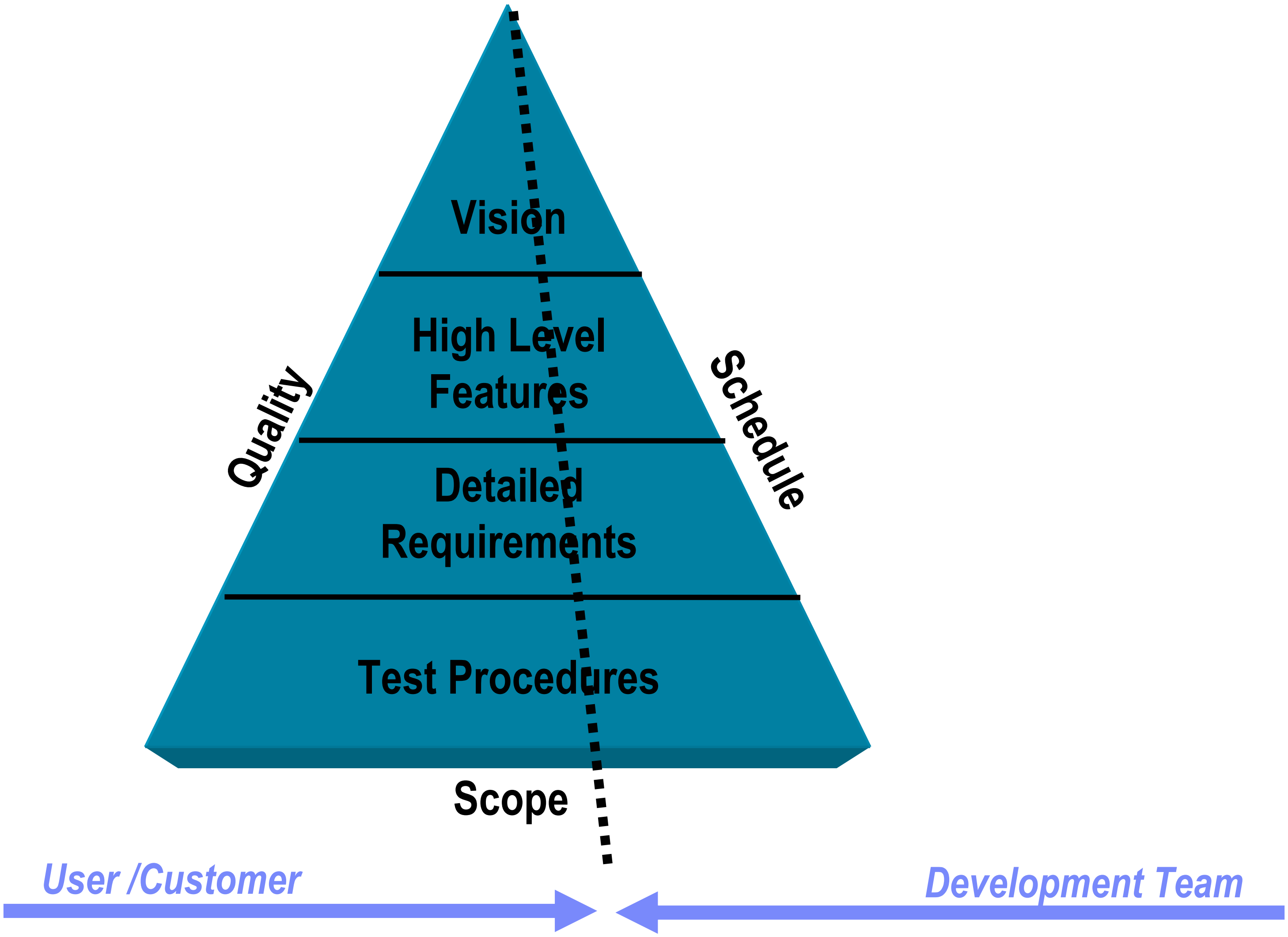
$$\begin{array}{c}\text{Time or Cost}\\ \text{To Build}\end{array} = (\text{Complexity})^{(\text{Process})} * (\text{Team}) * (\text{Tools})$$

Complexity	→	Volume of human-generated code
Process	→	Methods, notations, maturity
Team	→	Skill set, experience, motivation
Tools	→	Process automation

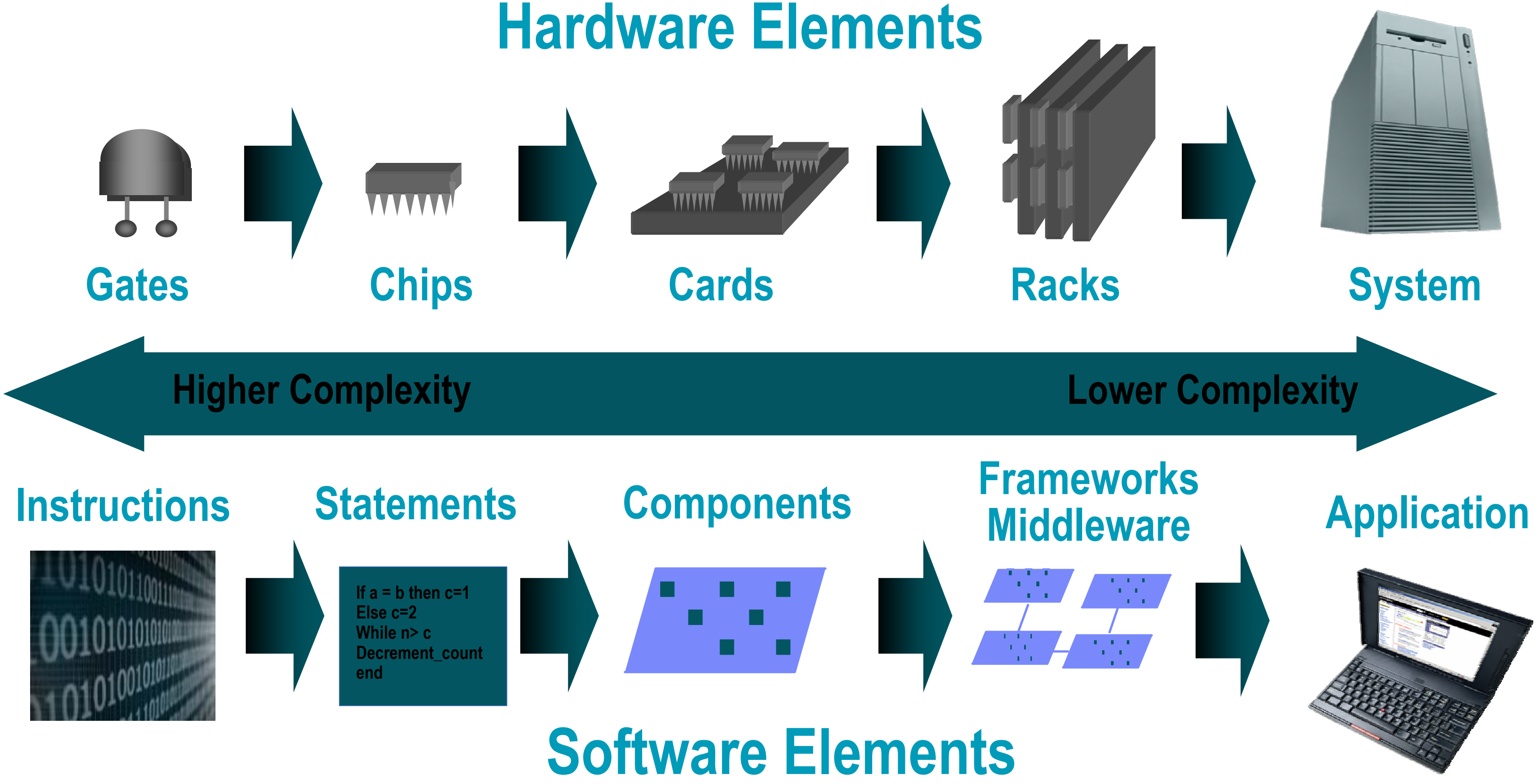
Reduce Complexity

- Manage scope
- Middleware and architecture frameworks
- Model driven development

Balance of Scope, Quality and Schedule



Reduce the Proportion of Hand-Written Code



Hardware Elements

Gates → Chips → Cards → Racks → System

Instructions → Statements → Components → Frameworks Middleware → Application

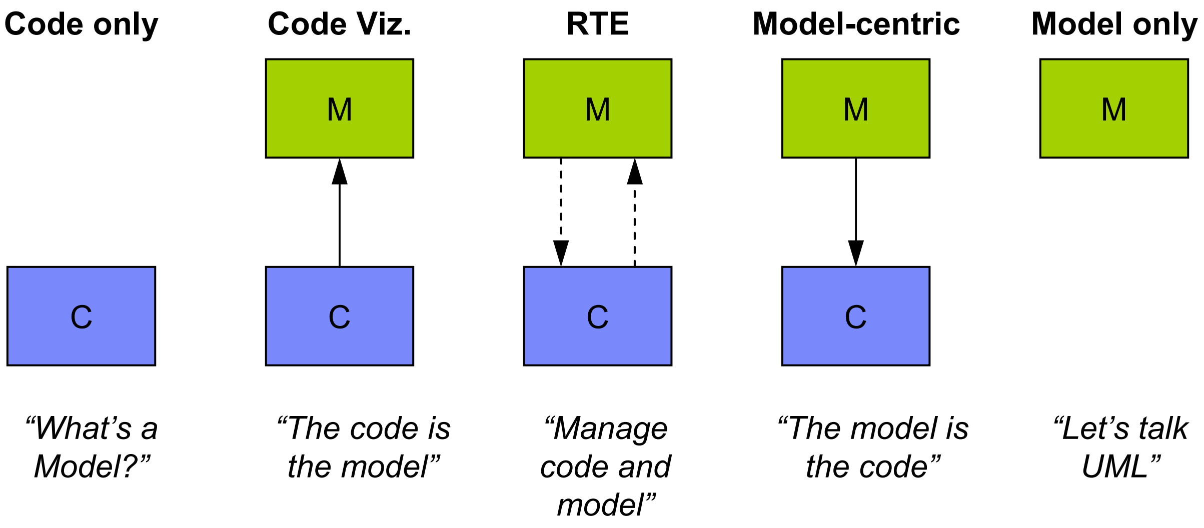
```
If a = b then c=1
Else c=2
While n> c
Decrement_count
end
```

Software Elements

Use higher level software components (services and patterns) to enable economically significant reuse

Raise the Level of Abstraction Via MDD

- Provide a blueprint of the software design
- Hide and emphasize details as necessary
- Promote unambiguous communication



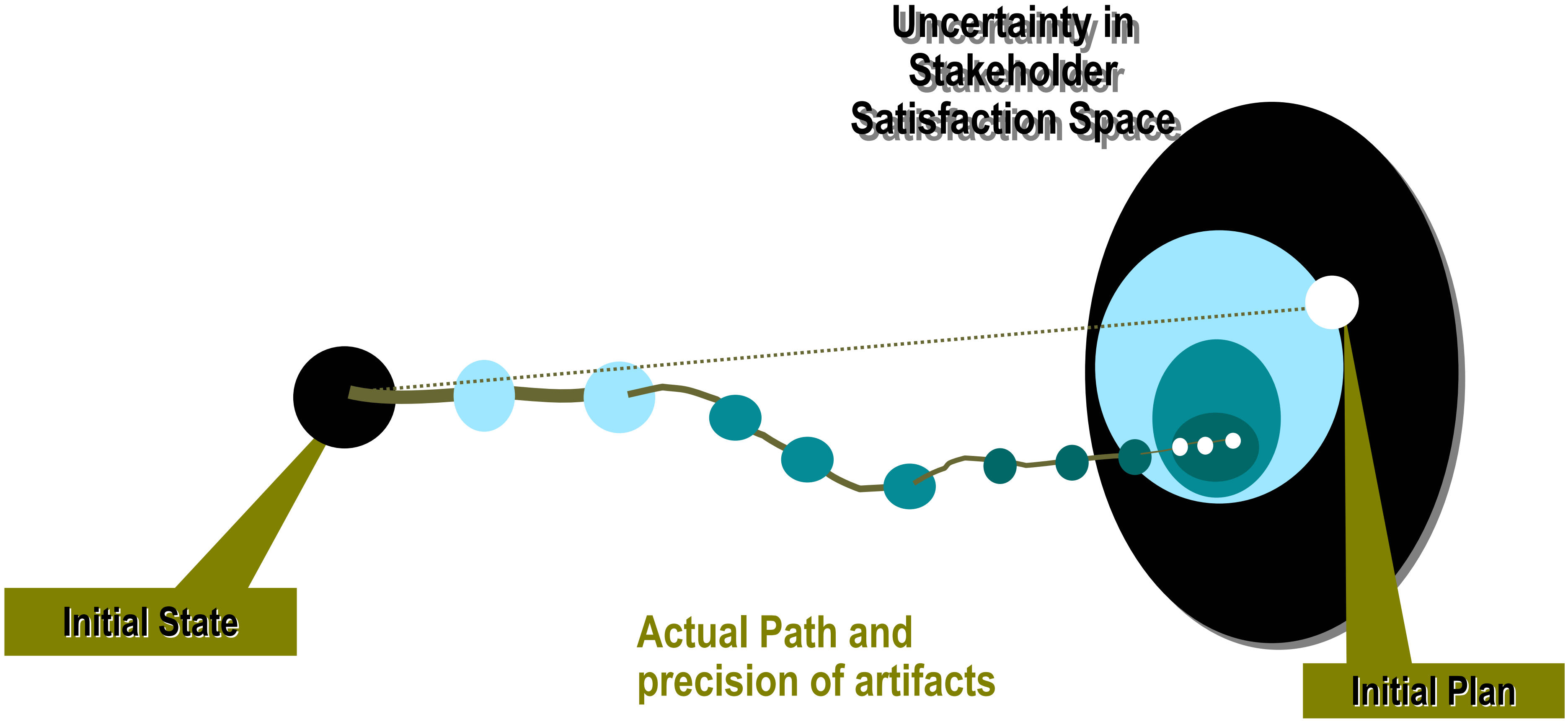
The Spirit of Iterative Development

Conventional	Modern
Activity-based management	Results-based management
Adversarial relationships	Honest communication
Requirements first	Architecture first
Early requirements freeze	Scope management
Early false precision	Evolving artifacts
“More detail = more quality”	Scope (Problem specs)
	Design (Solution specs)
	Constraints (Planning specs)
Context-independent recipes	Context dependent execution
Too much/too little process	Right-size the process

What Are the Most Useful (Tangible) Results?

- Business milestone achievement
  - Stakeholder consensus on canceling, continuing, accelerating a project
  - Award fees, payments, sales, profits, contract awards
- Demonstrable software releases with measurable characteristics
  - Executable components, test suites
  - Performance, feature growth, reliability, usability, test cases
  - Release trends: changes from previous release

Iterations & Demonstrable Progress


Uncertainty in Stakeholder Satisfaction Space
Initial State
Actual Path and precision of artifacts
Initial Plan

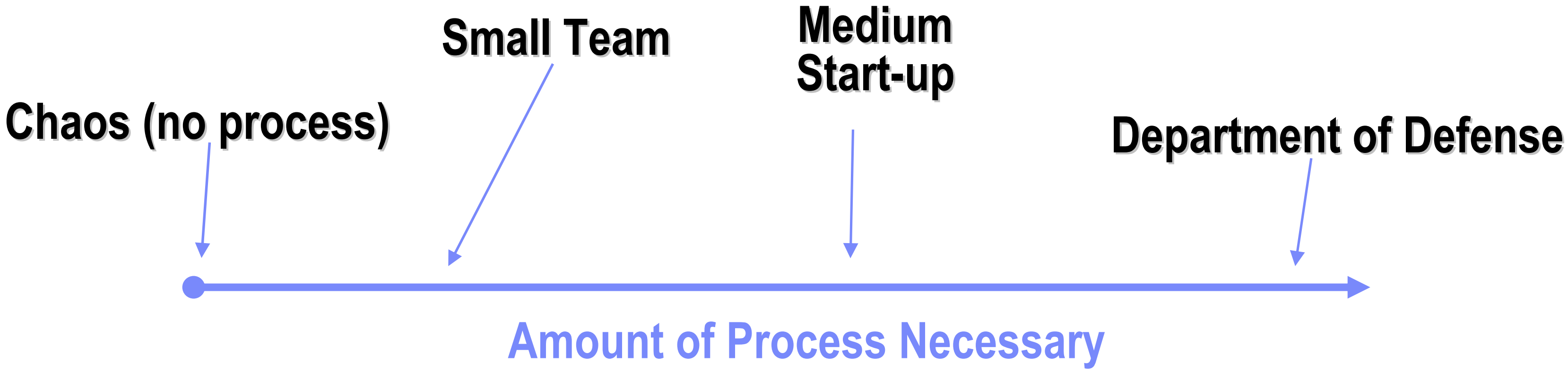
What Are Less Useful (Speculative) Results?

- Measurements of the “means” rather than the “ends”
  - Intermediate artifacts (documents, models, plans)
  - Activity measurements (productivities, requirements, design, etc)
  - Inspections and human reviews
  - Earned value derived from artifacts, activities, effort expended
  - Process implementations
- Process maturity
  - Measuring process attributes rather than products of the process

Why Do We Need Methods?

- We desire
  - Predictability
  - Repeatability
  - To attack risk
  - To deploy the right system
- Low ceremony methods are neither predictable nor repeatable
- High ceremony methods have a difficult time attacking risk and building the right system
- Ultimately, we desire a good return on investment

How Much Process Is Necessary?



Chaos (no process)

Small Team

Medium Start-up

Department of Defense

Amount of Process Necessary

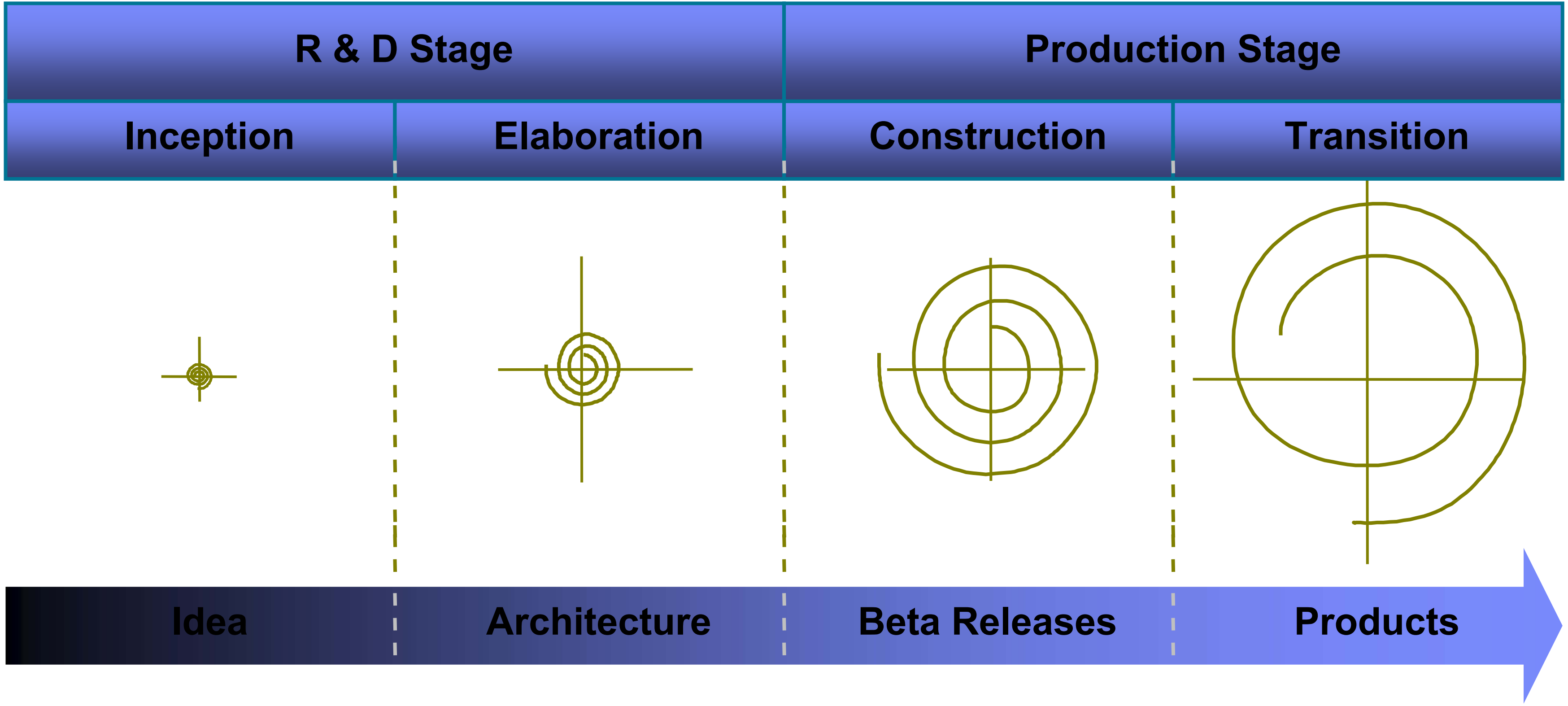
When is Less Appropriate?

- Co-located teams
- Smaller projects (less than 25)
- Straightforward projects
- Internally imposed constraints

When is More Appropriate?

- Distributed teams
- Large projects (25, 125, 625)
- Complex projects
- Externally imposed constraints
  - Standards
  - Contractual requirements
  - Legal requirements

Process Evolution Over the Life Cycle



Prototypes	Change managed baselines
Coarse artifacts	Elaborate artifacts
Major risk items	Low Risk Items
Creative, judgment	Engineering, reasoned
Maneuverable processes	Well-instrumented processes

Process “Strength” Over the Life Cycle

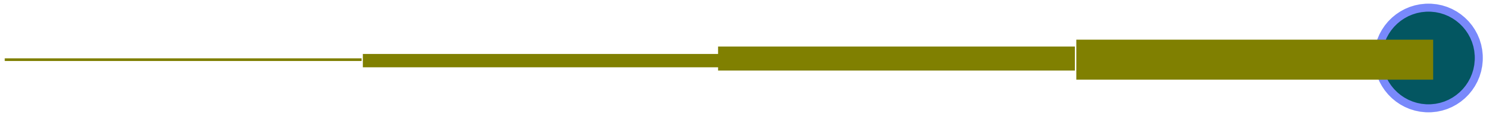
Weak process influence

(Optimized for rapid adaptation to change)

→

Strong process influence

(Optimized for converging on quality product releases)



Product

Release

$$\text{Process Weight} \rightarrow \frac{\text{Product Quality}}{\text{(Time to Release)}}$$

Top 10 Principles: Conventional Process

1. Freeze requirements before design.
2. Forbid coding prior to detailed design review.
3. Use a higher order programming language.
4. Complete unit testing before integration.
5. Maintain detailed traceability among all artifacts.
6. Document and maintain the design.
7. Assess quality with an independent team.
8. Inspect everything.
9. Plan everything early with high fidelit
10. Control source code baselines rigorously.

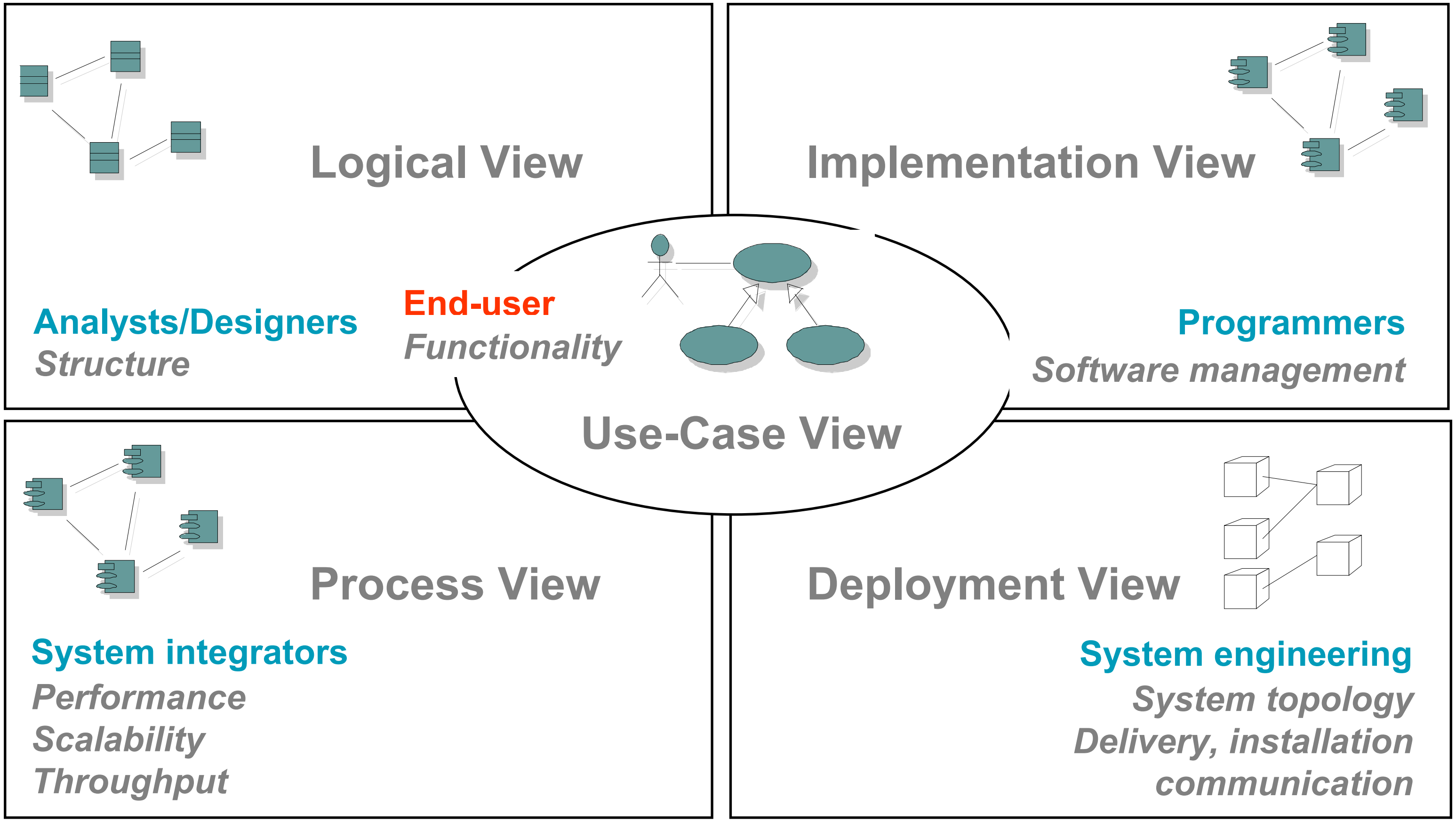
Top 10 Principles – Modern (Iterative) Process

1. Focus the process on the architecture first
2. Attack risks early with an iterative life cycle
3. Emphasize component-based development
4. Change management of all artifacts
5. Simplify change freedom with round-trip engineering
6. Use rigorous, model-based design notation
7. Instrument the process for objective quality control
8. Emphasize demonstration-based assessment
9. Plan releases with evolving levels of detail
10. Establish a scalable, configurable process

RUP Basics (Implicit)

- Develop only what is necessary
- Focus on valuable results, not on how the results are achieved
- Minimize the production of paperwork
- Be flexible
- Learn from your mistakes
- Revisit your risks regularly
- Establish objective, measurable criteria for your progress
- Automate what is human-intensive, tedious, and error-prone
- Use small, empowered teams
- Have a plan

Architecture First



Logical View

Analysts/Designers

Structure

Implementation View

Programmers

Software management

End-user

Functionality

Use-Case View

Process View

System integrators

Performance

Scalability

Throughput

Deployment View

System engineering

System topology

Delivery, installation

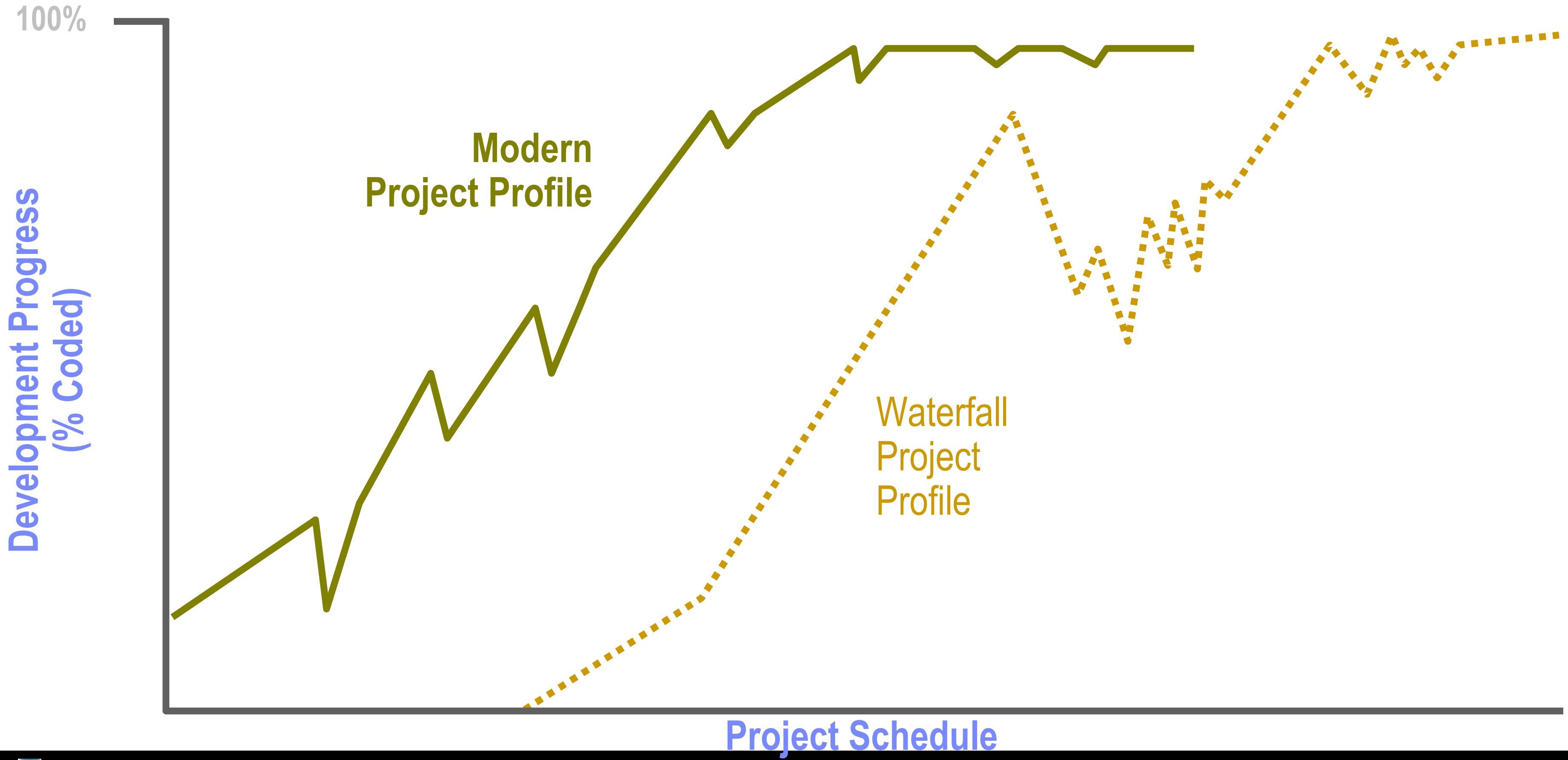
communication

Conceptual

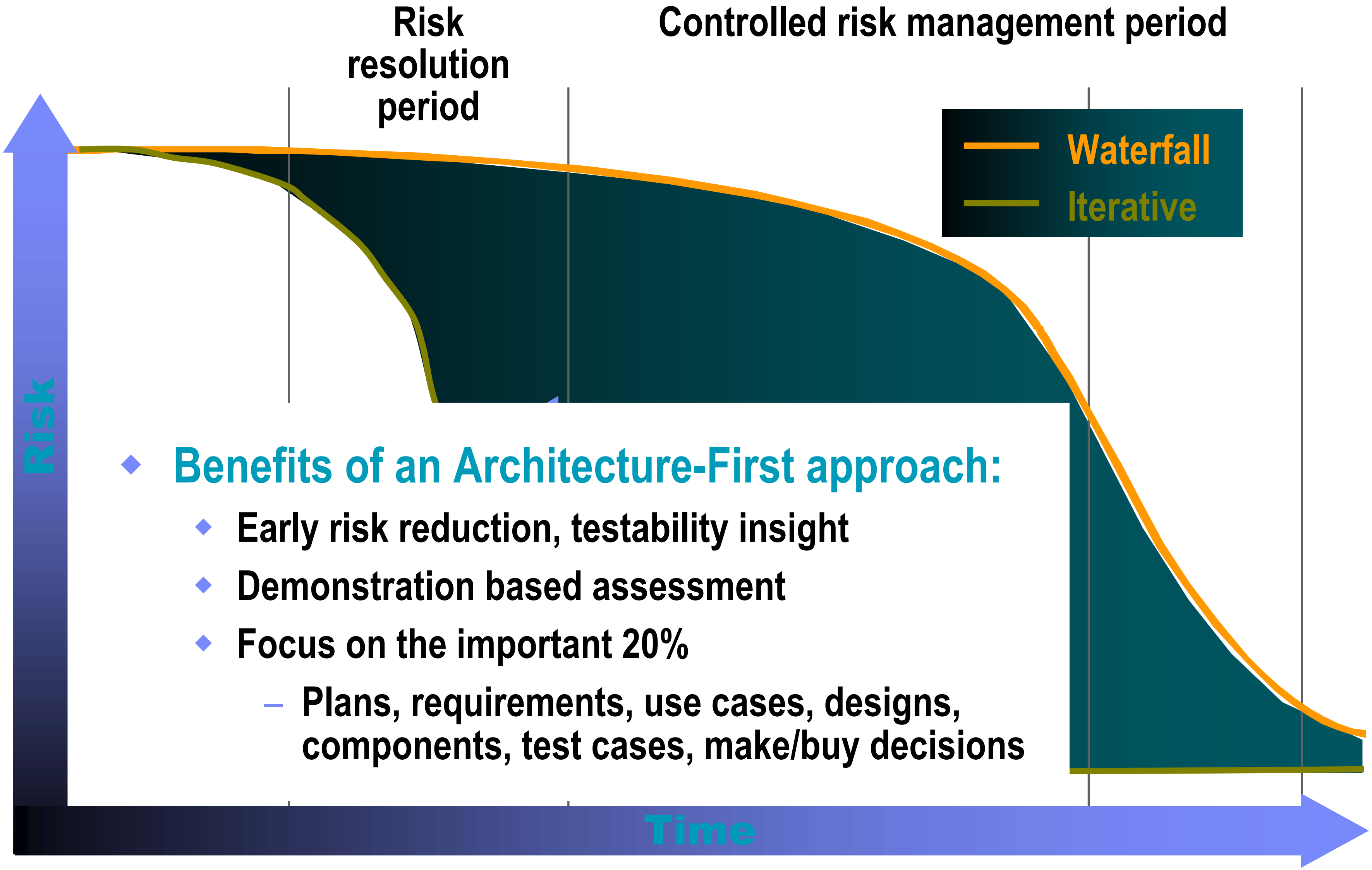
Physical

Reduce Scrap/Rework: *Use an Iterative Process*

Prototypes → Architecture → Functional Releases → Product Release



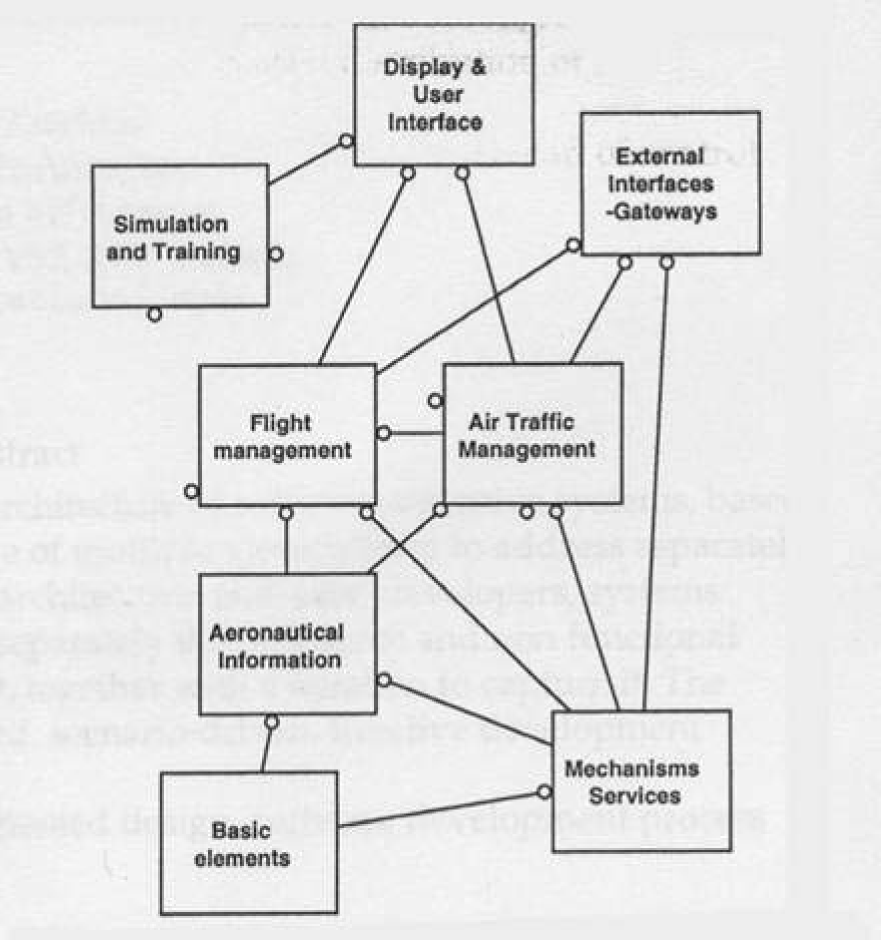
Attack Significant Risks Early: *Architecture First*



- **Benefits of an Architecture-First approach:**
  - Early risk reduction, testability insight
  - Demonstration based assessment
  - Focus on the important 20%
    - Plans, requirements, use cases, designs, components, test cases, make/buy decisions

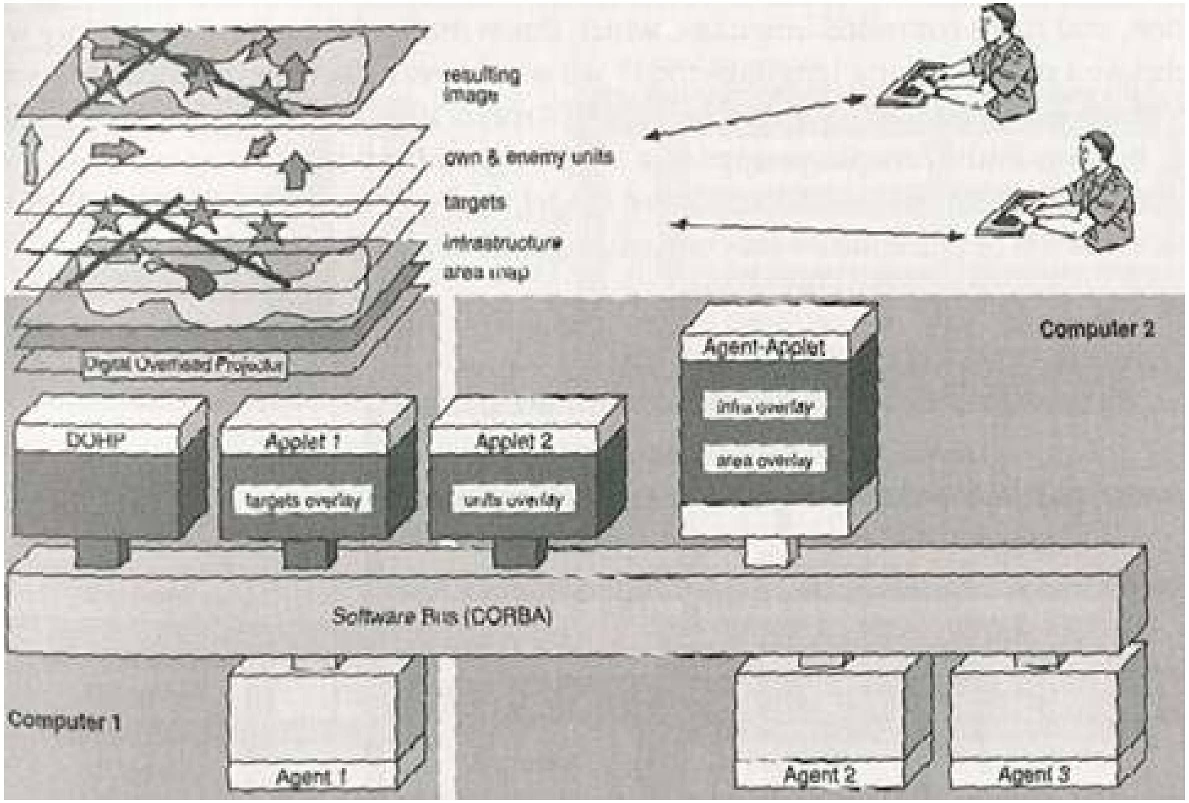
Everything has an architecture

Gallery of Software Architecture: Air Traffic Control



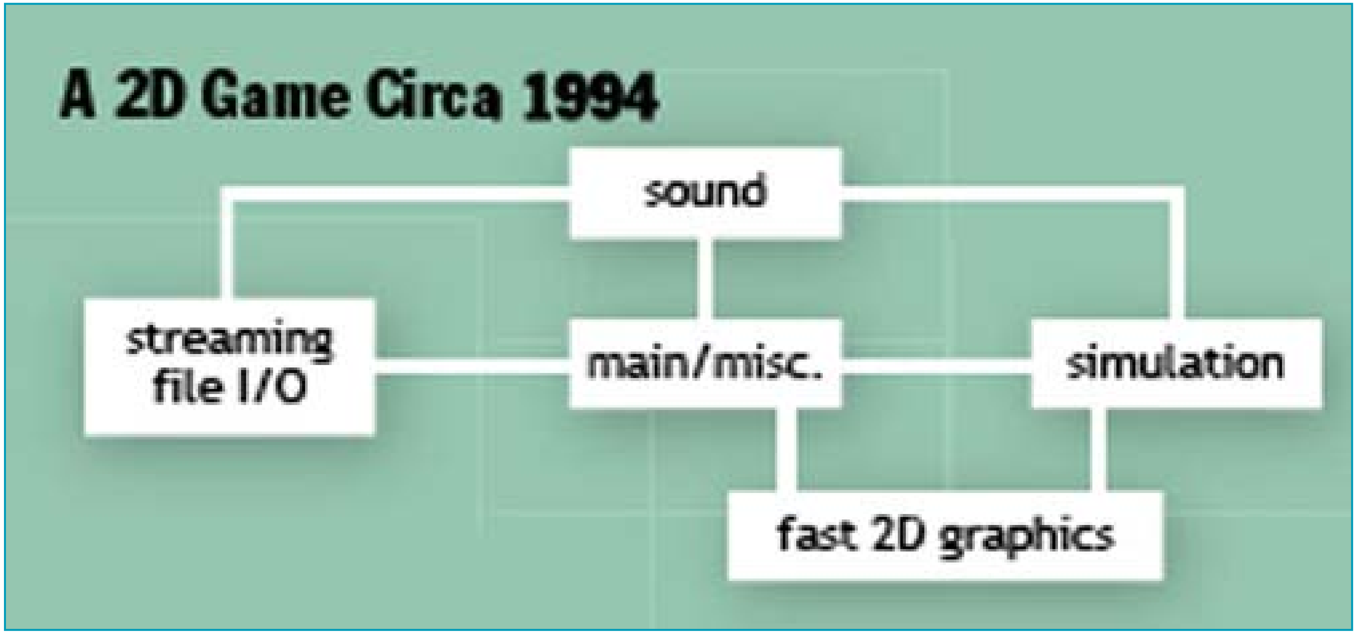
Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Software Architecture: C3I



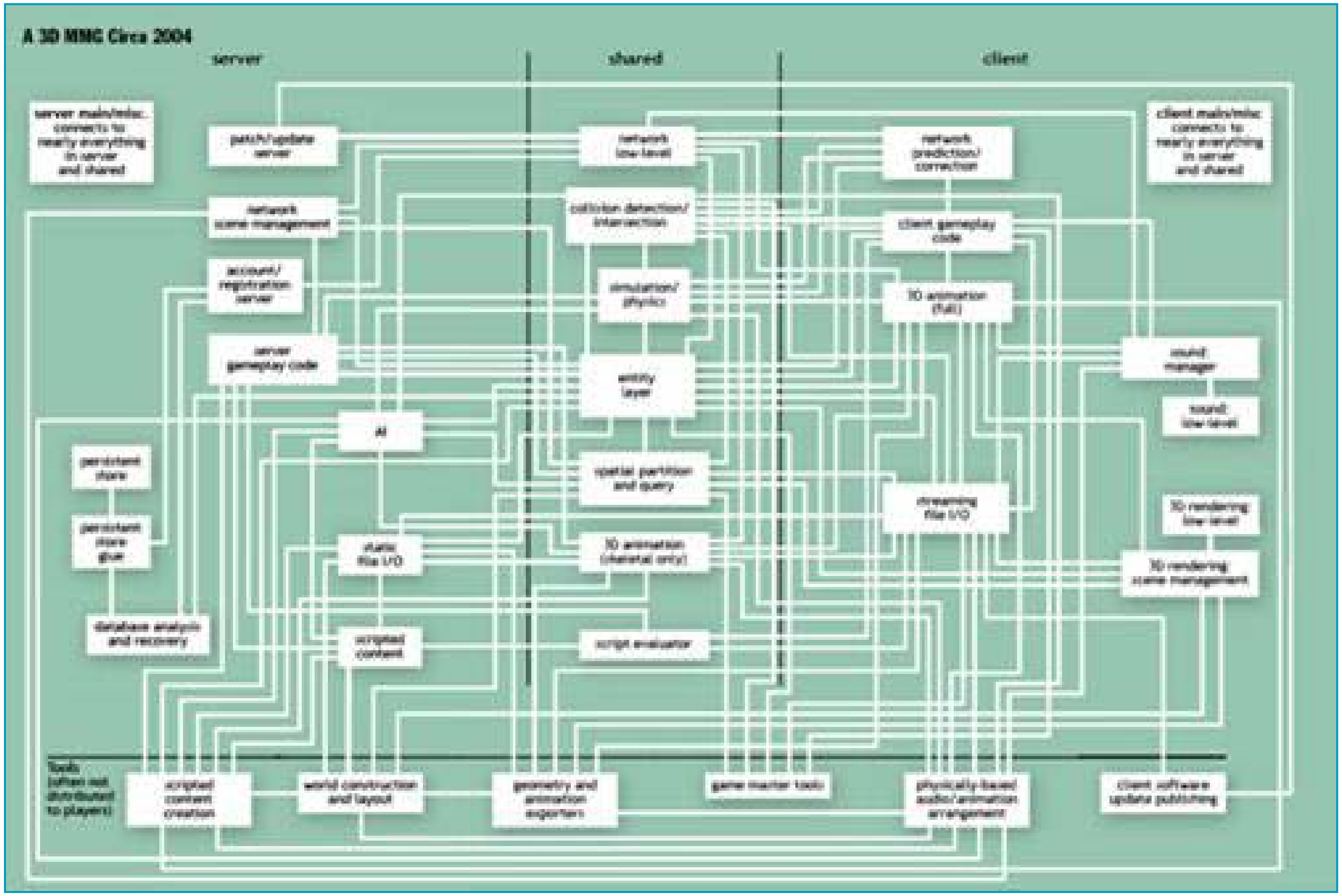
Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Software Architecture: Games



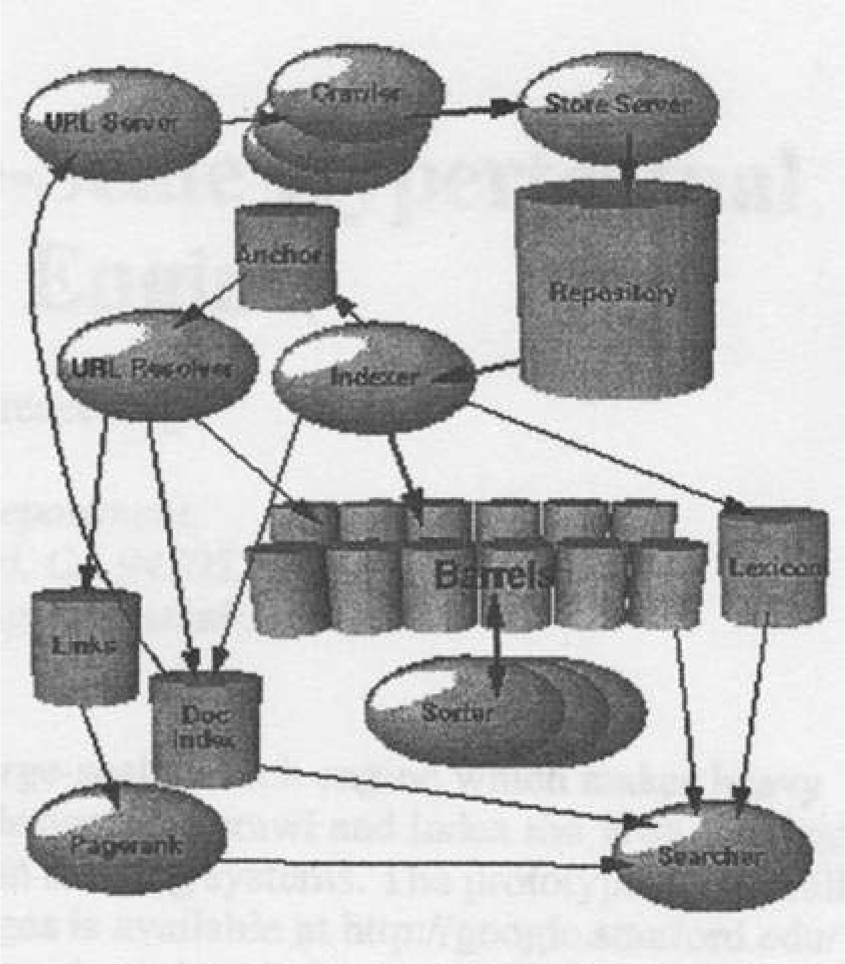
Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Architecture: Games


A 3D MMG Circa 2004
server
shared
client
server main/misc: connects to nearly everything in server and shared
patch/update server
network scene management
account/registration server
server gameplay code
AI
persistent store
persistent store glue
static file I/O
database analysis and recovery
scripted content
network low-level
collision detection/intersection
simulation/physics
entity layer
spatial partition and query
3D animation (skeletal only)
script evaluator
network prediction/correction
client gameplay code
3D animation (full)
streaming file I/O
client main/misc: connects to nearly everything in server and shared
sound manager
sound: low-level
3D rendering low-level
3D rendering scene management
Tools (often not distributed to players)
scripted content creation
world construction and layout
geometry and animation exporters
game master tools
physically-based audio/animation arrangement
client software update publishing

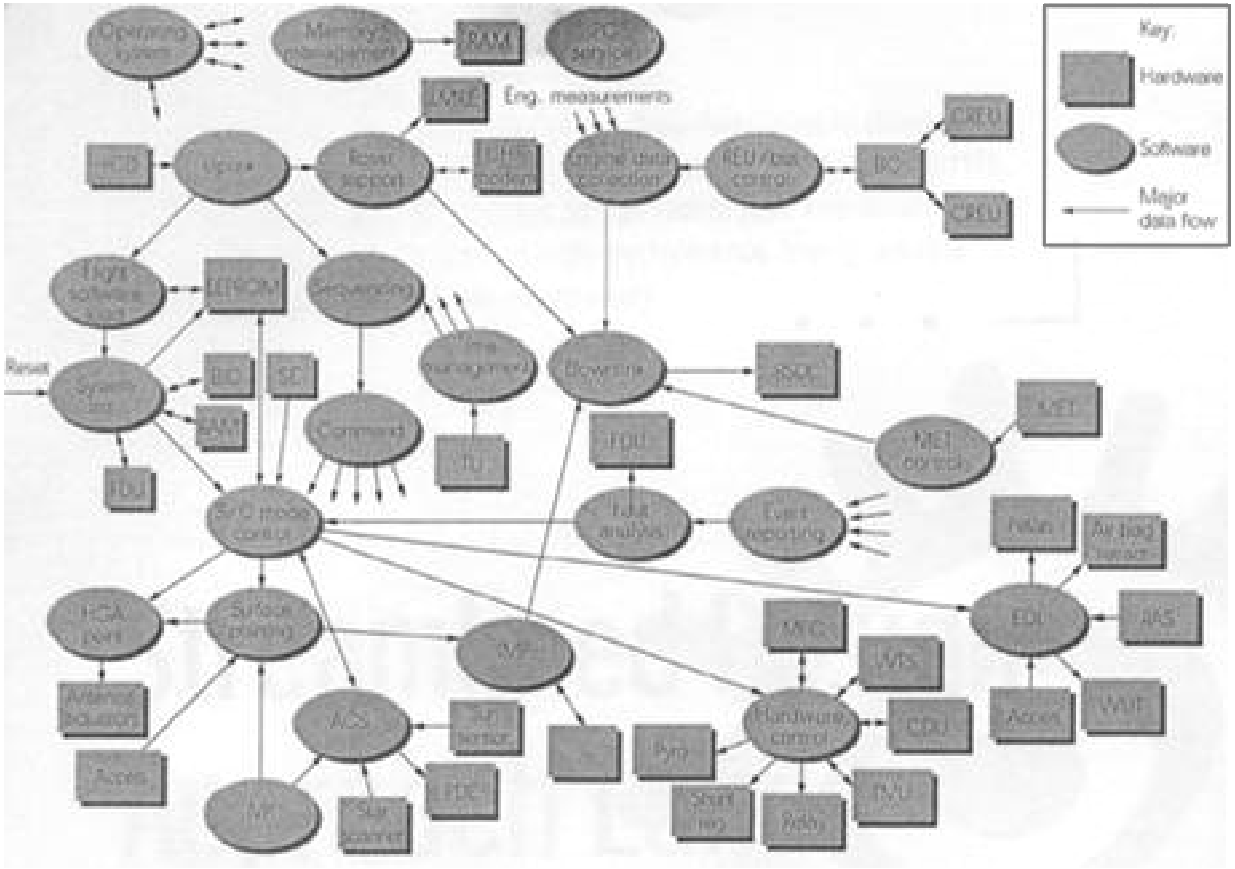
Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Software Architecture: Google



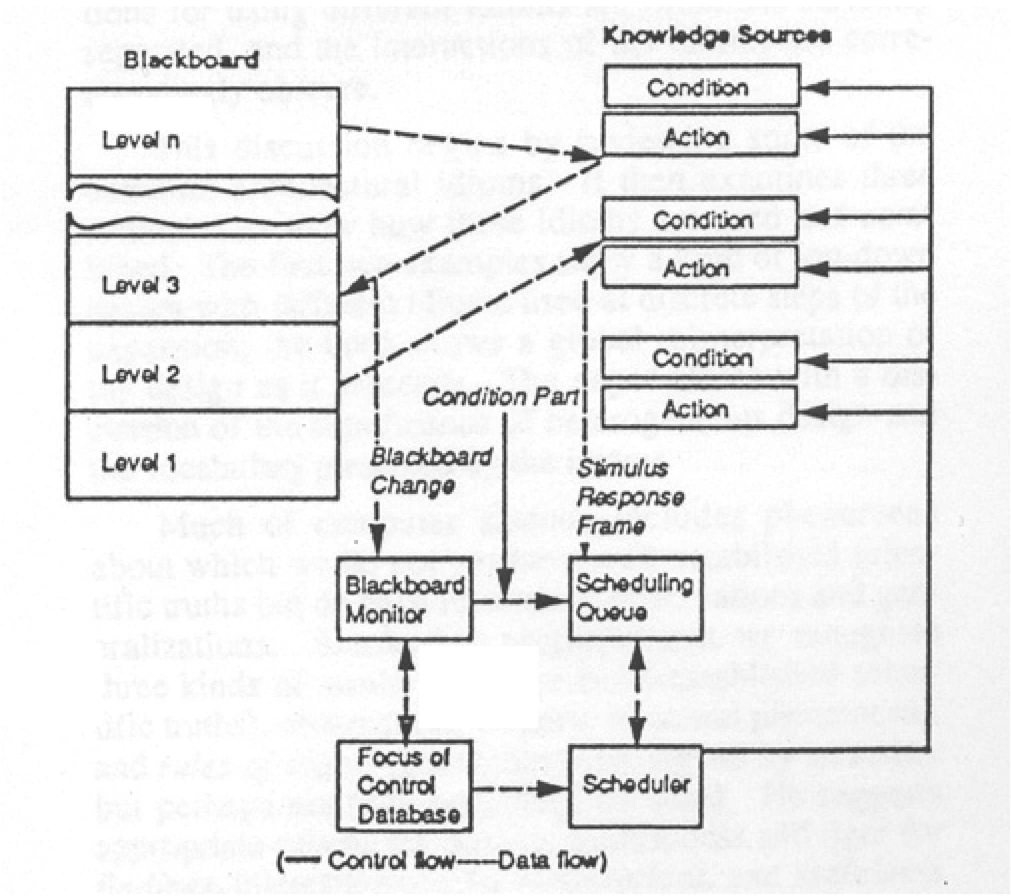
Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Software Architecture: Pathfinder



Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

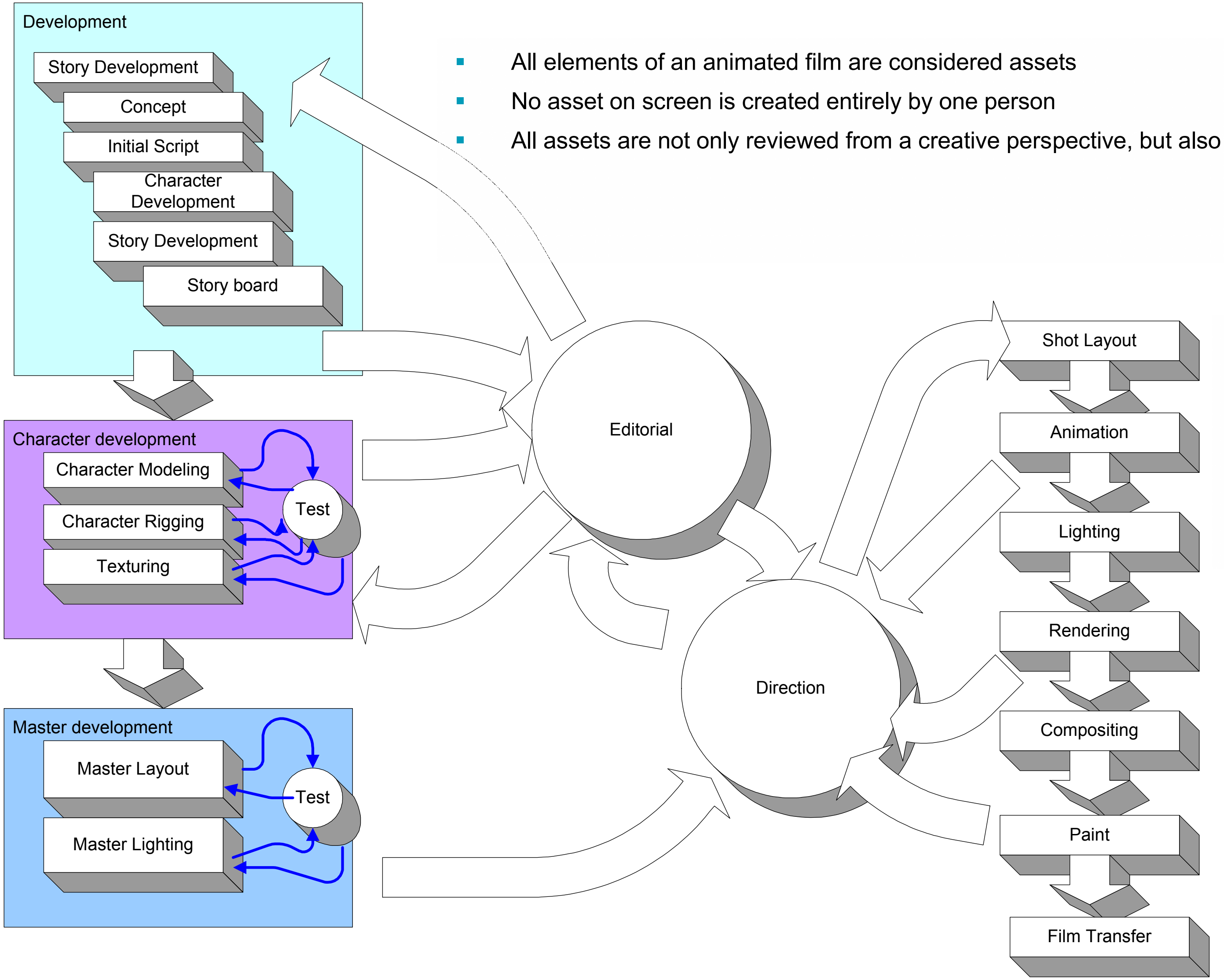
Gallery of Software Architecture: Speech Recognition



Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Software Architecture: Film

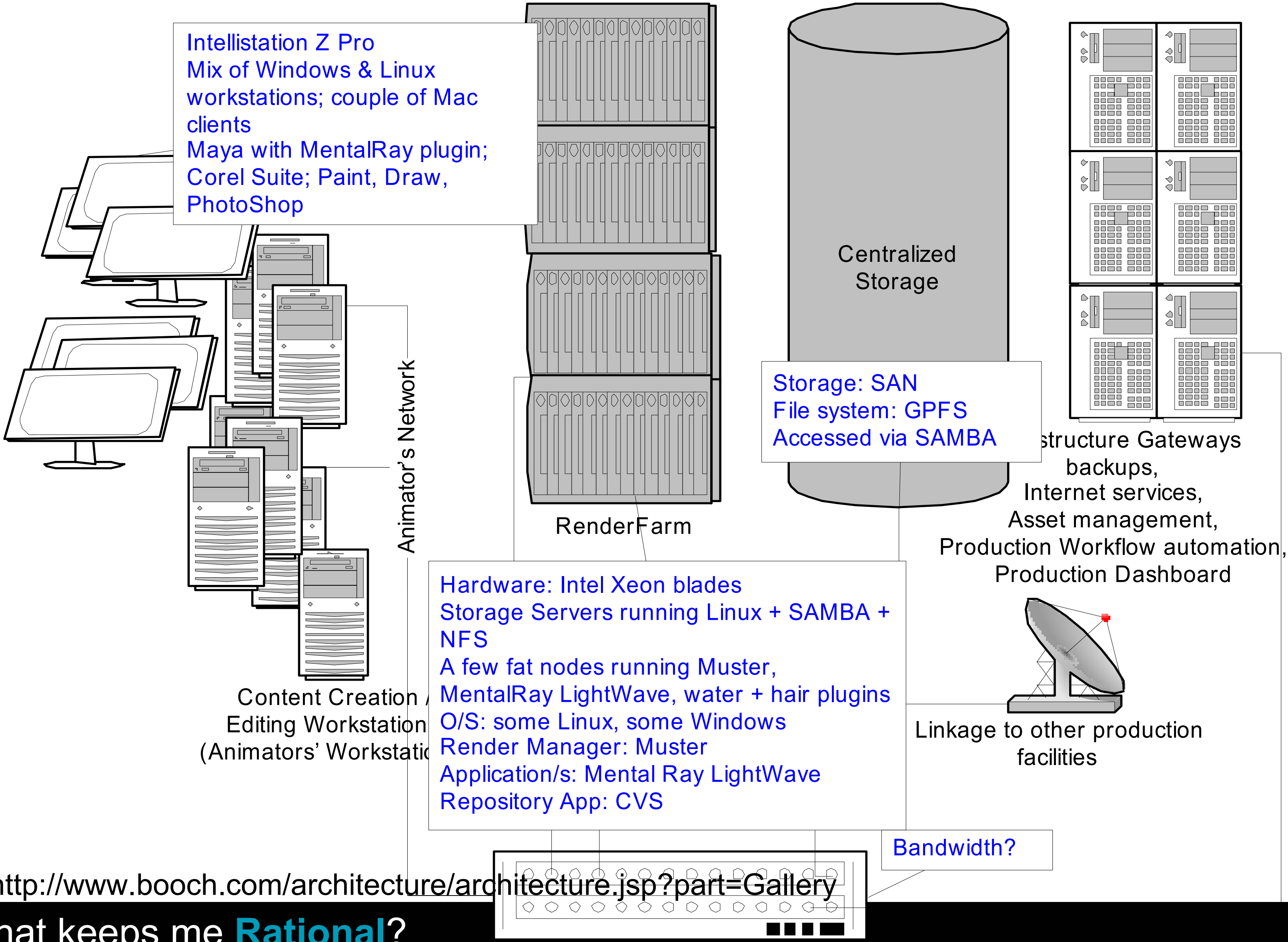
- All elements of an animated film are considered assets
- No asset on screen is created entirely by one person
- All assets are not only reviewed from a creative perspective, but also tested from a technical perspective



- Each frame of animation is made up of multiple elements
- Each element goes through the process

Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Gallery of Software Architecture: Film



Intellistation Z Pro
Mix of Windows & Linux workstations; couple of Mac clients
Maya with MentalRay plugin; Corel Suite; Paint, Draw, PhotoShop

Animator's Network

Content Creation /
Editing Workstation
(Animators' Workstatic

RenderFarm

Hardware: Intel Xeon blades
Storage Servers running Linux + SAMBA + NFS
A few fat nodes running Muster,
MentalRay LightWave, water + hair plugins
O/S: some Linux, some Windows
Render Manager: Muster
Application/s: Mental Ray LightWave
Repository App: CVS

Centralized Storage

Storage: SAN
File system: GPFS
Accessed via SAMBA

structure Gateways
backups,
Internet services,
Asset management,
Production Workflow automation,
Production Dashboard

Linkage to other production facilities

Bandwidth?

Source: http://www.booch.com/architecture/architecture.jsp?part=Gallery

Why Architecture?

- In hyper-productive projects
  - Process centers around growing an executable architecture
  - Well-structured systems are full of patterns
- Why architecture?
  - Risk-confrontive
  - Simplicity
  - Resilience

What We Know We Know

- Every software-intensive system has an architecture
- We generally understand what software architecture is and what it is not
- Different stakeholders have different concerns and therefore different viewpoints
- All well-structured software-intensive systems are full of patterns

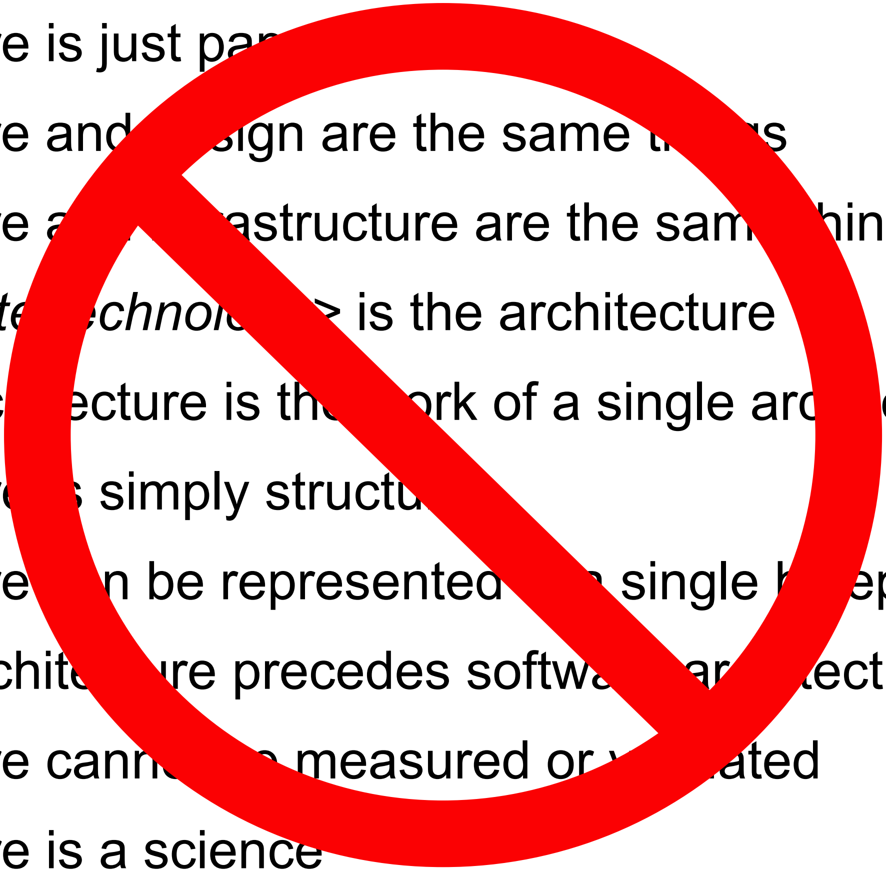
What We Are Fairly Certain We Know

- We are starting to develop a profession of software architecture
- We are starting to see the emergence of domain-specific software architectures

What We Know We Don’t Know

- We still don’t have formal architectural representations that scale
- We don’t yet have a good understanding of the architectural patterns that are found among domains.

Misconceptions About Architecture



- Architecture is just paper
- Architecture and design are the same things
- Architecture and infrastructure are the same things
- *<my favorite technology>* is the architecture
- A good architecture is the work of a single architect
- Architecture is simply structure
- Architecture can be represented in a single blueprint
- System architecture precedes software architecture
- Architecture cannot be measured or validated
- Architecture is a science
- Architecture is an art

Source: Kruchten

Architecture Defined

- IEEE 1471-2000
  - Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution
- Software architecture encompasses the set of significant decisions about the organization of a software system
  - Selection of the structural elements and their interfaces by which a system is composed
  - Behavior as specified in collaborations among those elements
  - Composition of these structural and behavioral elements into larger subsystems
  - Architectural style that guides this organization

Source: Booch, Kruchten, Reitman, Bittner, and Shaw

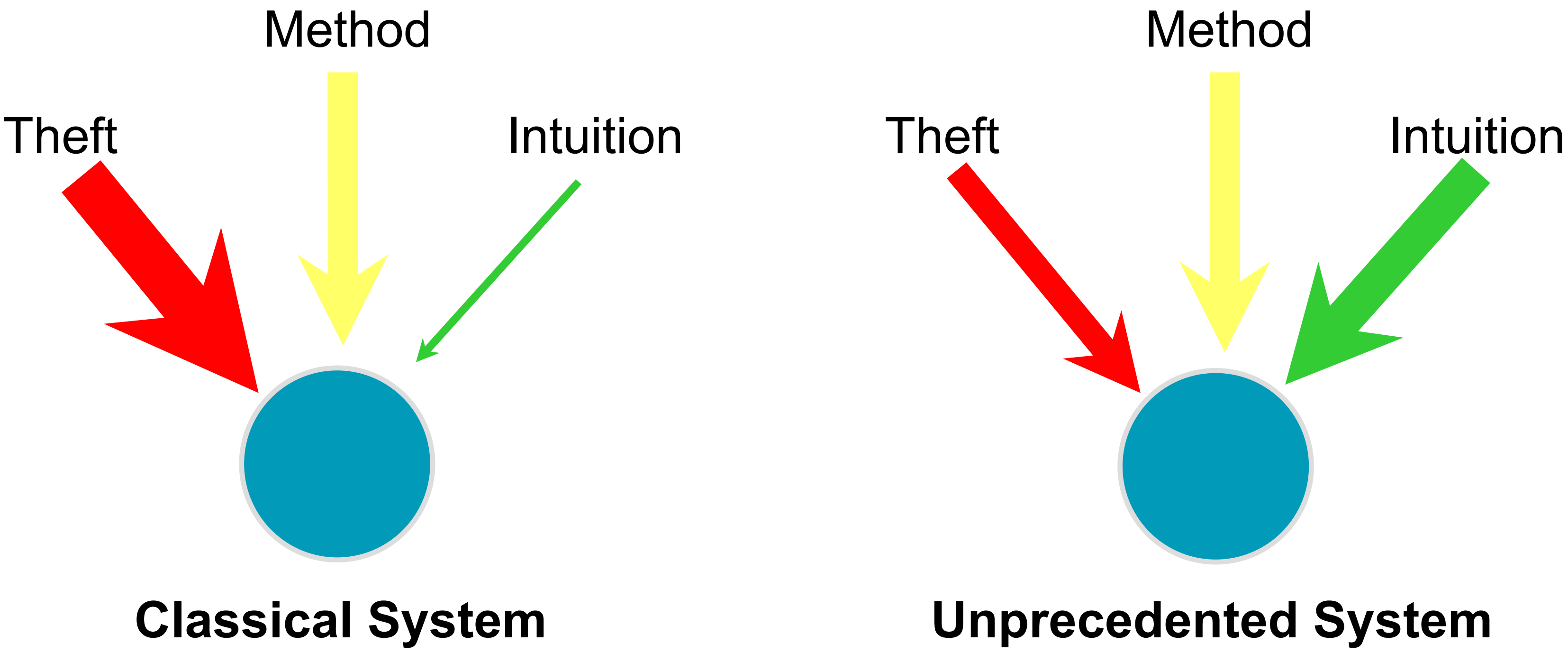
Architecture Defined

- Software architecture also involves
  - Functionality
  - Usability
  - Resilience
  - Performance
  - Reuse
  - Comprehensibility
  - Economic and technology constraints and tradeoffs
  - Aesthetic concerns

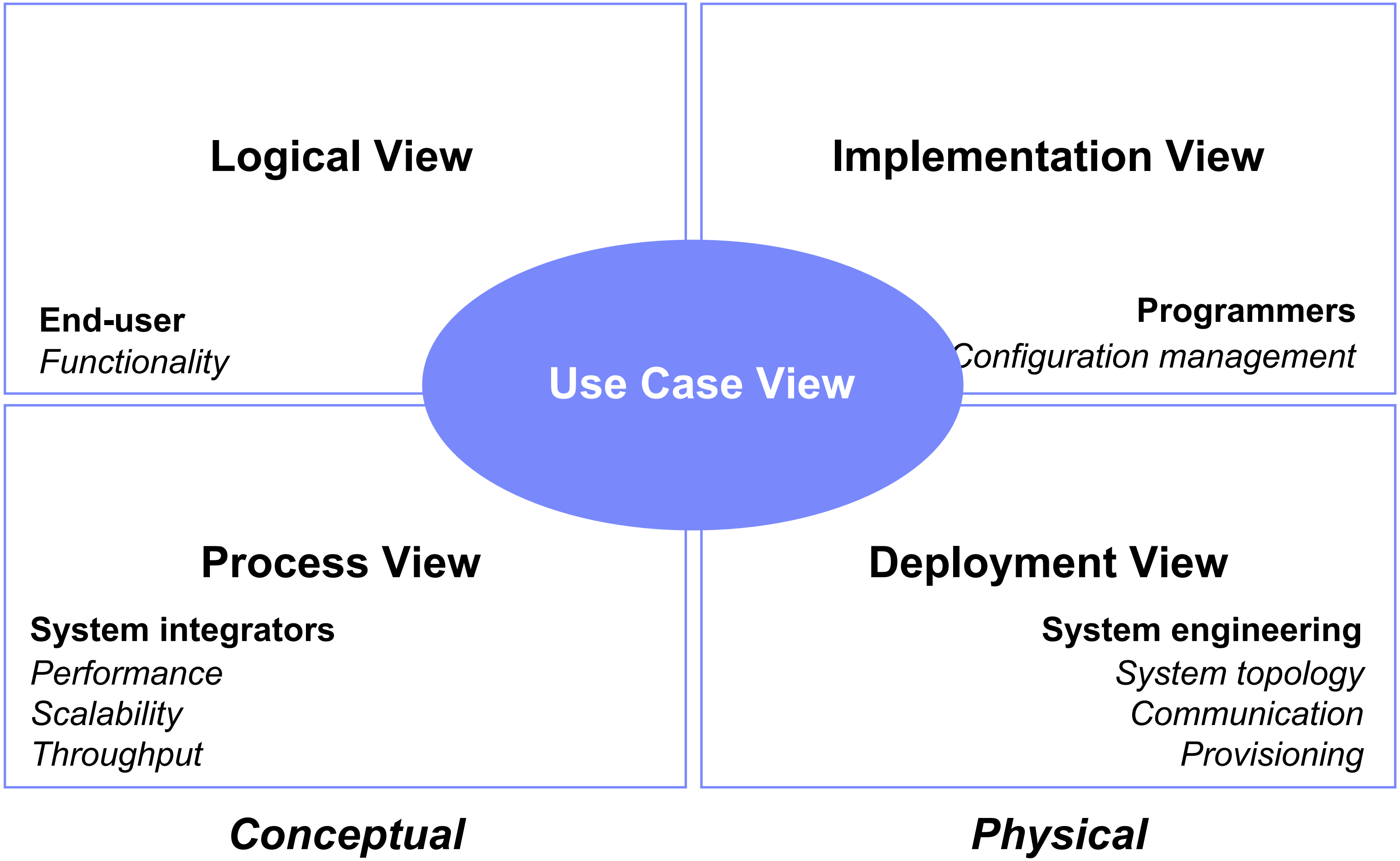
Architectural Style Defined

- Style is the classification of a system’s architecture according to those with similar patterns
- A pattern is a common solution to a common problem
  - Patterns may be classified as idioms, mechanisms, or frameworks

Sources of Architecture



Representing Software Architecture



Logical View

End-user
Functionality

Implementation View

Programmers
Configuration management

Use Case View

Process View

System integrators
Performance
Scalability
Throughput

Deployment View

System engineering
System topology
Communication
Provisioning

Conceptual

Physical

Source: Kruchten, “The 4+1 Model View”

Model, Views, Concerns, & Stakeholders

- A model is a simplification of reality, created in order to better understand the system being created; a semantically closed abstraction of a system
- A view is a representation of a whole system from the perspective of a related set of concerns
- A concern is those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders
- A stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system

Stakeholders & Views

- Architecture is many things to many different stakeholders
  - End user
  - Customer
  - Sys admin
  - Project manager
  - System engineer
  - Developer
  - Architect
  - Maintainer
  - Tester
  - Other systems
- Multiple realities, multiple views and multiple blueprints exist

Fundamentals

- Crisp abstractions
- Clear separation of concerns
- Balanced distribution of responsibilities
- Simplicity

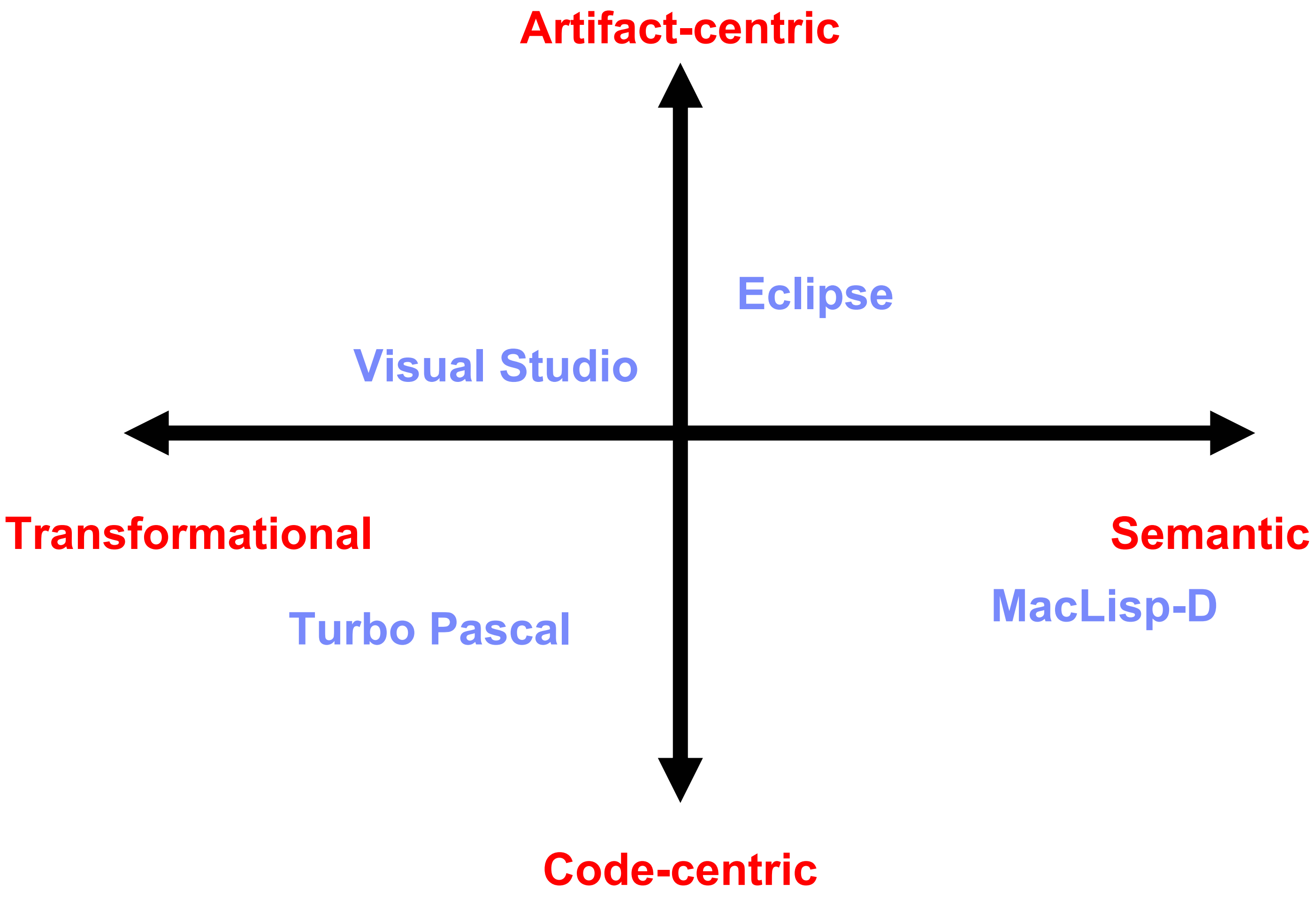
The best architectures are full of patterns

Patterns

- A pattern is a common solution to a common problem
- A pattern codifies specific knowledge collected from experience in a domain
- A pattern resolves forces in context
- All well-structured systems are full of patterns
  - Idioms
  - Mechanisms
  - Frameworks

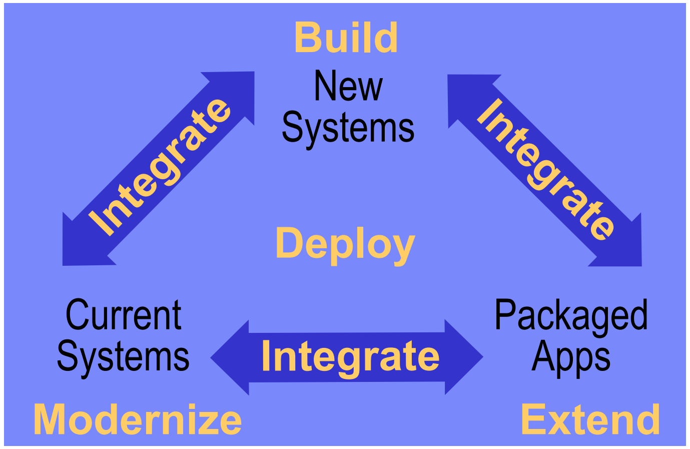
http://www.hillside.net & http://www.booch.com/architecture

The Architecture Of Development Environments



The Role Of Development Tools

- Tools exist to
  - Automate the creation and transformation of executable artifacts
  - Automate the creation and transformation of supporting artifacts
  - Analyze, reason about, explore within, and trace among all such artifacts
  - Automate and reduce the tedium associated with the processes associated with creating and transforming all such artifacts
- Tools may be classified as for
  - Heavy lifting
  - Visualizing
  - Measuring
  - Tracing
  - Polishing
  - Communicating
  - Daily hygiene



The Developer Experience

- The developer experience is defined by the conceptual environment formed by the surrounding toolset
- Such an environment should be
  - Sufficient
  - Complete
  - Resilient
  - Consistent
  - Simple

Developer Experience

- CLE -> IDE -> XDE - CDE
  - Development as
- The virtual project space
  - Addressing the needs of the team distributed in time and in space
- One hundred small things
  - Assets with CM
  - Presence and communication
  - Shared tools
- Experience from the open-source community
  - Sourceforge
  - Eclipse

An ideal environment provides a frictionless surface for development tuned to the specific concerns of its users

Development Environment Epochs

- 1945 – 1960 Early tools
- 1960 – 1980 Command Line Environments (CLE)
- 1980 – 2000 Integrated Development Environments (IDE)
- 2000 – 2004 eXtended Development Environments (XDE)
- 2005 - Collaborative Development Environments (CDE)

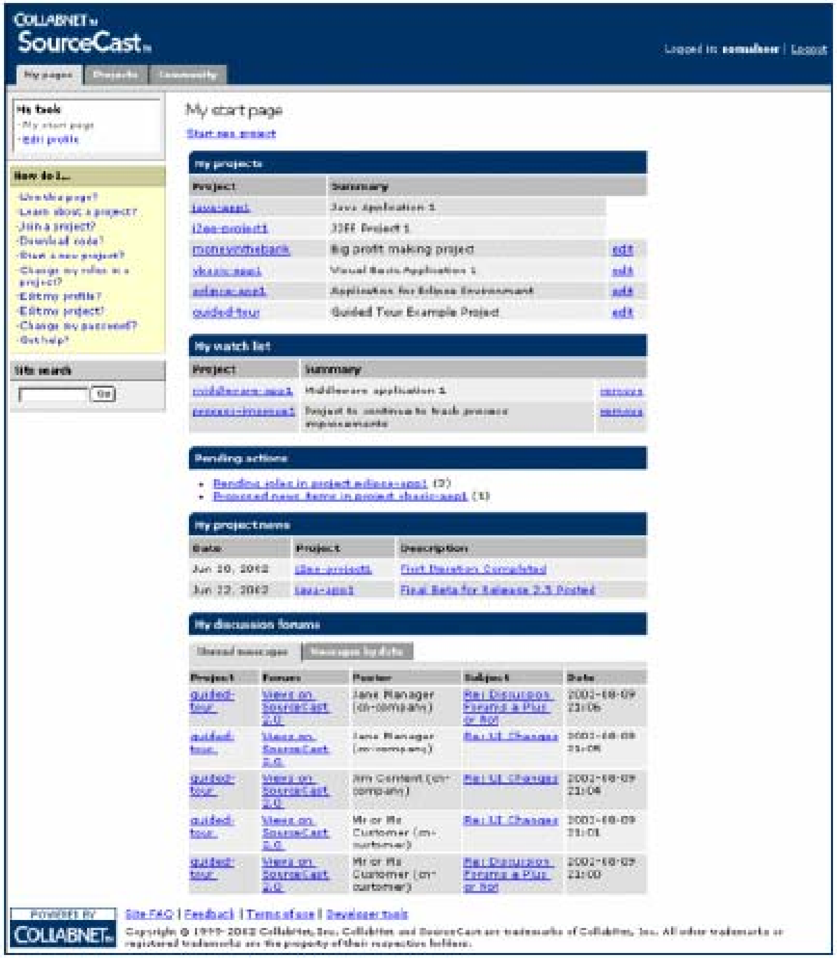
CDE: Focus And Features

- Focus
  - Run programs on distributed systems
- Users
  - Analysts, architects, developers, testers
  - Graphics and multimedia designers
  - Network and hardware engineers
  - Domain experts
- Target
  - Families of programs on native and hosted machines
  - Tools addressing the human factor
- Features (same as for XDE plus)
  - Virtualization of the team
  - IM, presence, web meetings, discussions, archives

CDE: Examples

- BuildTopia
- SourceForge
- SourceCast

Collab.net SourceCast (present)



COLLABNET SourceCast

Logged in: username | Logout

My pages | Projects | Community

My tools
- My start page
- Edit profile

How do I...
- Use this page?
- Learn about a project?
- Join a project?
- Download code?
- Start a new project?
- Change my roles in a project?
- Edit my profile?
- Edit my project?
- Change my password?
- Get help?

Site search [ ] Go

My start page

Start new project

My projects

Project	Summary	
java-app1	Java Application 1	
j2ee-project1	J2EE Project 1	
moneyonthebank	Big profit making project	edit
vbasic-app1	Visual Basic Application 1	edit
eclipse-app1	Application for Eclipse Environment	edit
guided-tour	Guided Tour Example Project	edit

My watch list

Project	Summary	
middleware-app1	Middleware application 1	remove
process-improvement	Project to continue to track process improvements	remove

Pending actions

- Pending roles in project eclipse-app1 (2)
- Proposed news items in project vbasic-app1 (1)

My project news

Date	Project	Description
Jun 10, 2002	j2ee-project1	First Iteration Completed
Jun 12, 2002	java-app1	Final Beta for Release 2.5 Posted

My discussion forums

Thread messages | Messages by date

Project	Forum	Poster	Subject	Date
guided-tour	Views on SourceCast 2.0	Jane Manager (in-company)	Re: Discussion Forums a Plus or Not	2002-08-09 23:06
guided-tour	Views on SourceCast 2.0	Jane Manager (in-company)	Re: UI Changes	2002-08-09 23:05
guided-tour	Views on SourceCast 2.0	Jim Content (in-company)	Re: UI Changes	2002-08-09 23:04
guided-tour	Views on SourceCast 2.0	Mr or Ms Customer (in-customer)	Re: UI Changes	2002-08-09 23:01
guided-tour	Views on SourceCast 2.0	Mr or Ms Customer (in-customer)	Re: Discussion Forums a Plus or Not	2002-08-09 23:00

POWERED BY COLLABNET

Site FAQ | Feedback | Terms of use | Developer tools

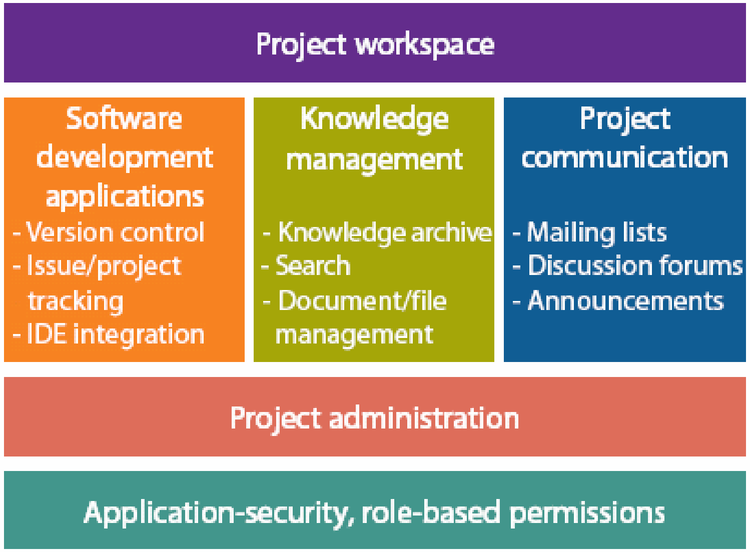
Copyright © 1999-2002 CollabNet, Inc. CollabNet and SourceCast are trademarks of CollabNet, Inc. All other trademarks or registered trademarks are the property of their respective holders.

Languages: various

Host: Web

Target: various

- Global development
- ASP



Project workspace

Software development applications
- Version control
- Issue/project tracking
- IDE integration

Knowledge management
- Knowledge archive
- Search
- Document/file management

Project communication
- Mailing lists
- Discussion forums
- Announcements

Project administration

Application-security, role-based permissions

www.collab.net

CDE: Forces

- Virtualization of the underlying hardware
  - Blurring of hardware and software
- Advances in domain-specific languages
- Economic and business pressure to develop more complex programs
  - Systems of systems
  - Separation of concerns

Collaborative Development Environment (CDE)

- A CDE is a virtual place where the stakeholders of a project - even if separated by time or space - can meet, share, brainstorm, discuss, reason about, negotiate, record, and generally labor togetehr to carry out some task.
- CDEs have emerged in a number of disciplines to address the social and technical problems involving distributed teams whose members must collaborate to solve some problem.
- Collaboration has always been an essential part of the fabric of the Internet
  - Email
  - Instant messaging
  - Chat rooms
  - Discussion groups
  - Wikis

CDE Properties

- A CDE is not a very unique or special thing
  - There are a hundred small things that already exist and that can be combined to form a virtual space
- The purpose of a CDE is to create a frictionless surface for development by eliminating of automating many of the daily, non-creative activities of the individual and the team and by providing mechanisms that encourage creative, healthy, and high-bandwidth modes of communication among a project's stakeholders.
- And IDE focuses on the productivity of the individual developer; a CDE focuses on the productivity of the team.

Infrastructure

- Blogs
  - information sites
- Mailing lists
  - small groups with a common purpose
- Message boards
  - Asking and answering questions
- Chat rooms
  - Holding scheduled events, real time discussion, back channel communication, hanging out
- Whiteboards
  - Brainstorming, communicating, discussing
- Net meetings
  - One-on-one discussions
- Portals
  - Communities of practice
- Wikis

Social Networking

- Slashdot
- Groove
- Sharepoint
- MySpace
- Facebook
- LinkedIn

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

CDE Features

- Instant messaging
- Virtual meeting room
- Application sharing
- Centralized information management
- Searching and indexing
- Configuration control of shared artifacts
- Co-browsing
- Electronic document routing and workflow
- Calendaring and scheduling
- Online event notification
- Project resource profiling
- Whiteboards

CDE Features

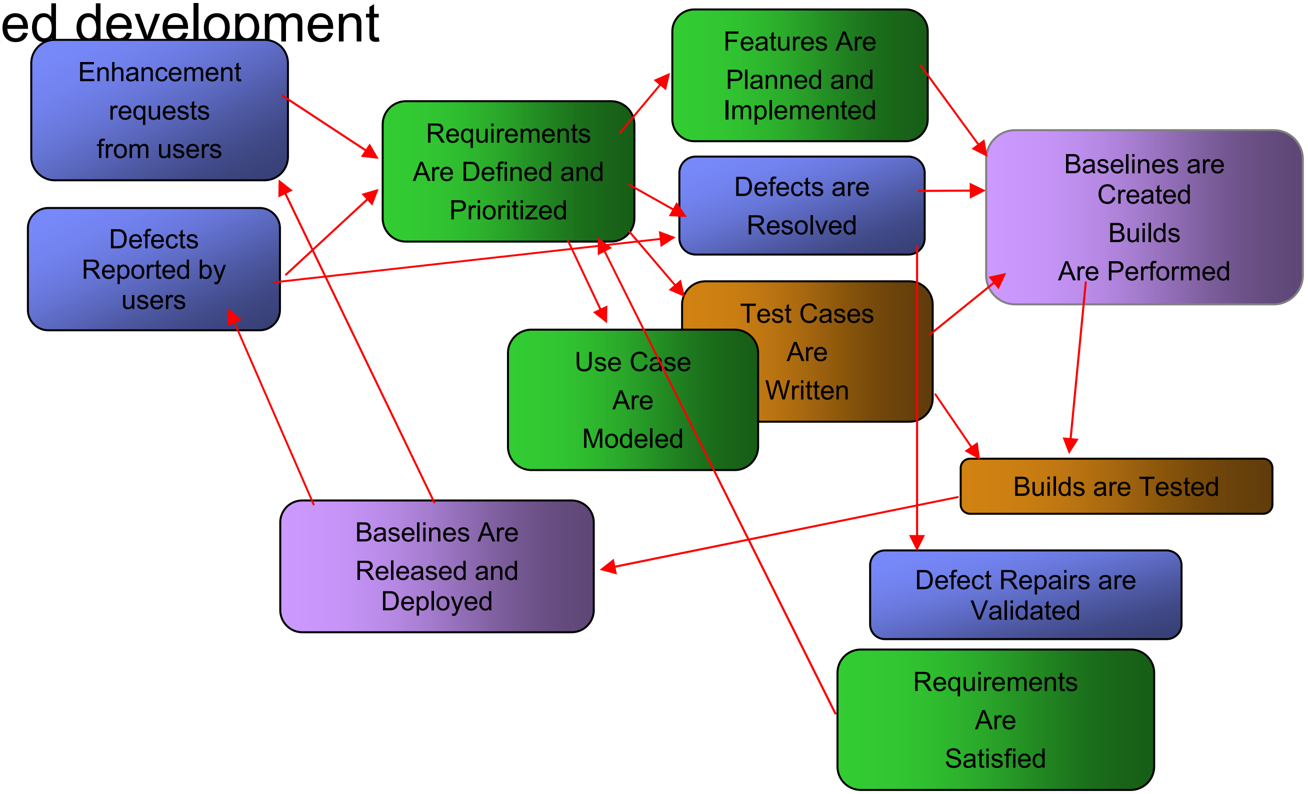
- Team member presence
- Tools for connected/disconnected use
- Threaded discussions
- Access to personal and project blogs
- Project dashboards and metrics
- Self-publication of content
- Self-administation of projects
- Lightweight peer-to-peer conferencing
- Lightweight group conferencing
- Non-intrusive auditing of changes
- Multiple levels of information visibiliyt
- Personalization of content
- Seamless access to tools that manipulate a project’s artifacts
- Multiple points of entry

CDE Out-Of-The Box Features

- Systems that break the constraints of single- or double-monitor per developer
- The use of non-human agents
- The exploitation of virtual worlds

Disruptive Technologies

- Deep semantic environments
- Pattern-centric development
- Aspect-oriented development


Enhancement requests from users
Defects Reported by users
Requirements Are Defined and Prioritized
Features Are Planned and Implemented
Defects are Resolved
Baselines are Created Builds Are Performed
Test Cases Are Written
Use Case Are Modeled
Builds are Tested
Baselines Are Released and Deployed
Defect Repairs are Validated
Requirements Are Satisfied

Disruptive Technologies

- Pervasive displays
- Telepresence

www.asktog.com/starfire

Disruptive Technologies

- Virtualization

Conclusion

- The world runs on software
  - Innovation
  - Economization
- The fundamental of software engineering never go out of style
- It is a privilege and a responsibility to be a software professional
  - Live your passion

Thank
You