Software development in HEP: a critical look

Federico Carminati
CERN – Geneva
ACAT10, February 22, 2010

- Physicists have always used computers
  - They invented them!
- The programs of the LHC era are of unprecedented complexity
  - Measured in units of 10^6 lines of code (MLOC)
  - Communities are very large (ATLAS > 2000 physicists and engineers)
- Failure to develop appropriate programs would jeopardise the extraction of the physics from the data
- … i.e. it would ultimately waste multi-million dollars investments in hardware and thousands of man years of highly qualified efforts

Developing software for HEP

- Physicists have always used computers
  - They invented them!
- The programs of the LHC era are of unprecedented complexity
  - Measured in units of 10^6 lines of code (MLOC)
  - Communities are very large (ATLAS > 2000 physicists and engineers)
- Failure to develop appropriate programs would jeopardise the extraction of the physics from the data
- … i.e. it would ultimately waste multi-million dollars investments in hardware and thousands of man years of highly qualified efforts

- In the LEP era the code was 90% written in FORTRAN
  - ~10 instructions!
  - The standard is 50 pages
- In the LHC era the code is written in many cooperating languages, the main one is C++
  - O(100) instructions
  - “Nobody understands C++ completely” (B.Stroustrup)
  - The standard is 700 pages
- Several new languages have been emerging with an uncertain future
  - C#, Java, Perl, Python, php…
- The Web world adds a new dimension to computing
- Not to talk about GRID…

The code

- In the LEP era the code was 90% written in FORTRAN
  - ~10 instructions!
  - The standard is 50 pages
- In the LHC era the code is written in many cooperating languages, the main one is C++
  - O(100) instructions
  - "Nobody understands C++ completely" (B.Stroustrup)
  - The standard is 700 pages
- Several new languages have been emerging with an uncertain future
  - C#, Java, Perl, Python, php…
- The Web world adds a new dimension to computing
- Not to talk about GRID…

- Physicists are both developers and users
- The community is very heterogeneous
  - From very expert analysts to occasional programmers
  - From 5% to 100% of time devoted to computing
- The community is very sparse
  - The communication problem is serious when developing large integrated systems
- People come and go with a very high rate
  - Programs have to be maintained by people who did not develop them
  - Young physicists need to acquire knowledge that they can use in their careers (also outside physics)
- The physicists have no strict hierarchical structure in an experiment

The people

- Physicists are both developers and users
- The community is very heterogeneous
  - From very expert analysts to occasional programmers
  - From 5% to 100% of time devoted to computing
- The community is very sparse
  - The communication problem is serious when developing large integrated systems
- People come and go with a very high rate
  - Programs have to be maintained by people who did not develop them
  - Young physicists need to acquire knowledge that they can use in their careers (also outside physics)
- The physicists have no strict hierarchical structure in an experiment

- In this complex and high-risk environment it seems natural to ask for help to those who make a living solving similar problems
- This is where the physicist meets the computer scientist
- But the interaction has been far from a honeymoon…
- … and the neighbour’s grass just looked greener

HEP & SE

- In this complex and high-risk environment it seems natural to ask for help to those who make a living solving similar problems
- This is where the physicist meets the computer scientist
- But the interaction has been far from a honeymoon…
- … and the neighbour's grass just looked greener

- Software Engineering is as old as software itself
  - H.D. Benington, "Production of Large Computer Programs", Proceedings, ONR Symposium, June 1956
  - F.L. Bauer, 1968, NATO conference

    "The whole trouble comes from the fact that there is so much tinkering with software. It is not made in a clean fabrication process, which it should be. What we need, is software engineering."

  - F.L. Bauer. Software Engineering. Information Processing 71, 1972

    "The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines."

Software, software crisis and SE

- Software Engineering is as old as software itself
  - H.D. Benington, "Production of Large Computer Programs", Proceedings, ONR Symposium, June 1956
  - F.L. Bauer, 1968, NATO conference

    "The whole trouble comes from the fact that there is so much tinkering with software. It is not made in a clean fabrication process, which it should be. What we need, is software engineering."
  - F.L. Bauer. Software Engineering. Information Processing 71, 1972

    "The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines."

- The software crisis comes from the failure of large software projects to meet their goals within budged and schedule
- Major worry of managers is not
  - Will the software work?
- But rather
  - Will the development finish within time and budget?
  - … or rather within which time and budget …
- SE has been proposed to solve the Software Crisis
  - More a goal than a definition!
  - A wild assumption on how engineers work
  - Can’t build it like a bridge if it ain’t a bridge

Software, software crisis and SE

- The software crisis comes from the failure of large software projects to meet their goals within budged and schedule
- Major worry of managers is not
  - Will the software work?
- But rather
  - Will the development finish within time and budget?
  - … or rather within which time and budget …
- SE has been proposed to solve the Software Crisis
  - More a goal than a definition!
  - A wild assumption on how engineers work
  - Can't build it like a bridge if it ain't a bridge

- Cost and Budget Overruns
  - Classic example is the OS/360 operating system. 10 years and 1000 programmers. F.Brooks claims in Mythical Man Month that he made a multi-million dollar mistake by not developing a coherent architecture before starting
- Property Damage
  - Identity stealing from hackers costs time, money, and reputations
- Life and Death
  - Some embedded systems used in radiotherapy machines failed so catastrophically that they administered lethal doses of radiation to patients

Software defects are dangerous

- Cost and Budget Overruns
  - Classic example is the OS/360 operating system. 10 years and 1000 programmers. F.Brooks claims in Mythical Man Month that he made a multi-million dollar mistake by not developing a coherent architecture before starting
- Property Damage
  - Identity stealing from hackers costs time, money, and reputations
- Life and Death
  - Some embedded systems used in radiotherapy machines failed so catastrophically that they administered lethal doses of radiation to patients

- Tools
  - Structured programming, object-oriented programming, CASE tools, Ada, Java, documentation, standards, and Unified Modeling Language were touted as silver bullets
- Discipline
  - The software crisis was due to the lack of discipline of programmers
- Formal methods
  - Apply formal engineering methodologies to software development, to make production of software as predictable as other branches of engineering, proving all programs correct
- Process
  - Processes and methodologies like the Capability Maturity Model
- Professionalism
  - This led to work on a code of ethics, licenses, and professionalism

Looking for silver bullets

- Tools
  - Structured programming, object-oriented programming, CASE tools, Ada, Java, documentation, standards, and Unified Modeling Language were touted as silver bullets
- Discipline
  - The software crisis was due to the lack of discipline of programmers
- Formal methods
  - Apply formal engineering methodologies to software development, to make production of software as predictable as other branches of engineering, proving all programs correct
- Process
  - Processes and methodologies like the Capability Maturity Model
- Professionalism
  - This led to work on a code of ethics, licenses, and professionalism

- Many of the early programmers were women
- As SE settled in as a discipline, programming became a male-only discipline
- Only very slowly women are finding back their place in programming

SE men and women…

- Many of the early programmers were women
- As SE settled in as a discipline, programming became a male-only discipline
- Only very slowly women are finding back their place in programming

SE men and women…

- Many of the early programmers were women
- As SE settled in as a discipline, programming became a male-only discipline
- Only very slowly women are finding back their place in programming

SE men and women…

- Many of the early programmers were women
- As SE settled in as a discipline, programming became a male-only discipline
- Only very slowly women are finding back their place in programming

1945: Grace Hopper discovers the first computer bug

SE men and women…

- Many of the early programmers were women
- As SE settled in as a discipline, programming became a male-only discipline
- Only very slowly women are finding back their place in programming

1945: Grace Hopper discovers the first computer bug

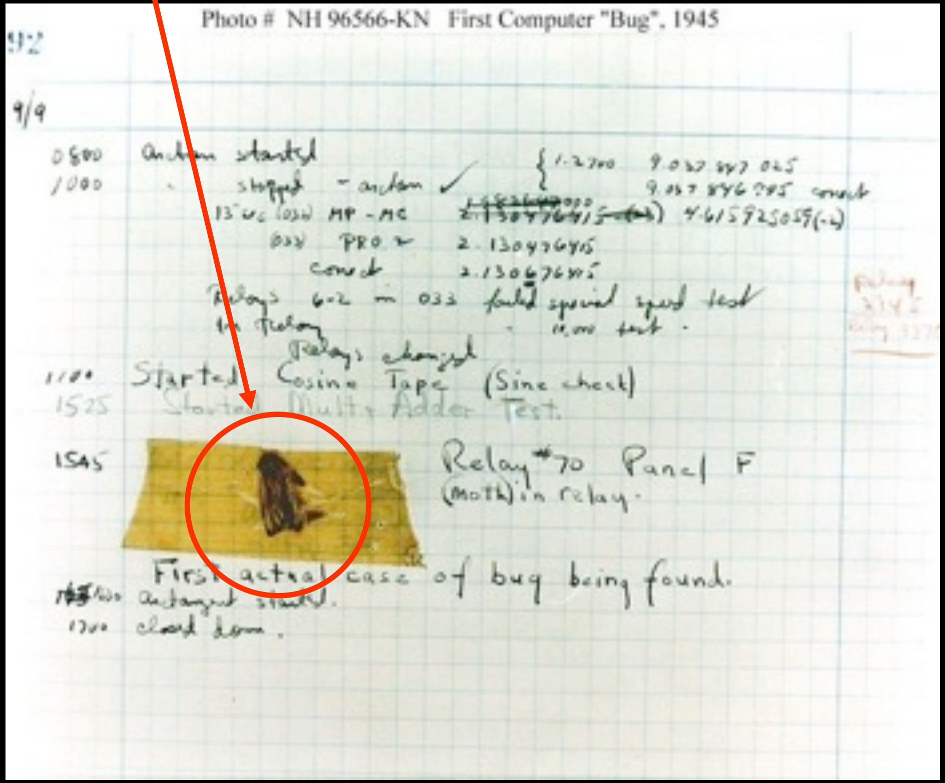


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Anctan started
1000 " stopped - anctan ✓ {1.2700 9.037 847 025
9.037 846 995 correct
13" UC (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 10.000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 antangent started.
1700 closed down.

Relay 2145
Relay 3370

SE men and women…

- Many of the early programmers were women
- As SE settled in as a discipline, programming became a male-only discipline
- Only very slowly women are finding back their place in programming

1945: Grace Hopper discovers the first computer bug

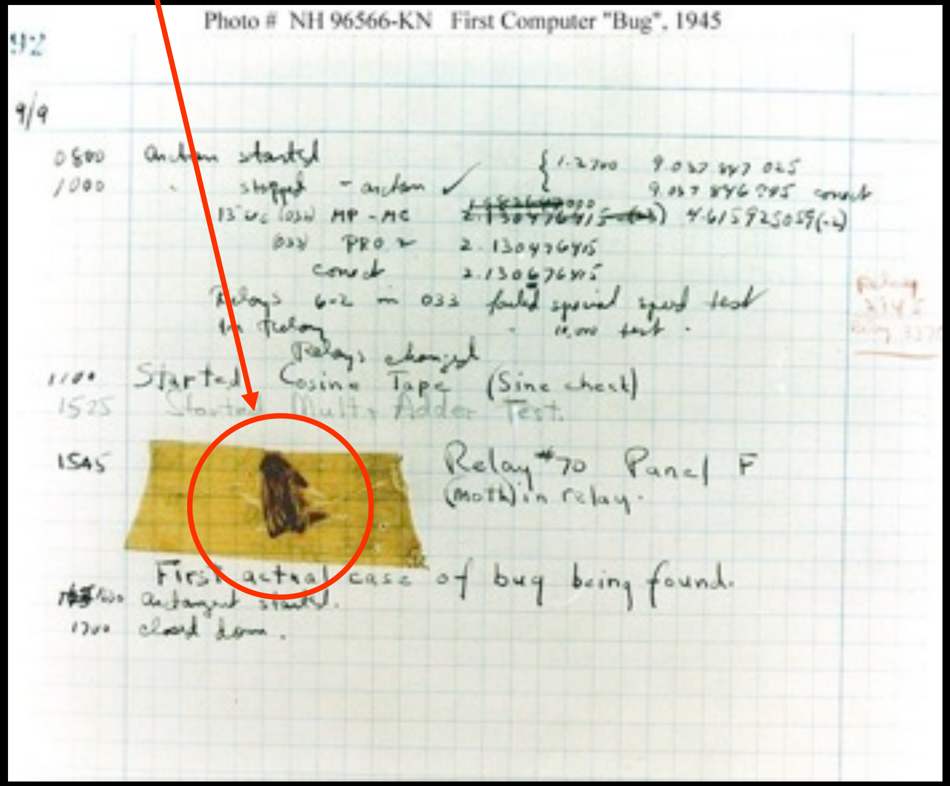


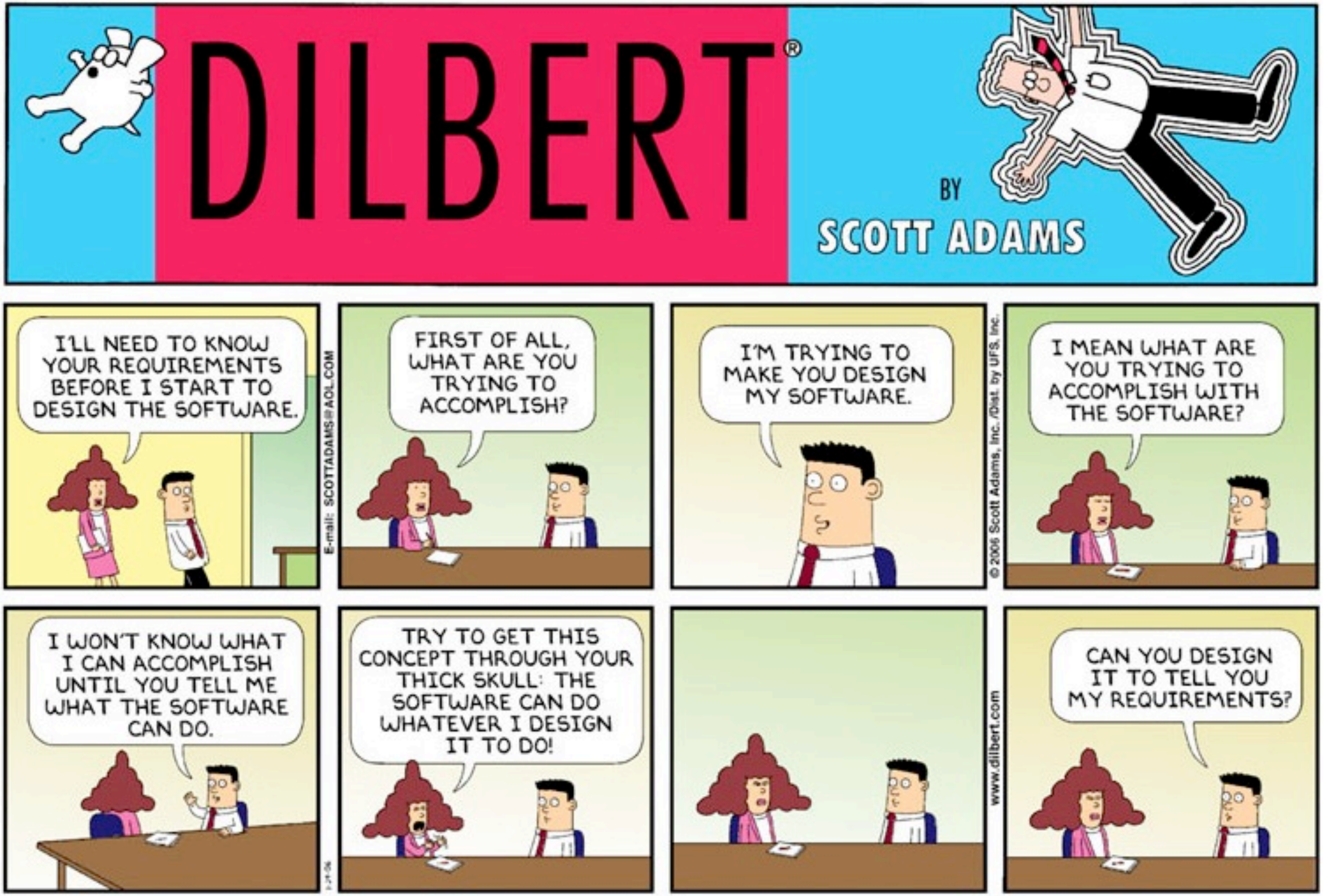
Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Arctan started
1000 " stopped - arctan ✓ {1.2700 9.037 847 025
13" uc (032) MP - MC ~~1.982647000~~ 9.037 846 995 correct
(033) PRO 2 ~~2.130476415~~ (-3) 4.615925059(-2)
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 10,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

Relay 2145
Relay 3370


DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
E-mail: SCOTTADAMS@AOL.COM
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
www.dilbert.com
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?

- Software is opposed to hardware because it should be flexible
- Yet the reason of the failure of software process is often identified in the changes intervening during the development
- The heart of SE is the limitation of the impact of changes
  - Changes are avoided by a better design
  - A better design is obtained by exhaustive requirements
  - The more complete the design, the less the changes, the smaller the cost of software

SE crisis

- Software is opposed to hardware because it should be flexible
- Yet the reason of the failure of software process is often identified in the changes intervening during the development
- The heart of SE is the limitation of the impact of changes
  - Changes are avoided by a better design
  - A better design is obtained by exhaustive requirements
  - The more complete the design, the less the changes, the smaller the cost of software

- SE splits the process in a sequence of controllable phases
  - Analysis, design, implementation, testing, maintenance…
  - A hierarchy and a roadmap to navigate among them
- Still software projects continue to fail: the SE crisis
- Orthodox SE diagnosis is: not enough SE was applied
  - More discipline and more strict observance of the rules
  - Too process kill the process, projects keeps failing
- Modern SE tries to find a different answer

SE crisis

- SE splits the process in a sequence of controllable phases
  - Analysis, design, implementation, testing, maintenance…
  - A hierarchy and a roadmap to navigate among them
- Still software projects continue to fail: the SE crisis
- Orthodox SE diagnosis is: not enough SE was applied
  - More discipline and more strict observance of the rules
  - Too process kill the process, projects keeps failing
- Modern SE tries to find a different answer

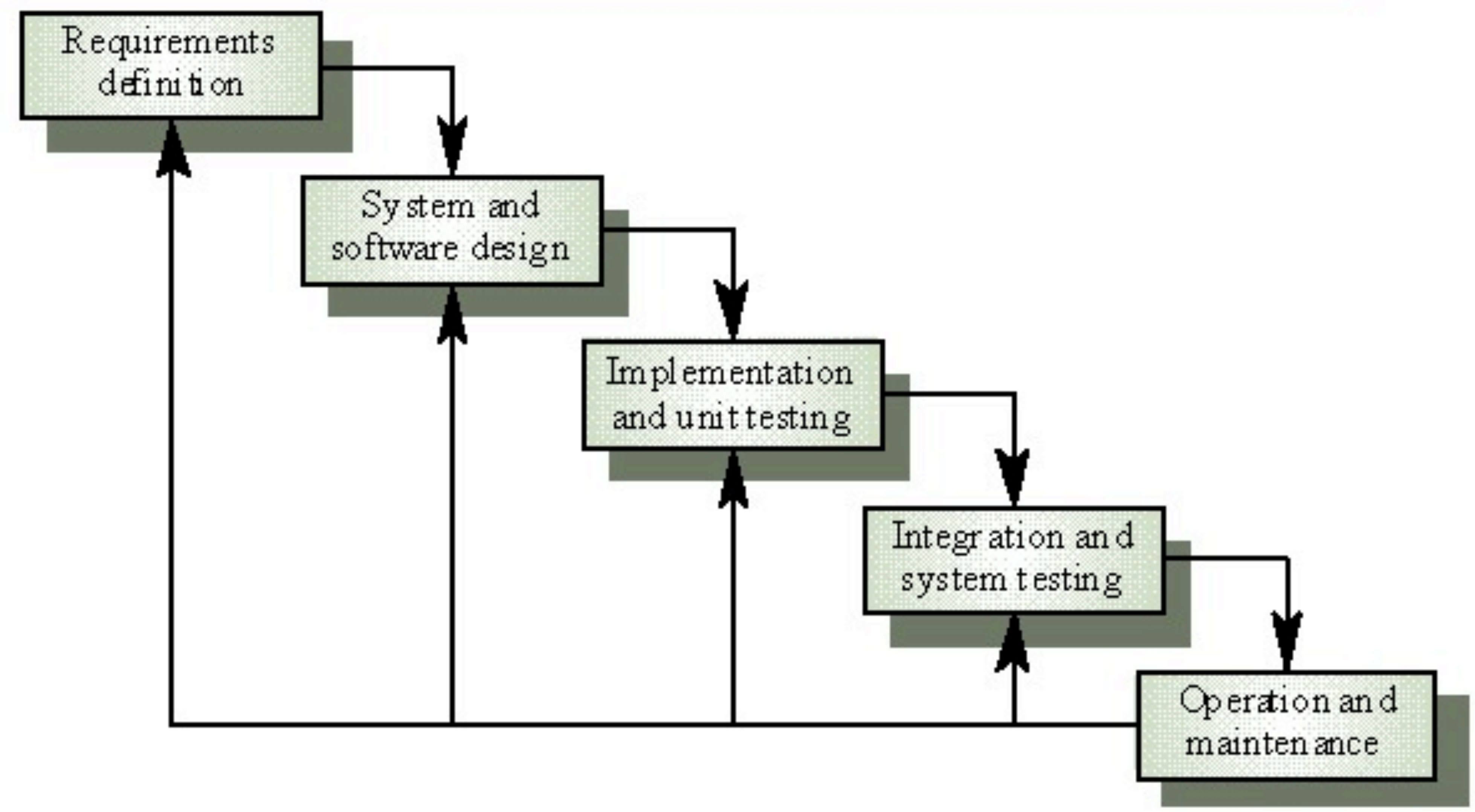
- Many formal paper documents
- Very detailed design models, difficult to read and understand
- Formal document ownership
- Distinct developer roles
- Communications through documents
- Formal process to follow
- HCP are suited for big projects, with stable requirements
  - The time elapsed from requirement gathering to start coding may be as long as 1-2 years
- In the e-business era (and in science!) projects are characterized by
  - High speed, change and uncertainty

High Ceremony Process

- Many formal paper documents
- Very detailed design models, difficult to read and understand
- Formal document ownership
- Distinct developer roles
- Communications through documents
- Formal process to follow
- HCP are suited for big projects, with stable requirements
  - The time elapsed from requirement gathering to start coding may be as long as 1-2 years
- In the e-business era (and in science!) projects are characterized by
  - High speed, change and uncertainty

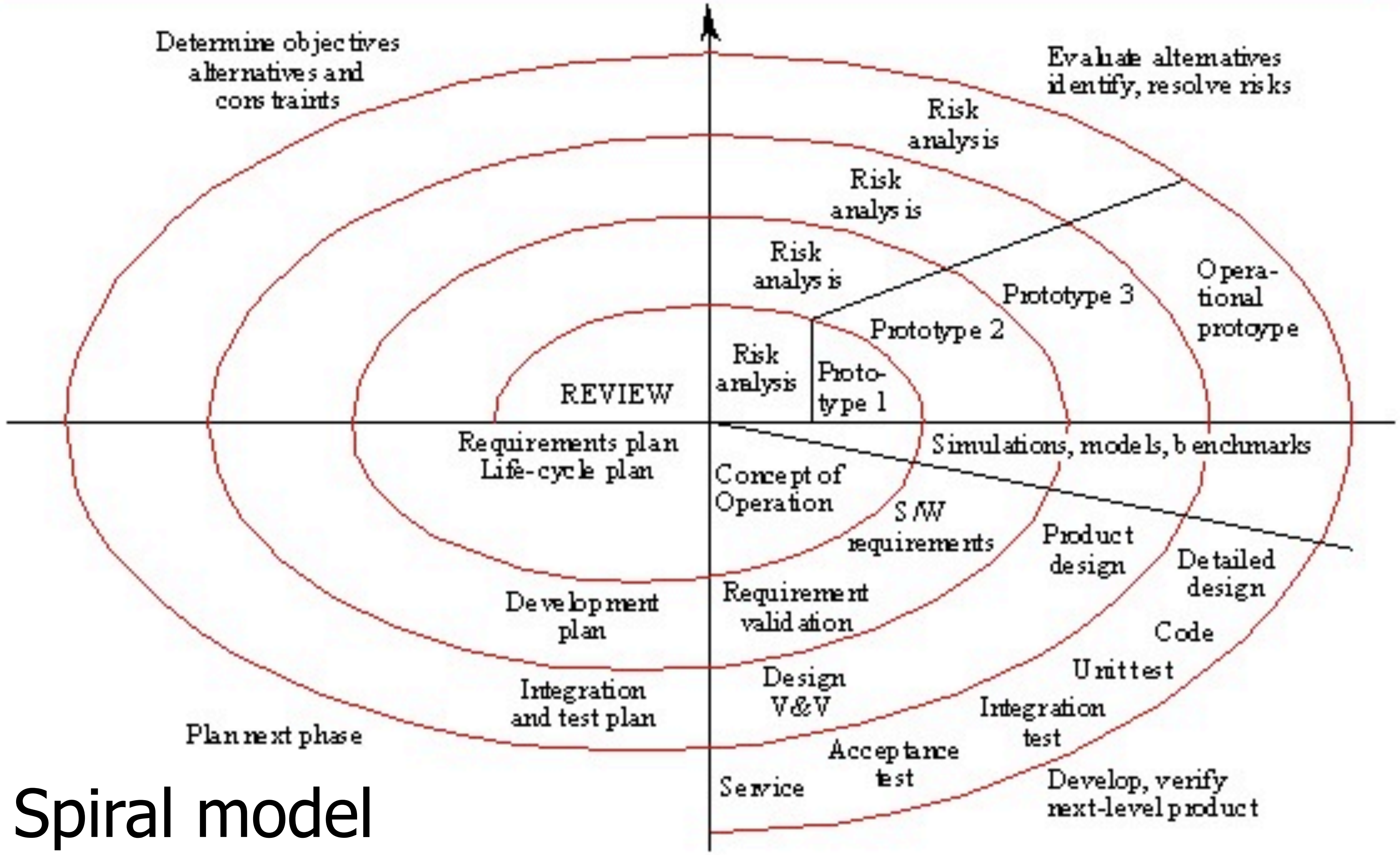
High Ceremony Process

- Many formal paper documents
- Very detailed design models, difficult to read and understand
- Formal document ownership
- Distinct developer roles
- Communi
- Formal pr
- HCP are s
  - The time
    long as I
- In the e-bu
  by
  - High spe


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

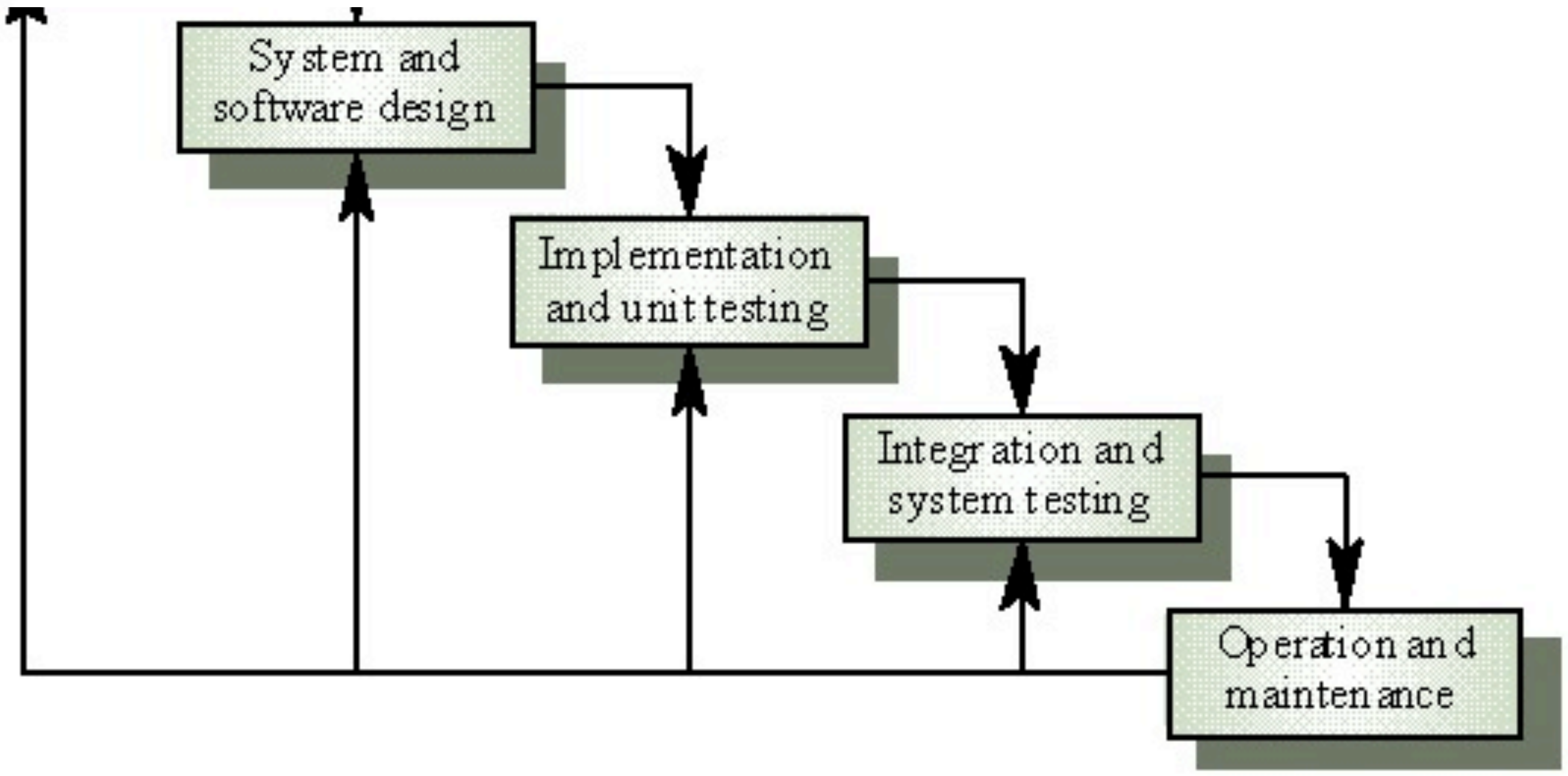
ess

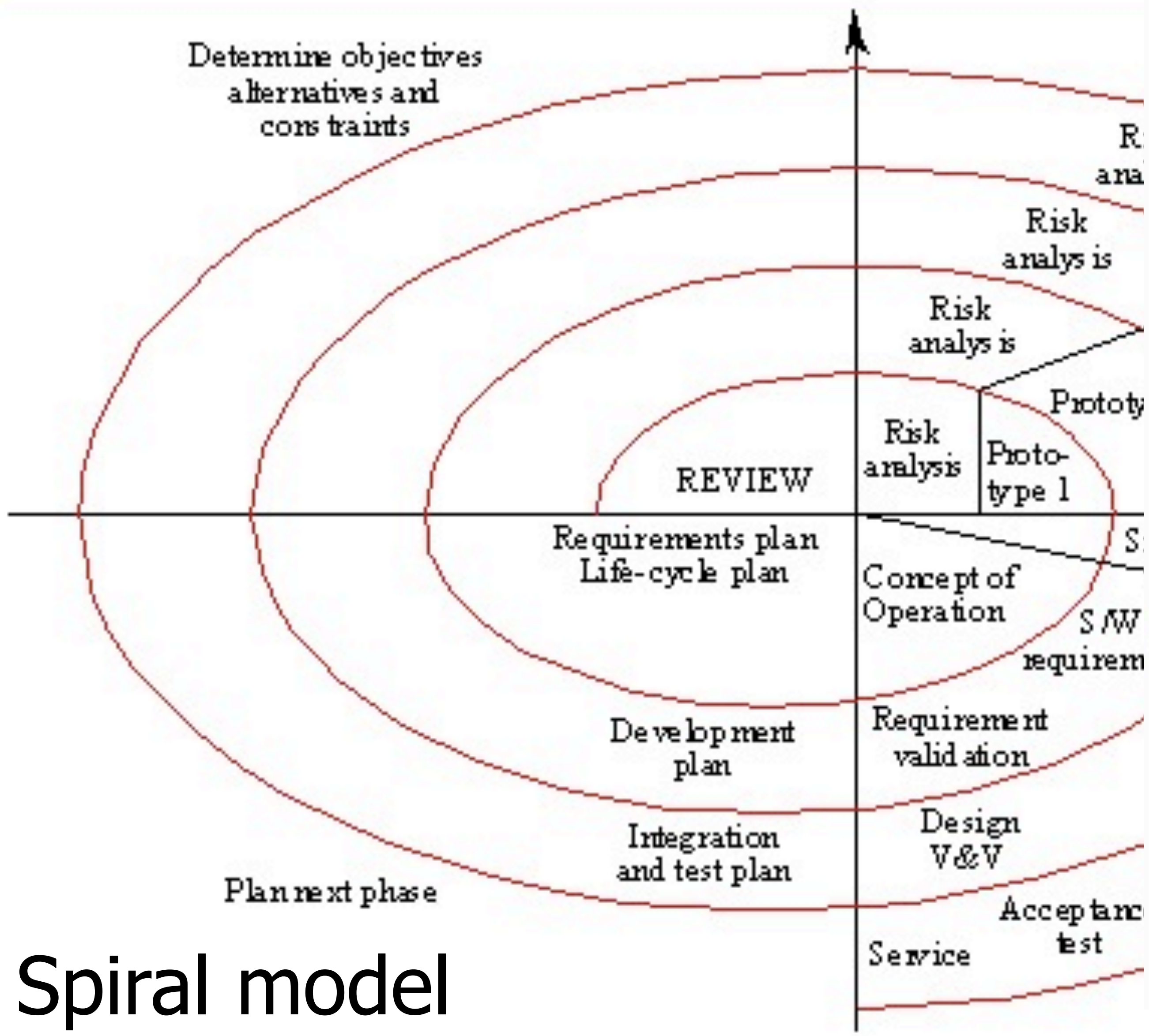
understand



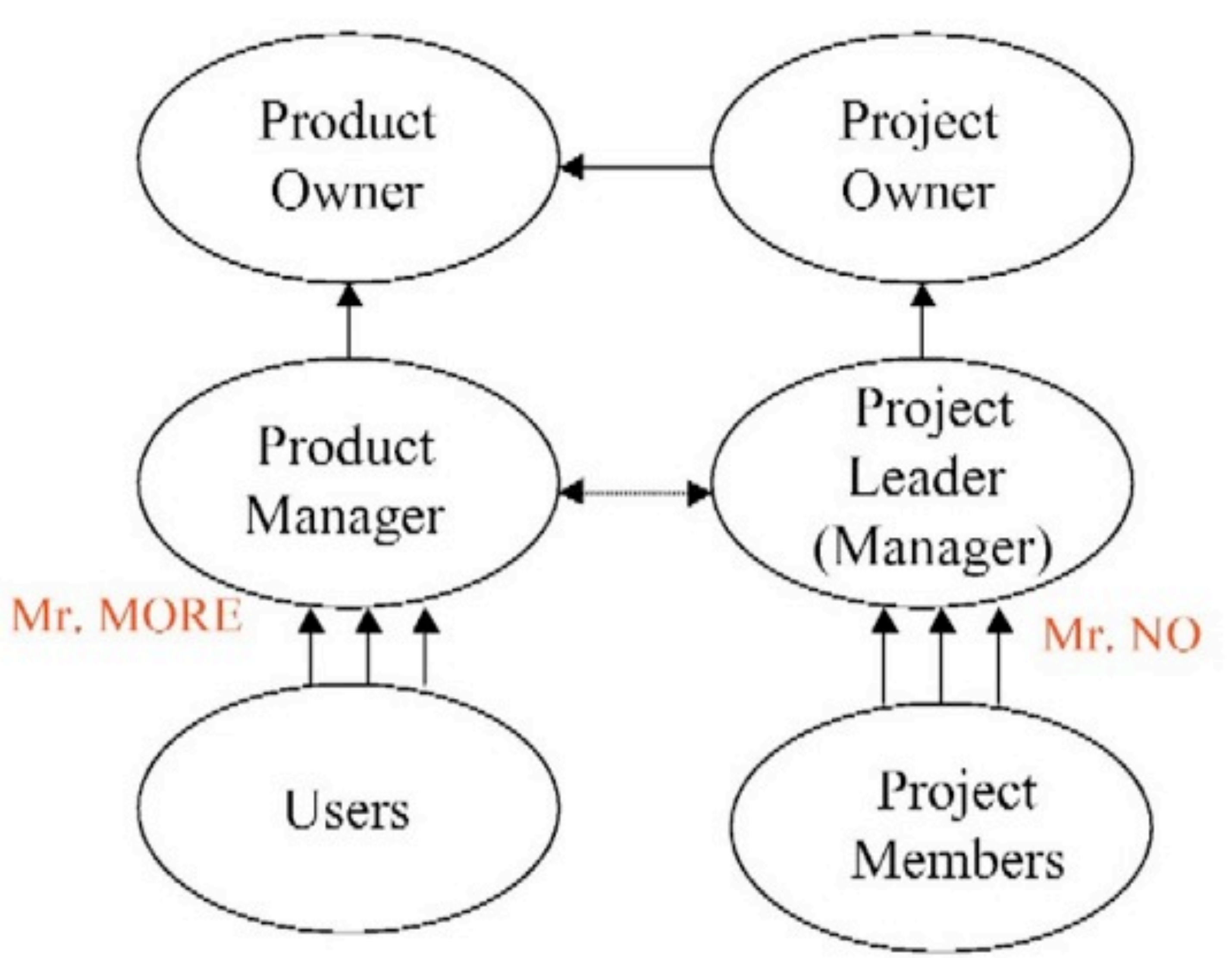
Spiral model

- HCP are s
    - The time long as
- In the e-bu by
    - High spe

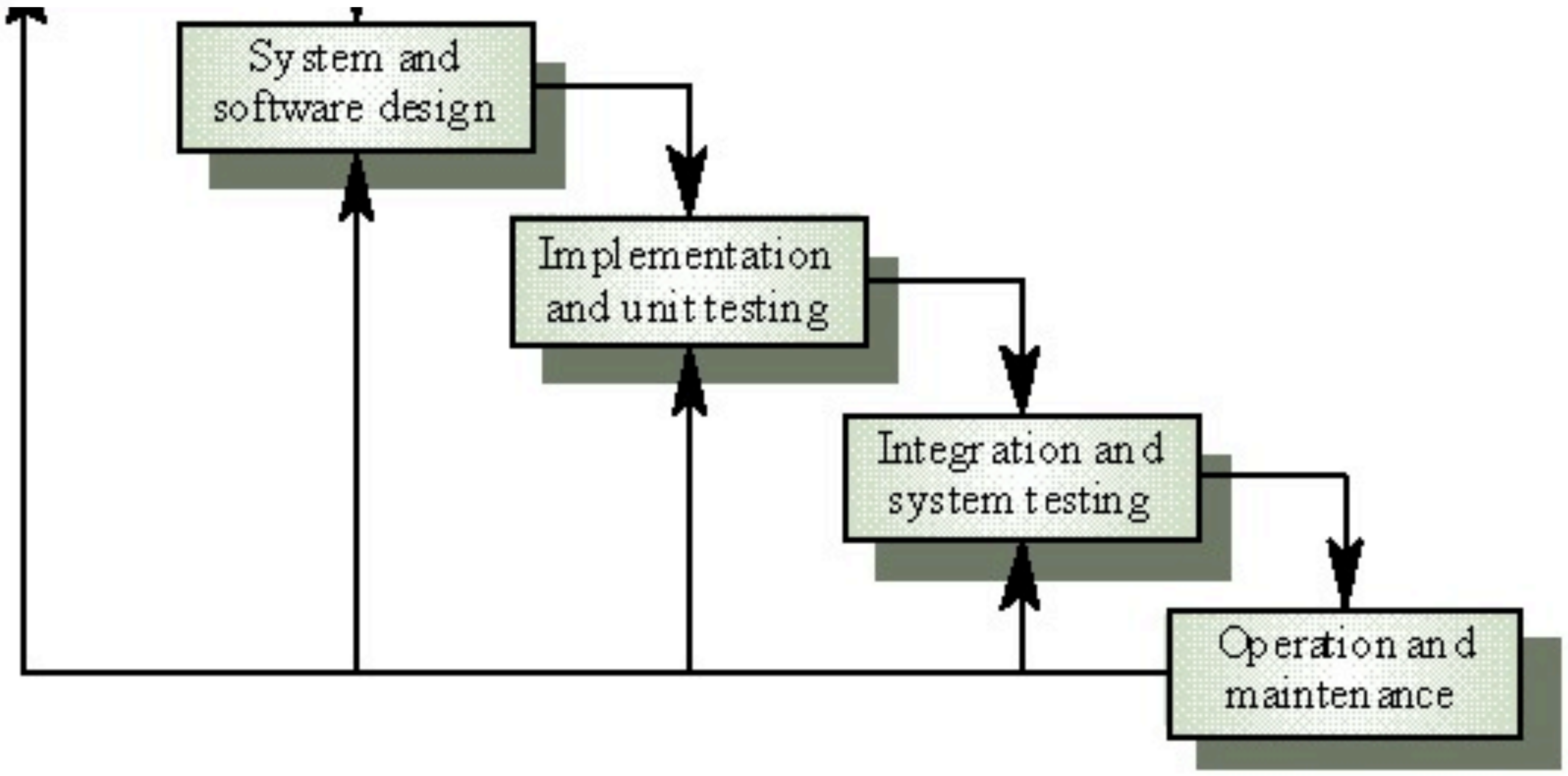

System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance



Spiral model



- HCP are s
  - The time
    long as l
- In the e-bu
  by
  - High spe



- A crisis that lasts 40 years is not a crisis, but a stationary state
- From mid 80's to mid 90's SE has been looking for the silver bullet
- From mid 90's onward came the realisation that developing working software was just very hard
- SE has given us a much deeper understanding of the process of software development
- But we still miss a "magic solution"

Did SE fail?

- A crisis that lasts 40 years is not a crisis, but a stationary state
- From mid 80's to mid 90's SE has been looking for the silver bullet
- From mid 90's onward came the realisation that developing working software was just very hard
- SE has given us a much deeper understanding of the process of software development
- But we still miss a "magic solution"

A Software Engineer’s nightmare

- A Soft Engineer’s view of HEP
  - ✓HEP has a complicated problem to solve
  - ✓SE is good for complicated problems
  - ✓These physicists cannot even spell SE correctly
  - ✓A bit of SE can do wonder
    - Or, in its weaker form, it should work at least here!
  - ✓Let’s introduce SE in HEP!
- A scheduled success!
- What can possibly go wrong?

- HEP has tried all new SE technologies, tools and formalisms
  - Yourdon's SASD, ER, Booch's OOADA, Rambaugh's OMT, Shlaer-Mellor's OL, ESA's PSS-05, UML, USDP
  - ADAMO, I-Logix Statemate, OMW, OMTool, StP, Rational Rose, ObjecTime, Together
- All have raised interest and then fallen into oblivion
- The OO projects started in '94 have used extensively SE
  - GEANT4 was late in entering production (8 years)
  - Spider has been cancelled
  - LHC++/ANAPHE/AIDA never could replace PAW / CERNLIB and have finally been abandoned
- The winning product (root!) never claimed any use of SE
- Did traditional SE fail to deliver?

HEP software & Software Engineering

- HEP has tried all new SE technologies, tools and formalisms
  - Yourdon's SASD, ER, Booch's OOADA, Rambaugh's OMT, Shlaer-Mellor's OL, ESA's PSS-05, UML, USDP
  - ADAMO, I-Logix Statemate, OMW, OMTool, StP, Rational Rose, ObjecTime, Together
- All have raised interest and then fallen into oblivion
- The OO projects started in '94 have used extensively SE
  - GEANT4 was late in entering production (8 years)
  - Spider has been cancelled
  - LHC++/ANAPHE/AIDA never could replace PAW / CERNLIB and have finally been abandoned
- The winning product (root!) never claimed any use of SE
- Did traditional SE fail to deliver?

- The largest Grid in operation is the LCG Grid
  - This is a success that cannot be denied
- However Grid MW projects are grossly over budget, late and under expectations
- Yet they have put an enormous focus on the usage of proper (classic!) SE methods
- What went so wrong?

A case study – the Grid

- The largest Grid in operation is the LCG Grid
  - This is a success that cannot be denied
- However Grid MW projects are grossly over budget, late and under expectations
- Yet they have put an enormous focus on the usage of proper (classic!) SE methods
- What went so wrong?

- Insistence on complete requirements
  - But users never saw a Grid before!
- Ask the same question till you get the answer you want
  - “This is not a requirement…”
- Insist on one single line of development
  - Bureaucracy before creativity
- Develop incompatible versions of the same product
  - Ping-pong support
- Multiply “testbeds” without users
  - “The users will continue creating bugs hindering us from developing new code!”

Grid anti-patterns

- Insistence on complete requirements
  - But users never saw a Grid before!
- Ask the same question till you get the answer you want
  - “This is not a requirement…”
- Insist on one single line of development
  - Bureaucracy before creativity
- Develop incompatible versions of the same product
  - Ping-pong support
- Multiply “testbeds” without users
  - “The users will continue creating bugs hindering us from developing new code!”

- Underestimate the importance of stability, portability and backward compatibility
- “Don’t repeat the root mistake”
  - Beat the outsiders into submission immediately!
- Delay deadlines instead of descoping them
  - Release late and release seldom
- Shortcut complicated processes instead of simplifying them

Grid anti-patterns

- Underestimate the importance of stability, portability and backward compatibility
- “Don’t repeat the root mistake”
  - Beat the outsiders into submission immediately!
- Delay deadlines instead of descoping them
  - Release late and release seldom
- Shortcut complicated processes instead of simplifying them

- HEP software has been largely successful!
  - Experiments have not been hindered by software in their scientific goals
- CERNLIB (GEANT3, PAW, MINUIT) has been an astounding success
  - From small teams in close contact with experiments
  - In use for over 20 years
  - Ported to all architectures and OS that appeared
  - Reused by hundreds of experiments around the world
- The largest grid in operation is, after all, the LCG grid
- And yet we (as a community) have not used canonical SE
- Did we do something right?

HEP software: the facts

- HEP software has been largely successful!
  - Experiments have not been hindered by software in their scientific goals
- CERNLIB (GEANT3, PAW, MINUIT) has been an astounding success
  - From small teams in close contact with experiments
  - In use for over 20 years
  - Ported to all architectures and OS that appeared
  - Reused by hundreds of experiments around the world
- The largest grid in operation is, after all, the LCG grid
- And yet we (as a community) have not used canonical SE
- Did we do something right?

i.e. getting rid of the mantra "let's do it as they do it in industry…"

- Fuzzy & evolving requirements
  - If we knew what we are doing we would not call it research
- Bleeding edge technology
  - The boundary of what we do moves with technology
- Non-hierarchical social system
  - Roles of user, analyst, programmer etc are shared
  - Very little control on most of the (wo)man power
- Different assessment criteria
  - Performance evaluation is not based on revenues
  - We do not produce wealth, we spend it!
  - We produce knowledge, but this is not an engineering standard item

HEP Software, what's special?

i.e. getting rid of the mantra "let's do it as they do it in industry…"

- Fuzzy & evolving requirements
  - If we knew what we are doing we would not call it research
- Bleeding edge technology
  - The boundary of what we do moves with technology
- Non-hierarchical social system
  - Roles of user, analyst, programmer etc are shared
  - Very little control on most of the (wo)man power
- Different assessment criteria
  - Performance evaluation is not based on revenues
  - We do not produce wealth, we spend it!
  - We produce knowledge, but this is not an engineering standard item

- Traditional SE does not fit our environment
  - Only applicable when requirements are well understood
  - Our non-hierarchical structure does not match it
  - We do not have the extra (wo)man power for it
  - It introduces a semantic gap between its layers and the additional work of translating, mapping and navigating between them
- It acts on the process and not on the problem
  - It structures the activity constraining it to a limited region, with precisely defined interfaces
  - A Tayloristic organization of work, scarcely effective when the product is innovation and knowledge

Is SE any good for us?

- Traditional SE does not fit our environment
  - Only applicable when requirements are well understood
  - Our non-hierarchical structure does not match it
  - We do not have the extra (wo)man power for it
  - It introduces a semantic gap between its layers and the additional work of translating, mapping and navigating between them
- It acts on the process and not on the problem
  - It structures the activity constraining it to a limited region, with precisely defined interfaces
  - A Tayloristic organization of work, scarcely effective when the product is innovation and knowledge

"In my experience I often found plans useless, while planning was always invaluable."

D.Eisenhower

- Change is no accident, it is **the** element on which to plan
  - As such it must be an integral part of the software process
- Need to reconsider the economy of change
  - Initial design needs not to be complete or late changes bad
- Designing is still fundamental
  - It brings understanding of the goals and code quality and robustness
- However sticking to an out-of-date design would
  - Hinder evolution
  - Limit the functionality of the code
  - Waste effort on no-longer needed features
  - Increase time-to-market

Change, change, change

"In my experience I often found plans useless, while planning was always invaluable."
D.Eisenhower

- Change is no accident, it is **the** element on which to plan
  - As such it must be an integral part of the software process
- Need to reconsider the economy of change
  - Initial design needs not to be complete or late changes bad
- Designing is still fundamental
  - It brings understanding of the goals and code quality and robustness
- However sticking to an out-of-date design would
  - Hinder evolution
  - Limit the functionality of the code
  - Waste effort on no-longer needed features
  - Increase time-to-market

- Start with an initial common story
  - A shared goal felt as part of a community identity

    "We know what we want because we know what we need and what did not work in the past"
  - More precision would be an artefact and a waste of time
- Develop a (functional) prototype with the features that are felt to be more relevant by the community
  - The story becomes quickly a reality (short time-to-market)
  - Interested and motivated users use it for day-by-day work
  - Must master equilibrium between too few and too many users

How do we work?

(an idealised after-the-fact account of events)

- Start with an initial common story
  - A shared goal felt as part of a community identity

    > "We know what we want because we know what we need and what did not work in the past"

  - More precision would be an artefact and a waste of time
- Develop a (functional) prototype with the features that are felt to be more relevant by the community
  - The story becomes quickly a reality (short time-to-market)
  - Interested and motivated users use it for day-by-day work
  - Must master equilibrium between too few and too many users

- Developers (most of them users) work on the most important (i.e. demanded) features
  - Continuous feed-back provided by (local and remote) users
  - Coherence by the common ownership of the initial story
  - More and more users get on board as the system matures

How do we work?

(an idealised after-the-fact account of events)

- Developers (most of them users) work on the most important (i.e. demanded) features
  - Continuous feed-back provided by (local and remote) users
  - Coherence by the common ownership of the initial story
  - More and more users get on board as the system matures

- Users collectively own the system and contribute to it in line with the spirit of the initial common story
  - New versions come frequently and the development one is available
- Redesigns happen, even massive, without blocking the system
- Users tend to be vocal but loyal to the system
  - It is their system and it has to work, their needs are satisfied
- Most of the communication happens via e-mail
- Relations are driven by respect and collaborative spirit
  - CERNLIB from late 70's to early 90's and of ROOT since

How do we work?

(an idealised after-the-fact account of events)

- Users collectively own the system and contribute to it in line with the spirit of the initial common story
  - New versions come frequently and the development one is available
- Redesigns happen, even massive, without blocking the system
- Users tend to be vocal but loyal to the system
  - It is their system and it has to work, their needs are satisfied
- Most of the communication happens via e-mail
- Relations are driven by respect and collaborative spirit
  - CERNLIB from late 70's to early 90's and of ROOT since

- Modern SE tries to find short time-to-market solutions for rapidly changing
  - Requirements
  - User community
  - Hardware/OS base
  - Developer teams
- This is the norm for HEP
  - Once more we are today where IT will be tomorrow
- Modern SE seems to formalise and justify the conventions and rituals of HEP software
  - Minimise early planning, maximise feedback from users, manage change, not avoid it
- Can we gain something out of it?

Is there method to this madness?

- Modern SE tries to find short time-to-market solutions for rapidly changing
  - Requirements
  - User community
  - Hardware/OS base
  - Developer teams
- This is the norm for HEP
  - Once more we are today where IT will be tomorrow
- Modern SE seems to formalise and justify the conventions and rituals of HEP software
  - Minimise early planning, maximise feedback from users, manage change, not avoid it
- Can we gain something out of it?

- Famous article from E.Raymond on software development (1997)
  - Rapid prototyping
  - User feedback
  - Release early release often
- One of the first fundamental criticisms to the traditional software engineering

"Linux is subversive..."

The Cathedral and the Bazaar

http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

- Famous article from E.Raymond on software development (1997)
  - Rapid prototyping
  - User feedback
  - Release early release often
- One of the first fundamental criticisms to the traditional software engineering

“Linux is subversive…”

> Perfection is achieved not when everything that can be added is added, but when everything that can be removed is removed
>
> Michelangelo

- Simplicity is the most complicated thing to achieve in software development
- Simplicity should be the objective of continual planning and refactoring
- Simplicity should be part of the initial design

Simplicity, emergence and the like

> *Perfection is achieved not when everything that can be added is added, but when everything that can be removed is removed*
>
> *Michelangelo*

- Simplicity is the most complicated thing to achieve in software development
- Simplicity should be the objective of continual planning and refactoring
- Simplicity should be part of the initial design

- Free Redistribution
  - Do not throw away long-term gains in order for little short-term money. Avoid pressure for cooperators to defect
- Availability of source Code
  - You can't evolve programs without modifying them
- Permission of derived works
  - For rapid evolution to happen, people need to be able to experiment with and redistribute modifications
- Integrity of The Author's Source Code
  - Users should know who is responsible for the software. Authors should know what they support and protect their reputations
- No Discrimination Against Persons, Groups or Fields
  - Insure the maximum diversity of persons and groups contributing to open sources, allow all commercial users to join
- Distributable, non specific and non restrictive License
  - Avoid all “license traps”, let distributors chose their media and format

Open Source (more than just the code…)

"Live free or die"

- Free Redistribution
  - Do not throw away long-term gains in order for little short-term money. Avoid pressure for cooperators to defect
- Availability of source Code
  - You can't evolve programs without modifying them
- Permission of derived works
  - For rapid evolution to happen, people need to be able to experiment with and redistribute modifications
- Integrity of The Author's Source Code
  - Users should know who is responsible for the software. Authors should know what they support and protect their reputations
- No Discrimination Against Persons, Groups or Fields
  - Insure the maximum diversity of persons and groups contributing to open sources, allow all commercial users to join
- Distributable, non specific and non restrictive License
  - Avoid all "license traps", let distributors chose their media and format

MIT/X-Consortium License: truly “no strings attached”

“Revised” BSD License: MIT License + No Endorsement Clause

“Original” BSD License: Revised BSD License + Attribution Clause

Apache License: Original BSD License + No Use of “Apache” Name

Different "Open Source" Licenses

MIT/X-Consortium License: truly "no strings attached"

"Revised" BSD License: MIT License + No Endorsement Clause

"Original" BSD License: Revised BSD License + Attribution Clause

Apache License: Original BSD License + No Use of "Apache" Name

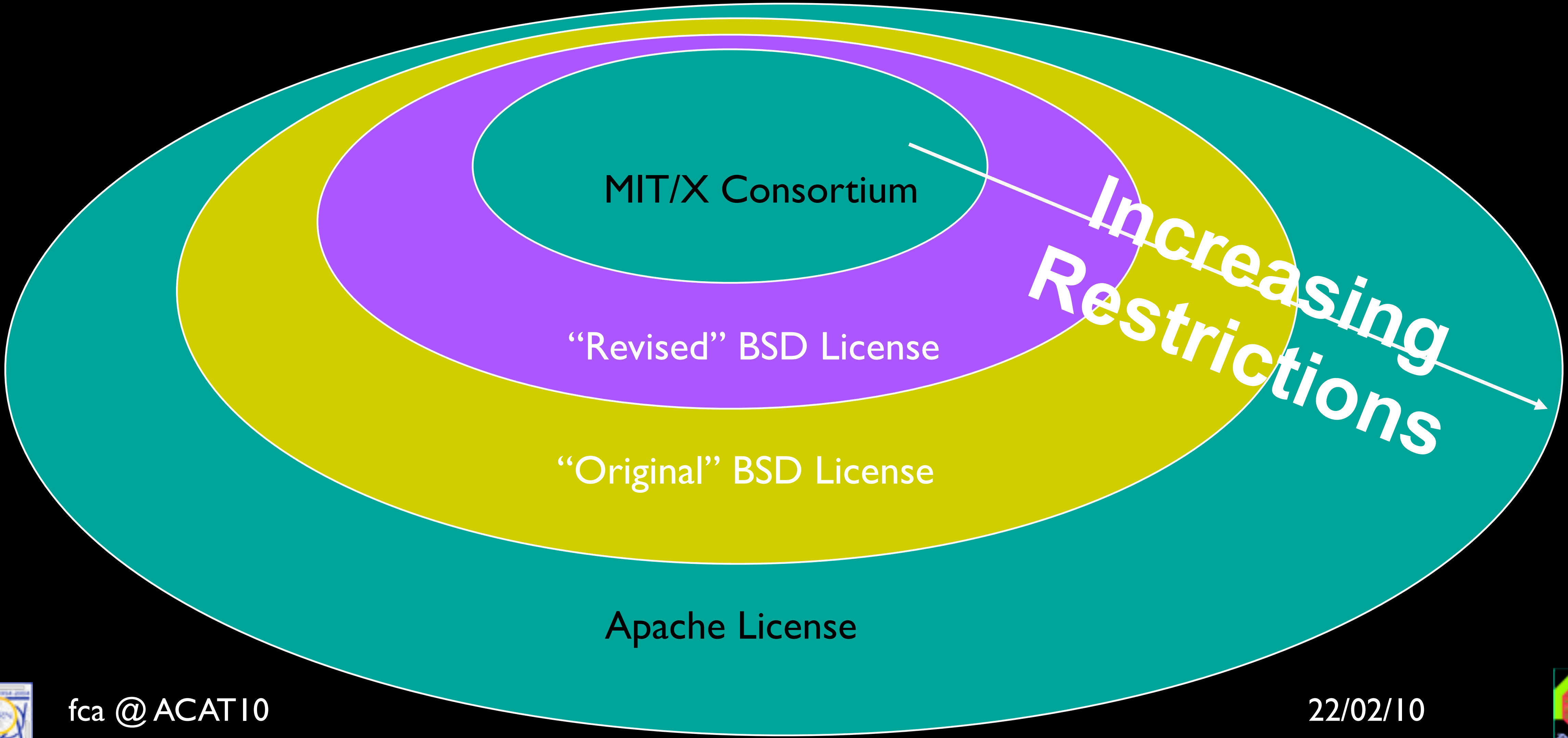
Different "Open Source" Licenses

MIT/X-Consortium License: truly "no strings attached"

"Revised" BSD License: MIT License + No Endorsement Clause

"Original" BSD License: Revised BSD License + Attribution Clause

Apache License: Original BSD License + No Use of "Apache" Name



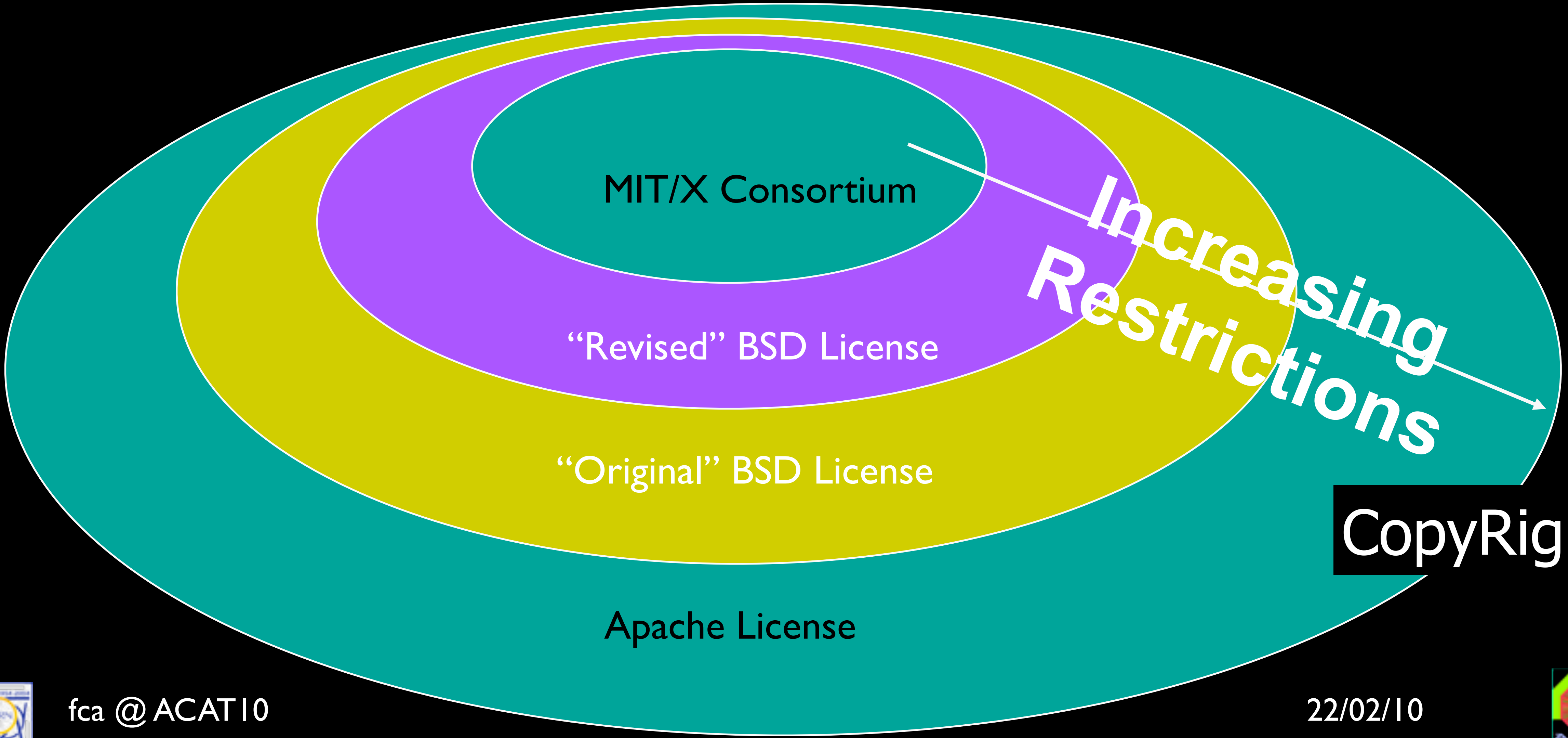
Different "Open Source" Licenses

MIT/X-Consortium License: truly "no strings attached"

"Revised" BSD License: MIT License + No Endorsement Clause

"Original" BSD License: Revised BSD License + Attribution Clause

Apache License: Original BSD License + No Use of "Apache" Name



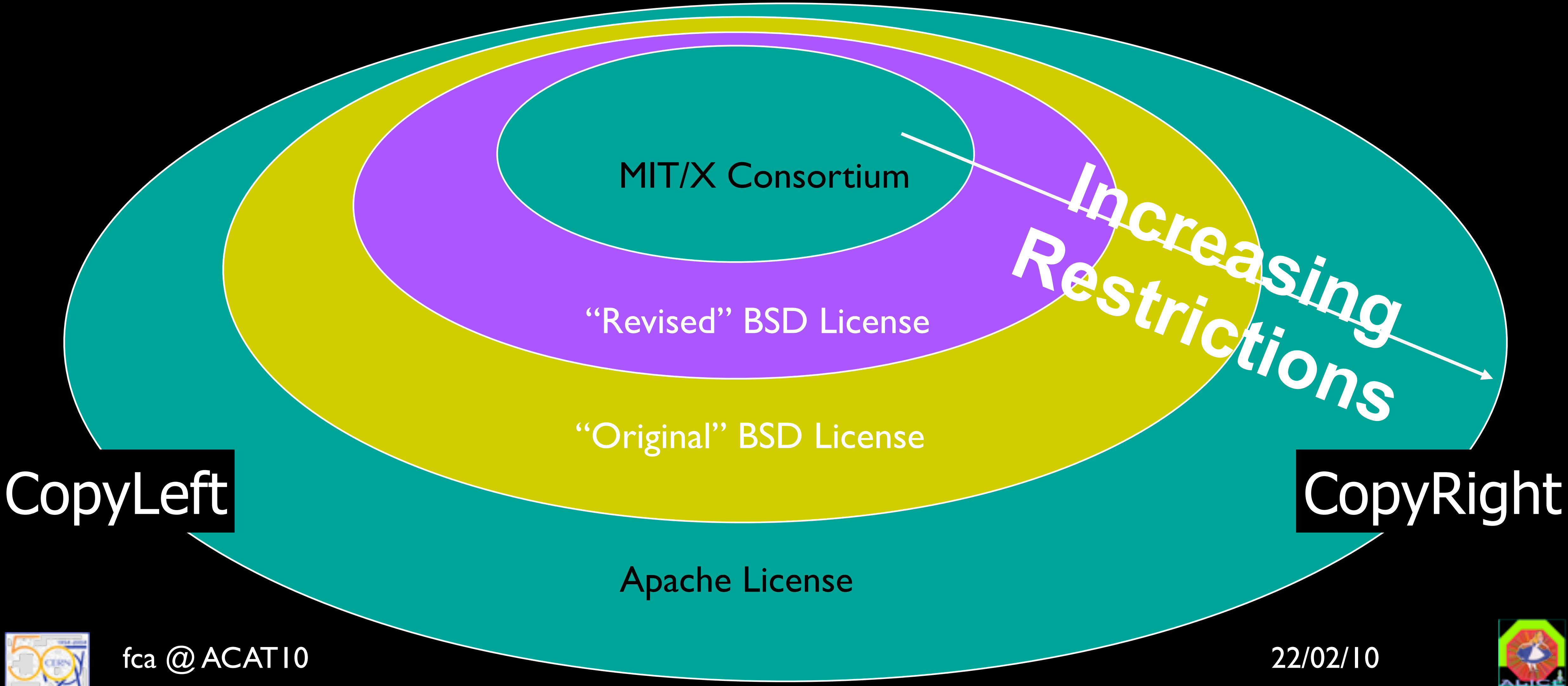
Different "Open Source" Licenses

MIT/X-Consortium License: truly "no strings attached"

"Revised" BSD License: MIT License + No Endorsement Clause

"Original" BSD License: Revised BSD License + Attribution Clause

Apache License: Original BSD License + No Use of "Apache" Name



OS licenses

- The world of Open Software licenses is very complicated

GNU General Public License (GPL)
GNU Library or "Lesser" Public License (LGPL)
BSD license
MIT license
Artistic license
Mozilla Public License v. 1.0 (MPL)
Qt Public License (QPL)
IBM Public License
MITRE Collaborative Virtual Workspace License (CVW License)
Ricoh Source Code Public License
Python license (CNRI Python License)
Python Software Foundation License
zlib/libpng license
Apache Software License
Vovida Software License v. 1.0
Sun Industry Standards Source License (SISSL)

Intel Open Source License
Mozilla Public License 1.1 (MPL 1.1)
Jabber Open Source License
Nokia Open Source License
Sleepycat License
Nethack General Public License
Common Public License
Apple Public Source License
X.Net License
Sun Public License
Eiffel Forum License
W3C License
Motosoto License
Open Group Test Suite License
Zope Public License

Agile Technologies
(aka SE catching up)

- SE response to HCP are the “Agile Methodologies”
  - Adaptive rather than predictive
  - People-oriented rather than process-oriented
  - As simple as possible to be able to react quickly
  - Incremental and iterative, short iterations (weeks)
  - Based on testing and coding rather than on analysis and design

- Uncovering better ways of developing software by valuing:

Individuals and interactions	**OVER**	processes and tools
Working software		huge documentation
Customer collaboration		contract negotiation
Responding to change		following a plan

That is, while there is value in the items on the right, we value the items on the left more.

- There are four factors to control a software project: time, manpower, quality and scope
- Time
  - The worst of them all… but the most widely used
- Manpower
  - The most misused … add people to a project which is late and you will make it later
- Quality
  - A parameter very difficult to control … writing bad software may take more time than writing good one
- Scope
  - The least used. It needs clear communication and courage, but is probably the most effective if well managed

Managing expectations

- There are four factors to control a software project: time, manpower, quality and scope
- Time
  - The worst of them all… but the most widely used
- Manpower
  - The most misused … add people to a project which is late and you will make it later
- Quality
  - A parameter very difficult to control … writing bad software may take more time than writing good one
- Scope
  - The least used. It needs clear communication and courage, but is probably the most effective if well managed

- XP in seven statements
  - Based on small, very interacting teams of people working in pairs
  - Testing is practiced since the very beginning
  - System integration is performed daily
  - Use cases driven, with specific techniques to estimate time and cost of the project
  - Programs are continuously refactored
  - Written documentation besides code is kept to minimum
  - Write the simplest system that can work!
- Move stability from plans to planning

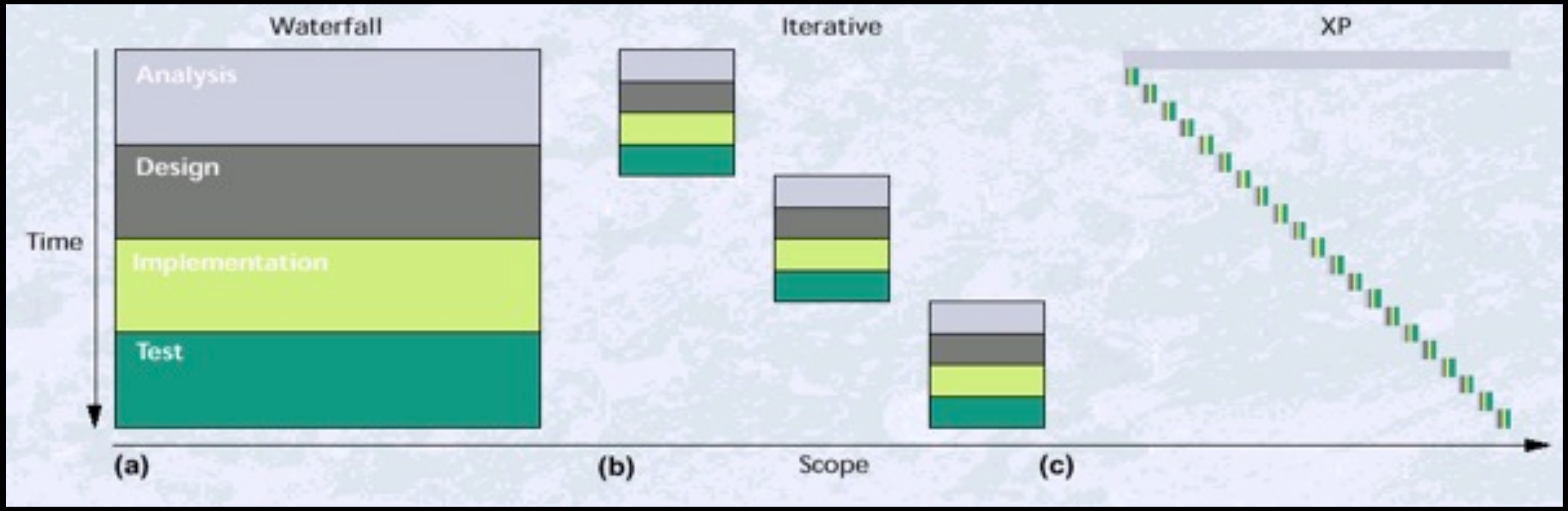
eXtreme Programming

- XP in seven statements
  - Based on small, very interacting teams of people working in pairs
  - Testing is practiced since the very beginning
  - System integration is performed daily
  - Use cases driven, with specific techniques to estimate time and cost of the project
  - Programs are continuously refactored
  - Written documentation besides code is kept to minimum
  - Write the simplest system that can work!
- Move stability from plans to planning

- **Communication**
  - A project needs continuous communication, with the customer and among developers
  - Design and code must be understandable and up to date
- **Simplicity**
  - Do the simplest thing that can possibly work
  - Later, a simple design will be easily extended
- **Feedback**
  - Continuous feedback from customers on a working system, incrementally developed
  - Test-based programming
- **Courage**
  - The result of the other three values is that we can be aggressive
  - Refactor mercilessly every time you spot a possible improvement of the system

eXtreme Programming

- **Communication**
  - A project needs continuous communication, with the customer and among developers
  - Design and code must be understandable and up to date
- **Simplicity**
  - Do the simplest thing that can possibly work
  - Later, a simple design will be easily extended
- **Feedback**
  - Continuous feedback from customers on a working system, incrementally developed
  - Test-based programming
- **Courage**
  - The result of the other three values is that we can be aggressive
  - Refactor mercilessly every time you spot a possible improvement of the system



- Some of the “rites” of HEP software find now a rationale explanation
  - That we were not able to express
- But our environment adds complexity to the one foreseen by agile methods
  - Large and distributed teams, no hierarchy
- Introducing (and modifying) agile methods in our environment effectively increase our efficiency
  - Help planning for distributed teams
  - Reduce the lead time for people to be effective
- A worthy goal for Software Engineers working in HEP!
- An occasion to collaborate with advanced Computer Science and Industry?

Agile technologies and HEP

- Some of the “rites” of HEP software find now a rationale explanation
  - That we were not able to express
- But our environment adds complexity to the one foreseen by agile methods
  - Large and distributed teams, no hierarchy
- Introducing (and modifying) agile methods in our environment effectively increase our efficiency
  - Help planning for distributed teams
  - Reduce the lead time for people to be effective
- A worthy goal for Software Engineers working in HEP!
- An occasion to collaborate with advanced Computer Science and Industry?

- HEP has developed and successfully deployed its own SE method but never realised it
- Market conditions now are more similar to the HEP environment
  - And modern SE is making justice of some HEP traditions and rituals
- This movement may be important for HEP as we can finally
  - Express our own SE culture
  - Customise and improve it
  - Teach and transmit it
- XP is not a silver bullet but rather the realisation that such a thing does not exist and a formalisation of common sense
- The big challenge will be for HEP to move agile technologies in the realm of distributed development

(a preliminary) Conclusion

- HEP has developed and successfully deployed its own SE method but never realised it
- Market conditions now are more similar to the HEP environment
  - And modern SE is making justice of some HEP traditions and rituals
- This movement may be important for HEP as we can finally
  - Express our own SE culture
  - Customise and improve it
  - Teach and transmit it
- XP is not a silver bullet but rather the realisation that such a thing does not exist and a formalisation of common sense
- The big challenge will be for HEP to move agile technologies in the realm of distributed development

"That's
all
folks!"