A REFLECTION ON SOFTWARE ENGINEERING IN HEP

F.CARMINATI
CHEP 2012, NEW YORK, MAY

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 ~ 3000 physicists and engineers)
- – Programs for the future machines will be, if possible, even more complicated
- 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

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 1300 pages
- Several new languages have been emerging with an uncertain future
  - C#, Java, Perl, Python, Ruby, php…
- The Web world adds a new dimension to computing
- Not to talk about GRID…
- What about the next generation?

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

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."

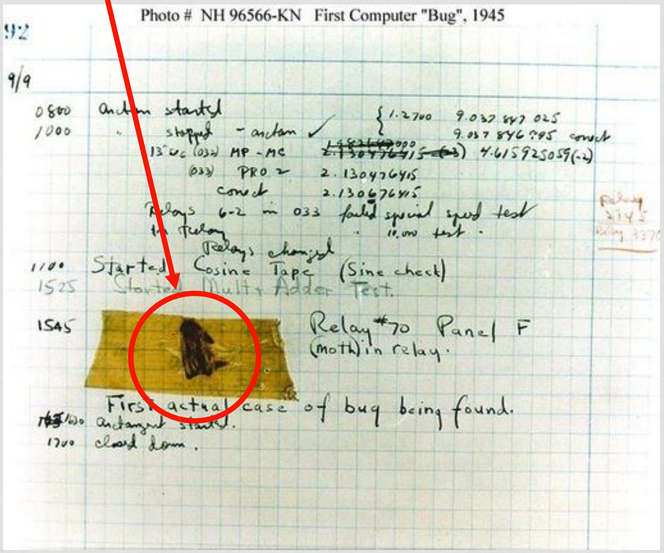
SOFTWARE, SOFTWARE CRISIS AND SE

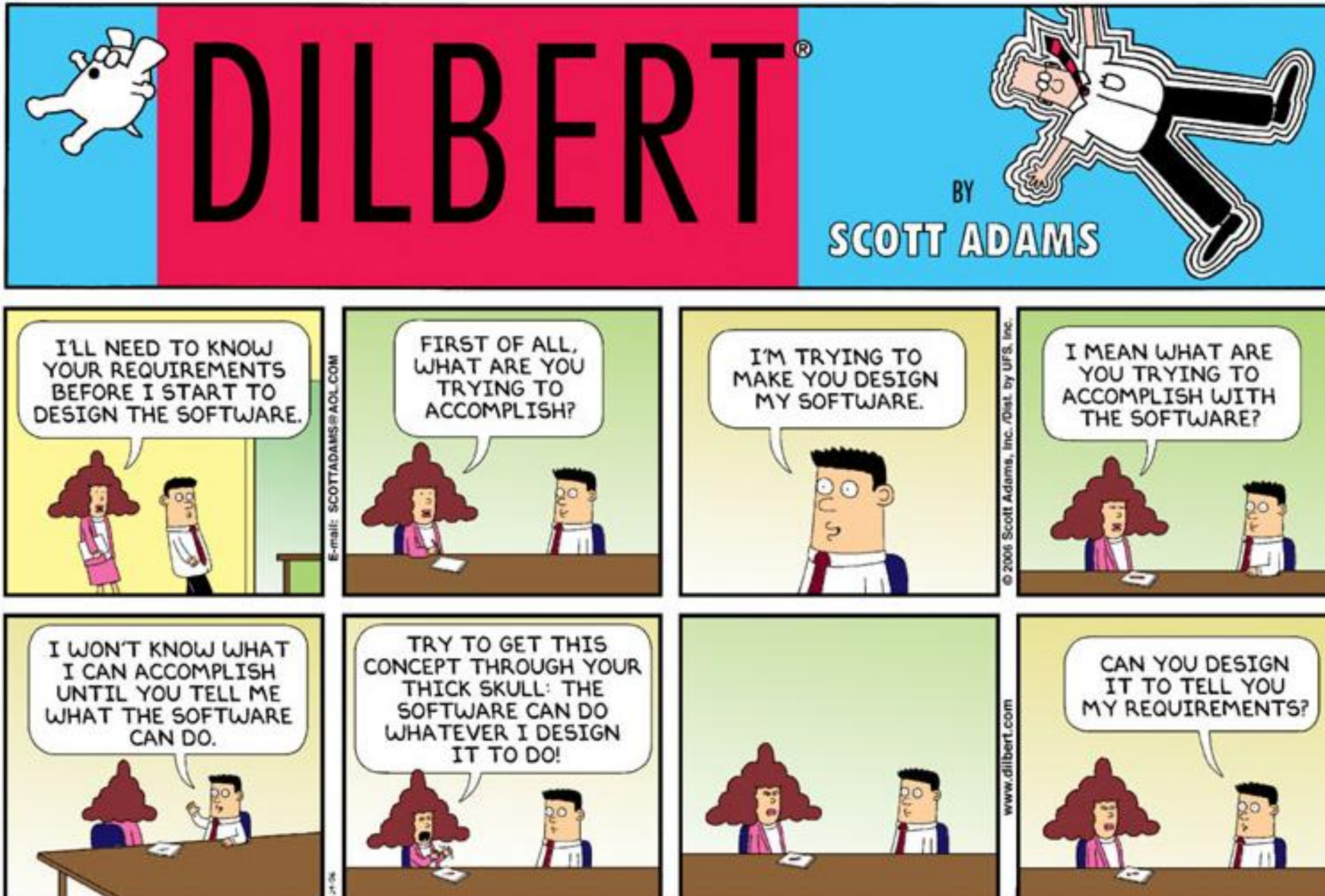
- The software crisis comes from the failure of large software projects to meet their goals within budged and schedule
- Major worry of managers is not
  - Will the software work?
- But rather
  - Will the development finish within time and budget?
  - … or rather within which time and budget …
- SE 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

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

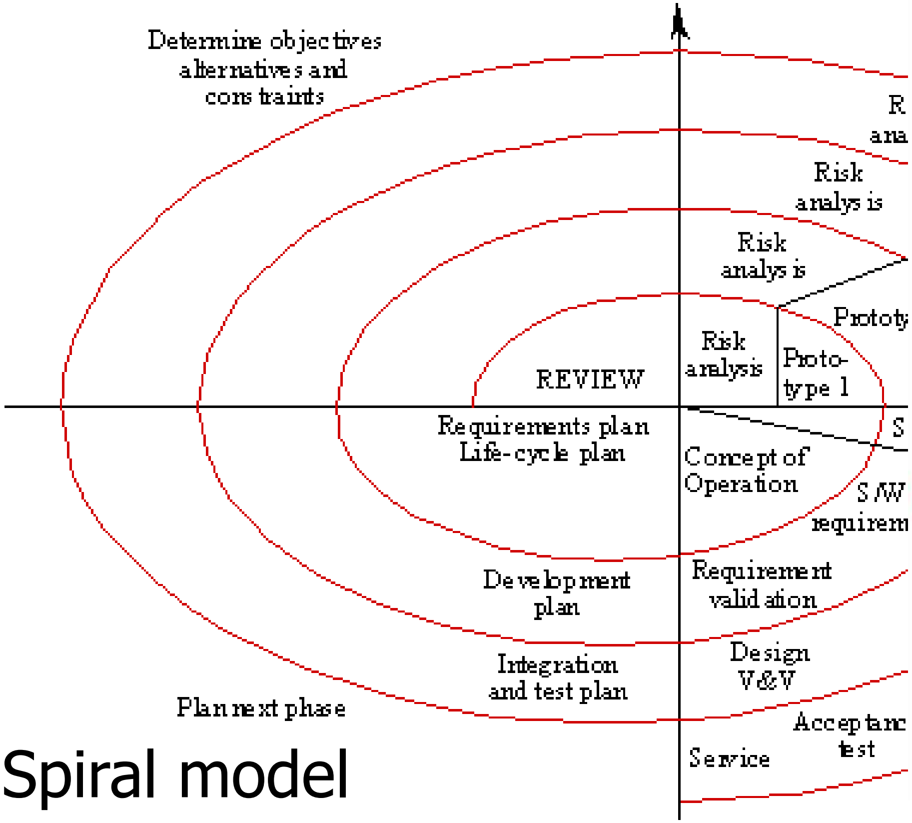
1945: Grace Hopper discovers the first computer bug



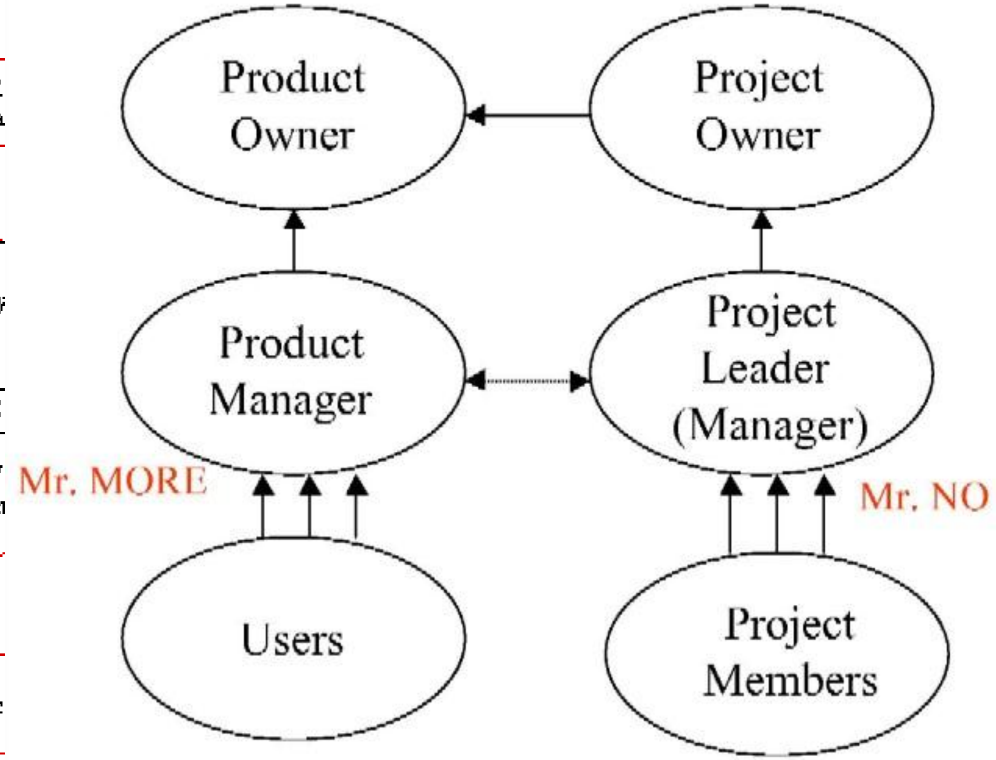

DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL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

SE CRISIS

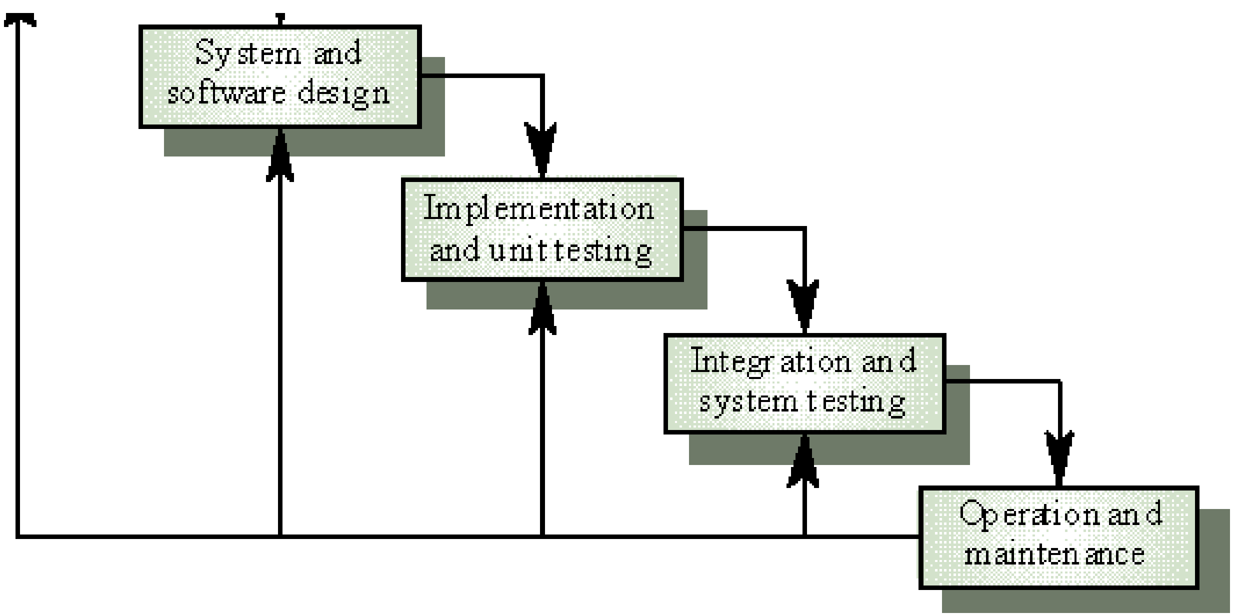
- Software is opposed to hardware because it should be flexible
- Yet the reason of the failure of software process is often identified in the changes intervening during the development
- The heart of SE is the limitation of the impact of changes
  - Changes are avoided by a better design
  - A better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



Spiral model



- HCP are sui
  - The time el
    be as long
- In the e-bus
  characteriz
  - High speed



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”

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
- ROOT and xrootd are de-facto standards
- And yet we (as a community) have not used canonical SE
- Did we do something right?

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

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

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

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 artifact 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)

- 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)

- 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

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?

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..."

AGILE TECHNOLOGIES
(AKA SE CATCHING UP)

- SE response to HCP are the "Agile Methodologies"
  - Adaptive rather than predictive

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

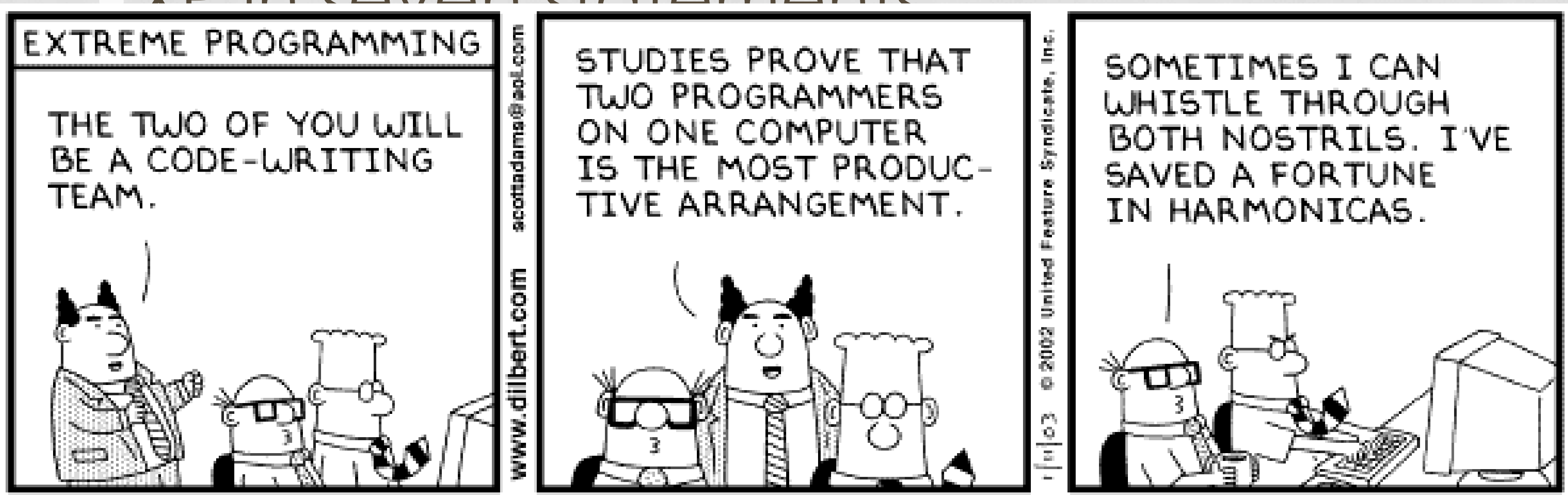
MANAGING EXPECTATIONS

- There are four factors to control a software project: time,

probably the most effective if well managed

EXTREME PROGRAMMING

- XP in seven statements

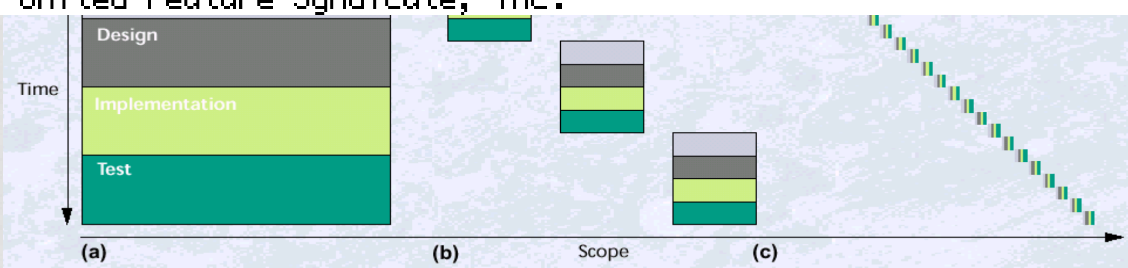


minimum

  - Write the simplest system that can work!
- Move stability from plans to planning

EXTREME PROGRAMMING

- **Communication**



(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!"