

The background features a dark blue gradient with faint, light blue concentric circles and degree markings (40, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260) on the left side, suggesting a technical or engineering theme.

# SOFTWARE ENGINEERING BASICS

YUNG-PIN CHENG  
SOFTWARE RESEARCH CENTER  
NATIONAL CENTRAL UNIVERSITY  
TAIWAN

# 開發軟體很簡單？

- 寫程式真的很簡單！
- 很多人國中就會寫程式了！
- 很多人熬個夜就趕的出數千行的程式了！
- Internet 到處都有程式可以抄！
- 錯了可以馬上改！
- 有新的 idea 可以馬上try！



If you are hired to build a  
dog house,  
what ability you should  
Have?



Win This Beautiful Dog House **CLICK  
HERE**



# DIFFERENT SCALE OF SOFTWARE DEVELOPMENT

- If you are hired to build a Taipei 101
- What skills and talent do you need to have?



# 困擾**COMPUTER SCIENCE**幾年的老問題

- 軟體的重用性(software reuse)
  - Why most software units cannot reused with minimum effort?
- 軟體的維護性(maintainability)
  - Why a programmer need to trace a program for a long time in order to make a small fix
- 軟體的品質(software quality)
  - why software quality is in general so bad compared with other discipline (台灣高鐵與台灣彩券)
- 軟體的擴充性 (software evolution )
  - Why extension is needed, a whole program need to be modified?



# THE JOY OF PROGRAMMING(CODING)

為什麼寫程式很有趣呢？寫程式的人期望從中得到什麼樣的樂趣呢？

首先，是創造的趣味，就好像小孩子的快樂地用泥巴做成一個派。大人們也一樣，從創造中可以得到十足的快樂，特別是自己設計的東西。我想這樣的樂趣一定是映自於上帝創造萬物的樂趣，你看每一片樹葉、每一片雪花的獨特與新奇，不正顯示了這種創造的樂趣嗎？

其次，令人感到愉快的，是我們所創造出來的新東西，竟然對別人有用。在我們的內心深處，都希望別人用我們做出來的東西，並且發現這東西對他很有幫助。從這點看來，軟體系統跟小孩子第一次為「父親辦公專用」所做的黏土筆筒沒有什麼不同。

第三，是那種打造精巧機制時，類似推理、解謎的過程，令人迷戀。把彼此聯動的零件組合起來，眼看著成果真的按照了我們原先所設想的方式微妙地在運作，受程式操控的那台電腦不但擁有彈珠台或自動點唱機的迷人魅力，並且將之發揮到淋漓盡致。

第四，是持續學習的樂趣。這種工作具有不重複的特質，也不知為什麼所要解決的永遠是全新的問題，而解決問題的人總是可以從中學到些東西：有時是在實務方面，有時是在理論方面，或兩者都有。

最後，是在如此易於操控的介質（tractable medium）上工作的快樂。程式設計師就像詩人一樣，只動動腦筋就可以做事，運用想像力，便可以憑空造一個城堡出來，很少創造性工作的介質是如此富於彈性、如此方便地讓你修修改改，並輕易地就可以把一個偉大的構想實現出來。（當然在後面我們會看到，這樣的易操控性也有它伴隨而來的問題。）

然而，程式又跟詩人所用的字詞不同，程式本身是沒什麼，但它可以製造出看得到的效果，讓你真實感受到它活生生的地在動、在做事。它能夠列印、畫圖、發出聲響、移動機械手臂，只要在鍵盤上敲入適當的咒文，整個螢幕的畫面就生氣蓬勃起來，顯現出了我們未曾見過、或在現實生活中不可能見到的事物，神話和傳說中的魔法在我們有生之年實現了。

所以寫程式實在很有趣，因為它滿足了我們潛藏於內心創造事物的渴望，並且激發了我們每個人原本就擁有的快樂感受。

# THE PAIN OF CODING

讓這些，可以了解這些，中慣己，你，上上，文習自，訊，中際義，這程試，蟲免，咒不適，資語域實名，的術領。於，的測正。臭都，樣類，一人為，相上任權相，而別碼以用毛的背後，法。認，提管像稱的，式式（錯、找尋創，魔來我，人句好相上，別套，任質，由，過責實，程程整出得尋何不，跟出，別套，任質，由，過責實，系寫不去手，任也，難處，面展完，標不與多，對所也間隨趣。程，的，方施麼1，標不與多，在法到分資作任得到，天生有。腦魔做部給工責上得，也對電，要的供與的義會，它面。池需難人節下名才，然美差少困別細扛在，的坦完有很最由作所曾功，趣易常稍也程，工你未成，有容非要動工標的擔都做，是更得只活體目已承位作，都，現，的軟定自以職工，樣候表頓類習設排足的把，樣時須停人學人安不好先，是的必個，是別行並做得。非的誤，偉，不難你一美美由自力情你力點也良錯裡了，能，並困，或完完，夠權事：權一況不件世界為，就只，遇先字麼求次能的把是）有情個文世說，就枯，而遭首個這追其少有能況的還的是，的雖（bug）附，然在一到於，很擁正情得，特往）想，不，你的做習，你所真的應，獨往例理，（不

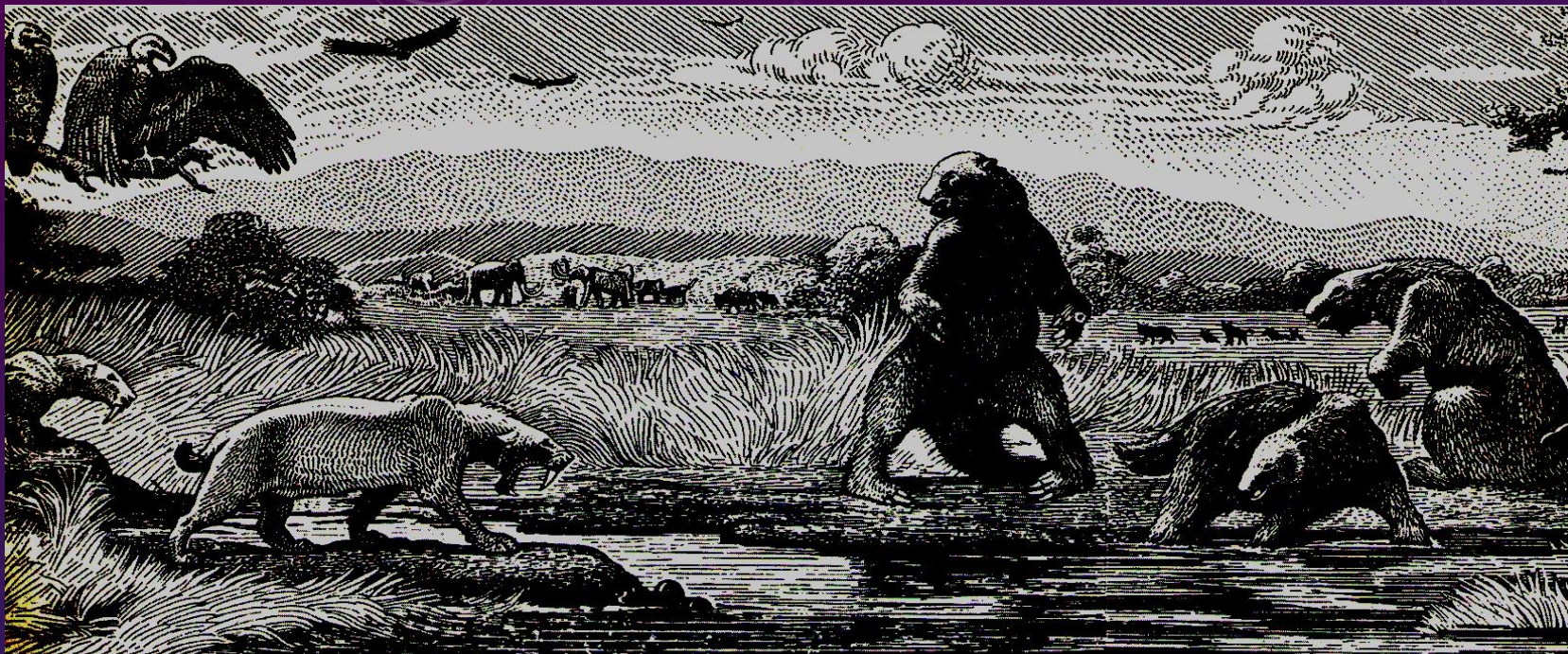






你身處湖心當中，划著一艘小船，船底有幾個漏洞，水正慢慢地滲進來。你想要把漏洞堵起來，但卻遭到一大群仲夏凶惡蚊子的攻擊。能夠隨時注意到是否有問題的發生，或更進一步能預先加以防範（修補好漏洞），在理論上已經算不錯的了。但是你得一直忙著去驅趕昨日的問題（不停地去打蚊子），因而完全抽不出一絲的空閒來實踐任何的理論。





史前時期最駭人的景象，莫過於一群巨獸在焦油坑裏做垂死前的掙扎。不妨閉上眼睛想像一下，你看到了一群恐龍、長毛象、劍齒虎正在奮力掙脫焦油的束縛，但越掙扎，焦油就纏得越緊，就算牠再強壯、再厲害，最後，都難逃滅頂的命運。過去十年間，



## 摘錄自



### 人月神話：軟體專案管理之道（20週年紀念版）

The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition

作者：Frederick P. Brooks, Jr./著

譯者：錢一一

出版社：經濟新潮社


出版日期：2004年04月04日

語言：繁體中文 ISBN：9867889185

裝訂：平裝

定價：~~480~~ 元 優惠價：7 折 **336** 元

優惠期限：2010 年 03 月 31 日止

抵用  最多再省 **101** 元 [»詳情](#)

# 開發應用軟體和寫學校作業真的很不同

## Real Software

- User
  - 軟體是寫給人用的. 軟體的價值決定於它的實用性
  - 給誰用? 如何用? 作什麼用?
- Specification
  - Part of the work is defining the spec.
  - To change, or not to change: that is the question
  - 彈性? 變化? 修改?
- Quality
  - 正確, 一致, 穩定 : Do as you said.
  - Life Time of Years
  - Write, Read, Rewrite, Read, Rewrite ..

## Programming in School

- User
  - Only yourself .
- Specification
  - Fixed. Well defined.
  - Few Constraints
- Quality
  - Nobody cares at all
  - Life Time of one semester
  - Write Once, Never Read
  - Developer = Maintainer



# 另一個大差別 : **SIZE DOES MATTER!**

風多雜枝分羊  
招夢手生力亡  
大長多外多路  
樹夜人多節備歧



一個小程序



一個大軟體

GODZILLA

大大的頭痛!





客戶解釋他們想要的



專案主持人對客戶需求的認知



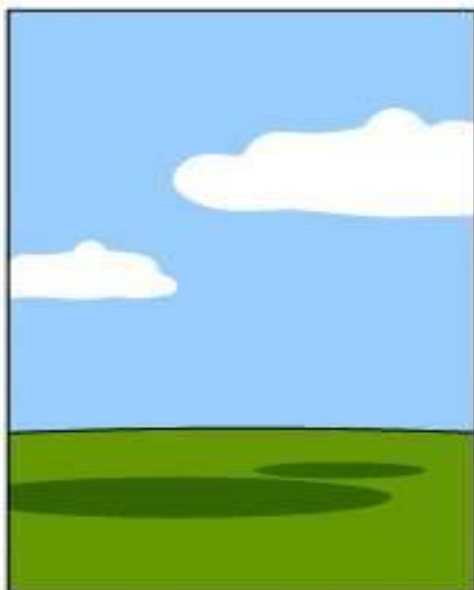
系統分析師所設計的



程式設計師所寫出來的



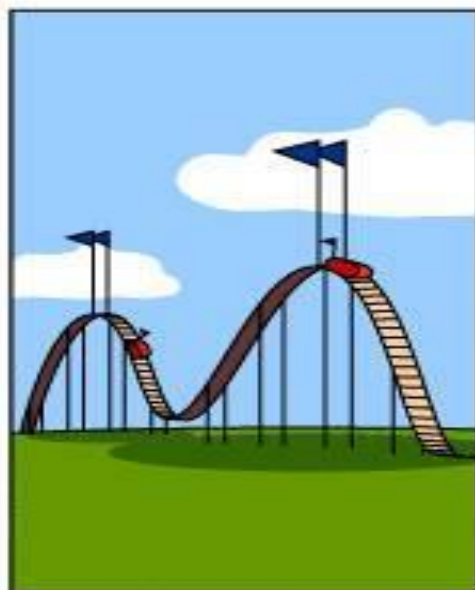
顧問所描繪的願景



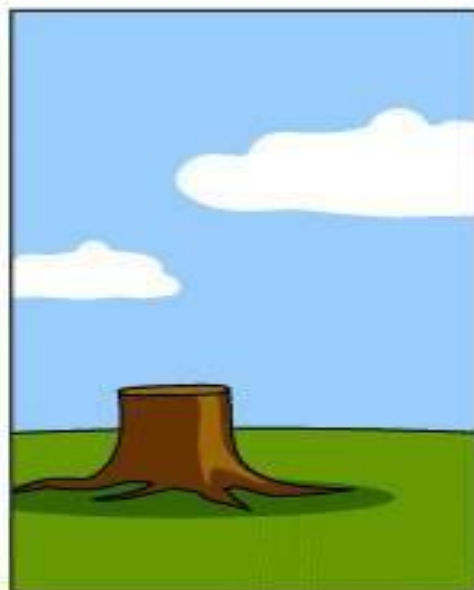
專案的文件



最後交付給客戶的軟體



客戶所付的錢



上線後的技術支援



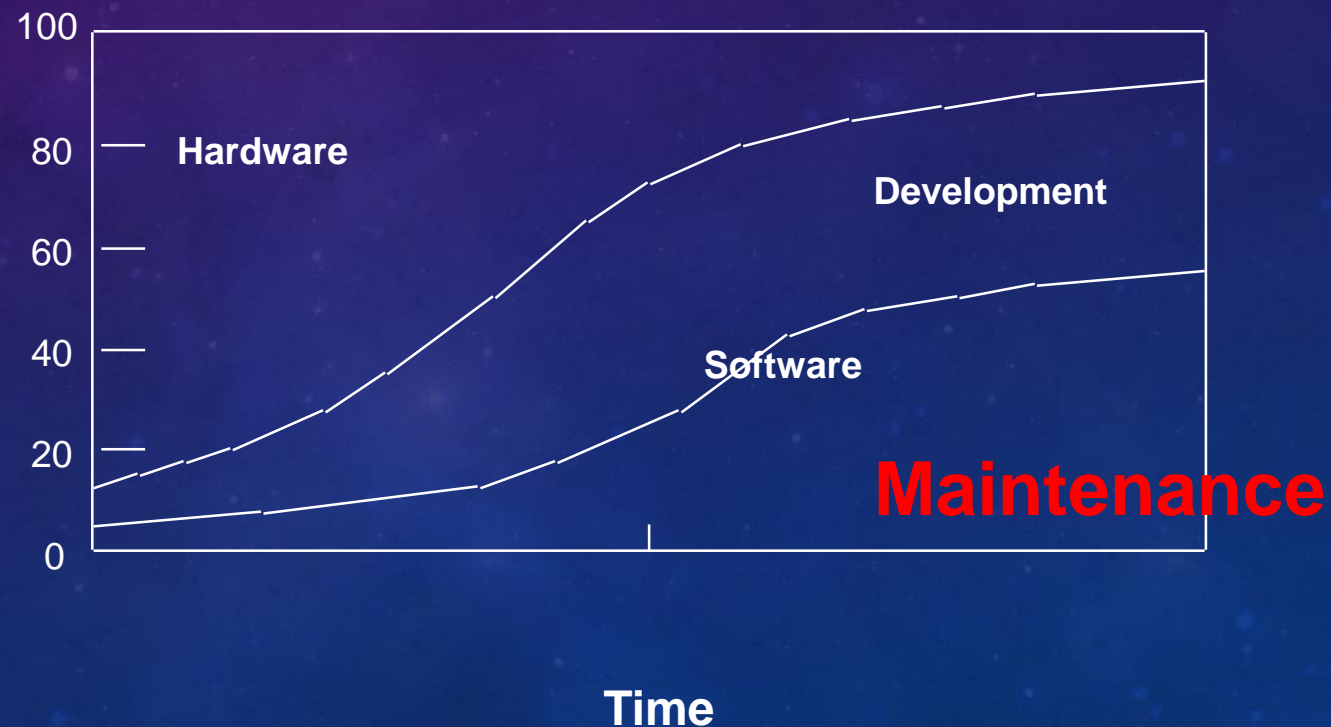
客戶真正需要的





# 作好應用軟體的維護

- 生小孩不容易, 養小孩更難!
- 不只是 debug, 更是服務!



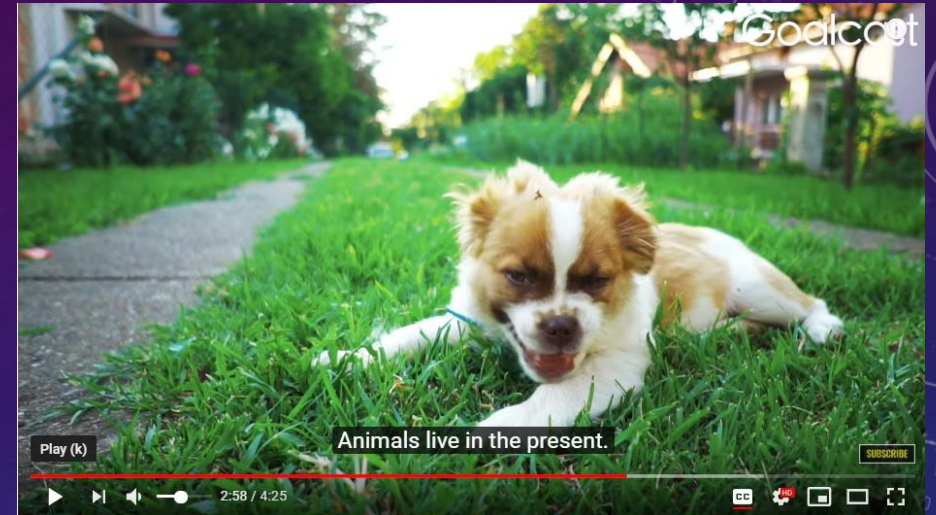
# 軟體工程突破盲腸的一句話

- 土木工程裡，蓋了橋，可以用幾十年，有標準工程圖，有基本建造的元件（如磚頭）
- 電子產品出產之後，基本上功能已經 fixed。很難進行修改。有新的需求，通常採用的是將舊的機器報廢掉。少有機會可以在既有的機器基礎上做翻修。有標準電子路圖，有基本建造元件。
- 軟體呢？
  - 幾乎產品一 released 就會有立即的修改
  - Requirement 是個 moving target
  - 只能又舊有的軟體基礎上進行維護與更新。
  - 沒有標準工程圖
  - 新的技術如UML 不像其他的工程具備有幾何關係，也就是說軟體的藍圖不能靠單一的圖形表示方式來展示





# MARSHMALLOW TEST



- <https://www.youtube.com/watch?v=47DmUM7MW7s>

當你的產品在市場上存活了，你的軟體維護與擴充的工作會佔據  $2/4-3/4$  的開發成本與時間

為何你在學校無法體會？因為你在學校寫的軟體都沒有存活到被擴充與修改



# ANALOG

- Software engineering 的研究學者已經發現
  - 以前的軟體工程學者與產業一直想把土木工程那一套弄進來。現在基本上大家接近放棄這條路，改走敏捷開發
  - 想要與任何工程領域類比，都會遇到相當的困難
  - 軟體開發具有非常獨特的工程性質 –

# MANUFACTURING IN OTHER DISCIPLINE

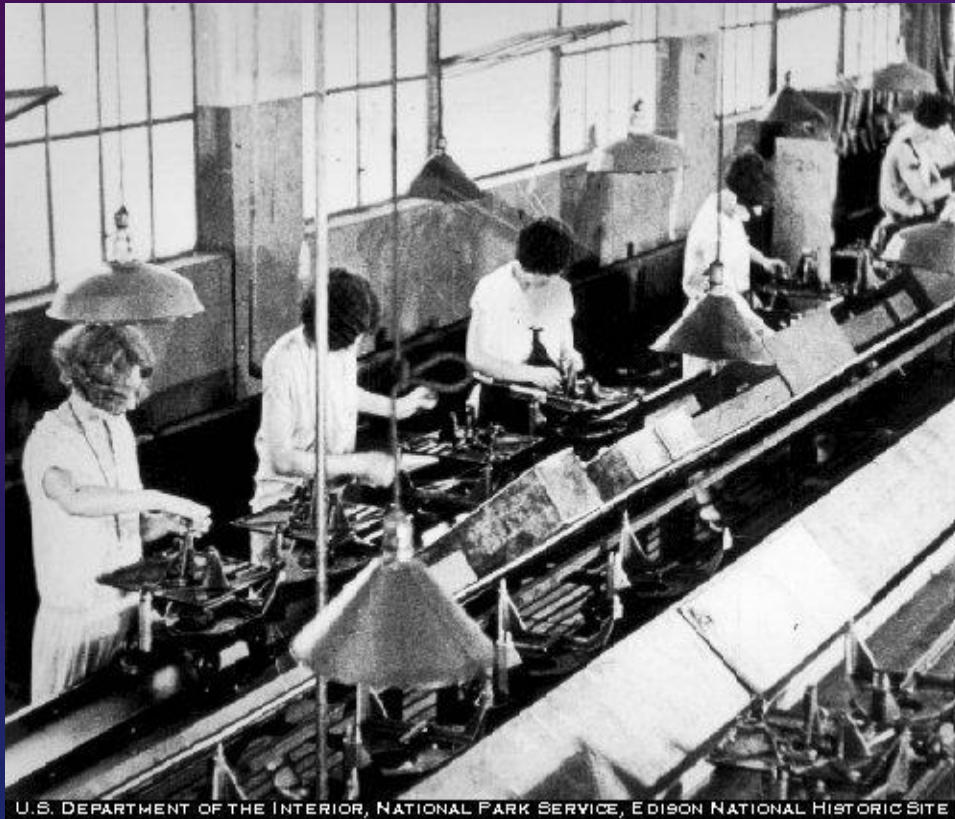
- 在製造業的世界裡面
  - 產品品質(quality)是很重要的一個因素
  - 當你投入非常多成本與原物料時你永遠希望你生產出來100個產品有100個都是可以賣的
- QC (Quality Control) 品質控管
  - 製造過程從設計到生產可能有數十道到幾百道程序
  - 每一道程序都可能影響到後面的品質
  - 如何透過製造流程的改善(process improvement)來提高良率，一直都是製造工業的重要課題



# HOW TO IMPROVE QUALITY?

- During manufacturing If it can be done by machine, the best way to improve quality is 自動化。機器不容易犯錯而且可以不斷地重複單調無聊的工作。機器會出錯的時候通常是由於製造機器的磨損，必須重新校正。
  - Ex. 日本的步進馬達精確度超高的秘密。
- Question
  - 當某部分的工作不能由機器來做的時候，製造過程如何做到品質控管？

# ASSEMBLY LINE IN MANUFACTURING



U.S. DEPARTMENT OF THE INTERIOR, NATIONAL PARK SERVICE, EDISON NATIONAL HISTORIC SITE





## IN ASSEMBLY LINE

- Each person is doing one thing that is simple enough, which is not easy to make errors
- A labor does not need to be smart to do the task in the assembly line
- Assembly line decomposes the work into many smaller tasks which can be done by a labor in a short time and minimum skills
- Discovering the cause of defect is not a difficult task.

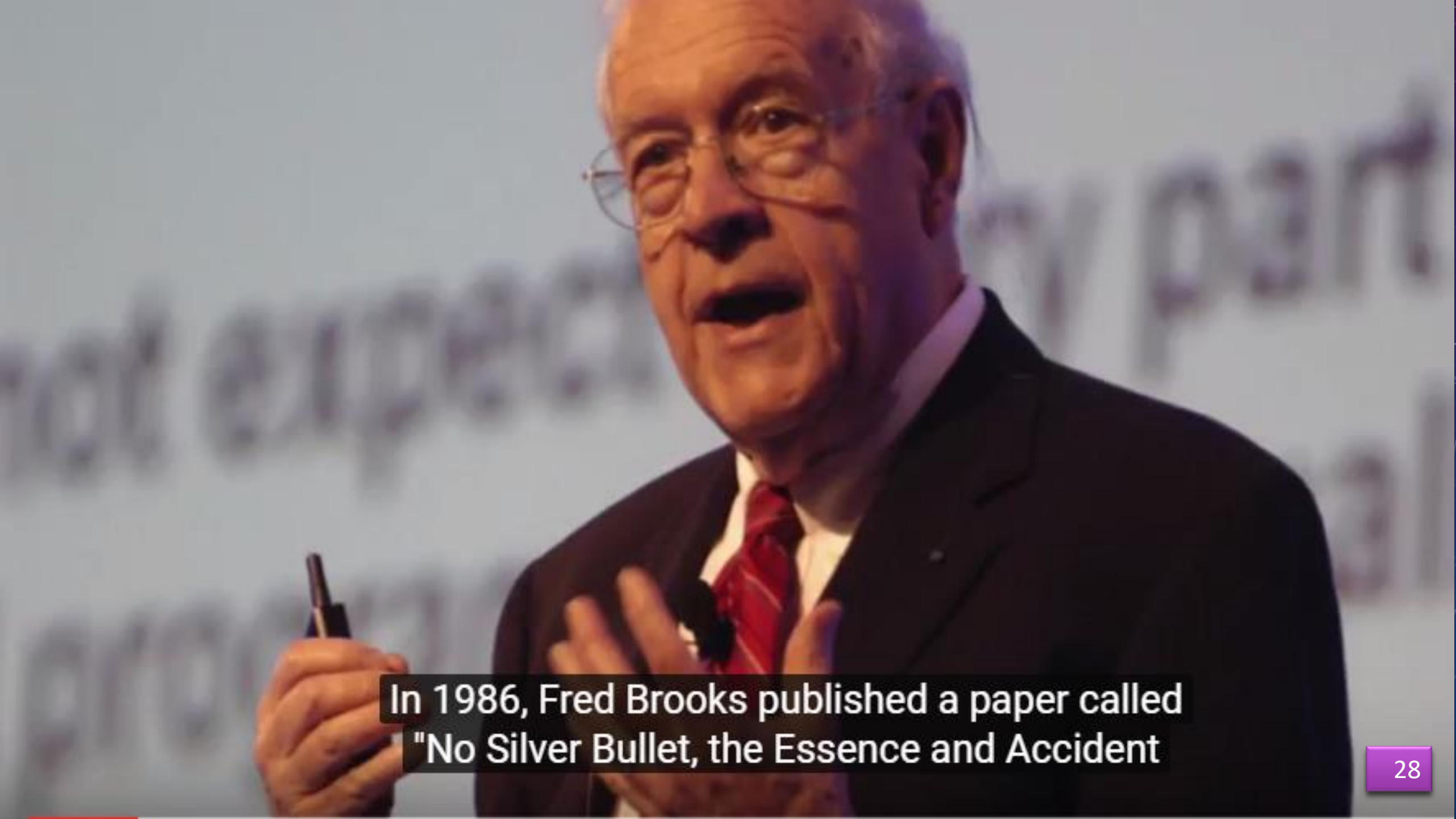
# AGAIN, LET'S REVIEW THE ENGINEERING PROCESS OF OTHER ENGINEERING FIELDS.

1. Idea
  2. Marketing analysis (Requirement Analysis)
  3. Analysis and Design
  4. Manufacturing (QC)
  5. Testing (QA)
  6. Release product
- Guess, who get the higher pay?



# SOFTWARE ENGINEERING

1. Idea
2. Requirement analysis and specification (100% design)
3. Design and analysis (100% design, QC here?)
4. Implementation (95% design?, QC here?)
5. Manufacturing (compilation – no cost manufacturing)
6. Testing



In 1986, Fred Brooks published a paper called  
"No Silver Bullet, the Essence and Accident



There is no single  
development, in either  
technology or  
management technique,  
which by itself promises  
even one  
order-of-magnitude  
improvement  
within a decade in ...

or management technique,  
which by itself promises even one order-of-magnitude



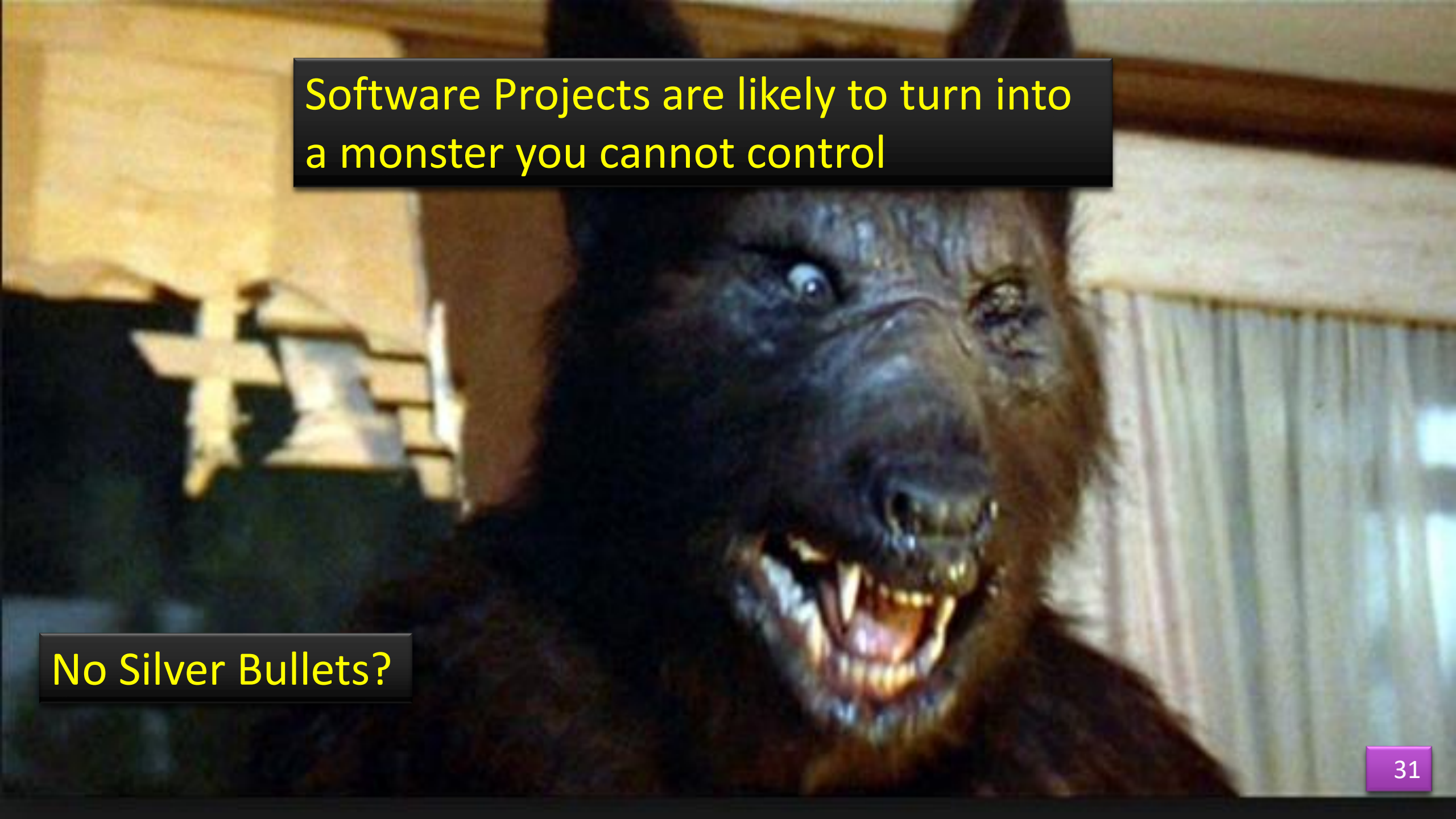
**PRODUCTIVITY**

**RELIABILITY**

**SIMPLICITY**

improvement within a decade in productivity,  
in reliability, in simplicity”

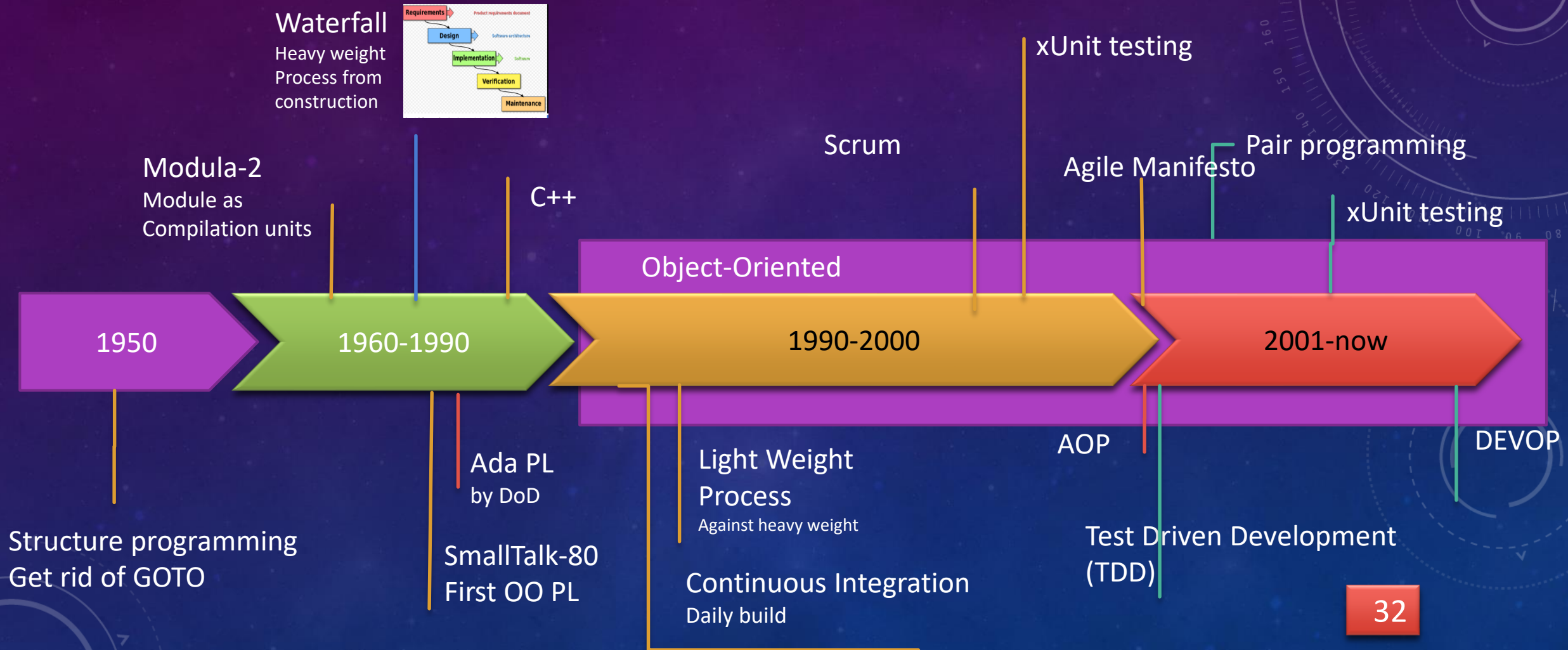




Software Projects are likely to turn into  
a monster you cannot control

No Silver Bullets?

# THE HISTORY OF SILVER BULLET ATTEMPTS





# THE DAILY WORK OF A PROGRAMMER

