الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

مقرر بنيان البرمجيات | كلية الهندسة المعلوماتية | المحاضرة الثانية

# General Principles in Software Design and Architecture
## "System decomposition"

د. رياض سنبل

# Software Crisis

- The term 'software engineering' was suggested at conferences organized by NATO in **1968 and 1969 to discuss the 'software crisis'**.
- Projects running
  - over-budget,
  - over-time,
  - with low quality,
  - and without meeting requirements

# 50+ Years After the Crisis

**MODERN RESOLUTION FOR ALL PROJECTS**

| | 2011 | 2012 | 2013 | 2014 | 2015 |
|---|---|---|---|---|---|
| SUCCESSFUL | 29% | 27% | 31% | 28% | 29% |
| CHALLENGED | 49% | 56% | 50% | 55% | 52% |
| FAILED | 22% | 17% | 19% | 17% | 19% |

*The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.*

these problems. The software crisis has been slowly fizzling out, because it is unrealistic to remain in crisis mode for more than 20 years. Software engineers are accepting that the problems of software engineering are truly difficult and only hard work over many decades can solve them

*Deshpande Shweta - The Ohio State University, 2011.*

# 50+ Years After the Crisis

- Even in NASA!
- Mars Polar Lander
  - 1998
  - 110 million USD!

*The cause of the communication loss is not known. However, the Failure Review Board concluded that the most likely cause of the mishap was a* ***SOFTWARE BUG*** *that incorrectly identified vibrations, caused by the deployment of the stowed legs, as surface touchdown. The resulting action by the spacecraft was the shutdown of the descent engines, while still likely 40 meters above the surface. Although it was known that leg deployment could create the false indication, the software's design instructions did not account for that eventuality*

*From the "Report of the Loss of the Mars Polar Lander and Deep Space 2 Missions -- JPL Special Review Board (Casani Report) - March 2000".*

- Larger List of well-know software bugs:
  - https://en.wikipedia.org/wiki/List_of_software_bugs

# But.. why software projects fails?

Complexity & Changes

# But.. why software projects fails?

*when errors are discovered*

*Many functions*

*when the developers have a better understanding of the application.*

*Many participants from different disciplines*

## Complexity & Changes

*The time between technological changes is often shorter than the duration of the project.*

*Many systems are so hard to understand*
*(even during their development phase)*

***No idea what to build***

*The widespread assumptions of a software project manager that all changes have been dealt with and that the requirements can be frozen*

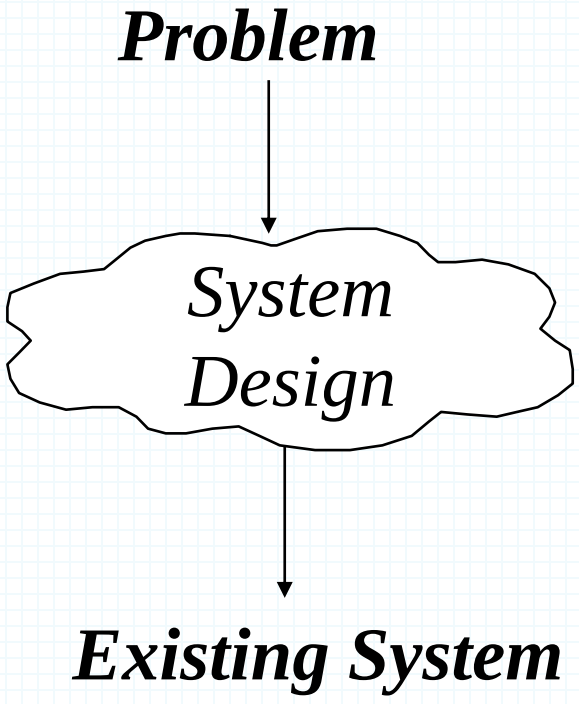
# So.. how to make software projects succeed!

- Very Big Question! But these are some of the main research directions:
  - More abstract programming languages!
    - OOP, High level languages, UML…
    - But still “the complexity is in the problem, not in the solution”
  - Better software design
    - Modularity, Abstraction, Anticipation of change, etc
  - Formal methods:
    - Mathematical approaches to solve software problems!
    - Eliminate human error by proving that our software is “correct”!
  - Development processes:
    - Waterfall development, Spiral development, Agile, etc.

*the main focus of this course*

# Software Architecture Definition

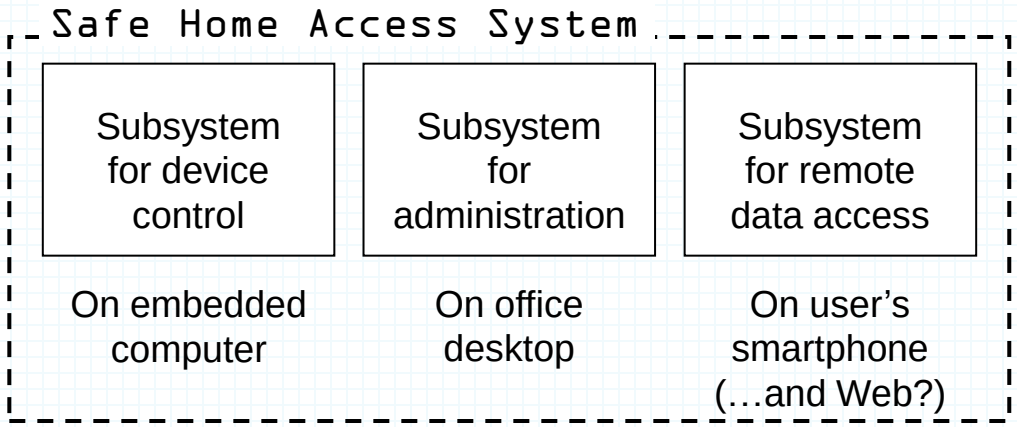
- Software Architecture =
  a set of high-level **decisions** that determine the structure of the solution
  (parts of system-to-be and their relationships)
- Bridge the gap between a problem and an existing system in a manageable way




*BUT What are these decisions?*

*Software is not a long list of program statements but it has structure*

# Example Architectural Decisions




## Example decisions:

- *Decision on system decomposition*
- *Decision on mapping software-to-hardware*
- *Decision on development platform or operating system (Android vs. iOS, etc.)*
- *Decision on how to fit the subsystems together ("architecture style")*

# But..
# Why We Want To Decompose Systems!

# Why We Want To Decompose Systems

- Tackle complexity by “divide-and-conquer”
- See if some parts already exist & can be reused
- Support flexibility and future evolution by decoupling unrelated parts, so each can evolve separately (“separation of concerns”)

# Why We Want To Decompose Systems

*The system is designed to scrape social media data, perform in-depth analysis, and predict specific product limitations based on user reviews. It then presents the findings through comprehensive reports and charts, effectively illustrating the results. Users can access their accounts, enter their products info (including their related social media pages) and use various functionalities either from a mobile app or a website.*

design choices?

system maintainability and scalability

Required resource and technologies?

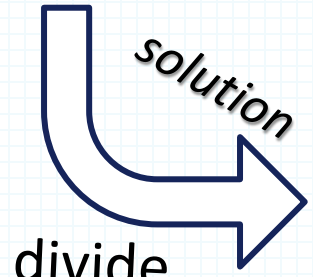
plan for changes?

Testing?

Time estimation?

# Why We Want To Decompose Systems

*The system is designed to scrape social media data, perform in-depth analysis, and predict specific product limitations based on user reviews. It then presents the findings through comprehensive reports and charts, effectively illustrating the results. Users can access their accounts, enter their products info (including their related social media pages) and use various functionalities either from a mobile app or a website.*




divide
And
conquer