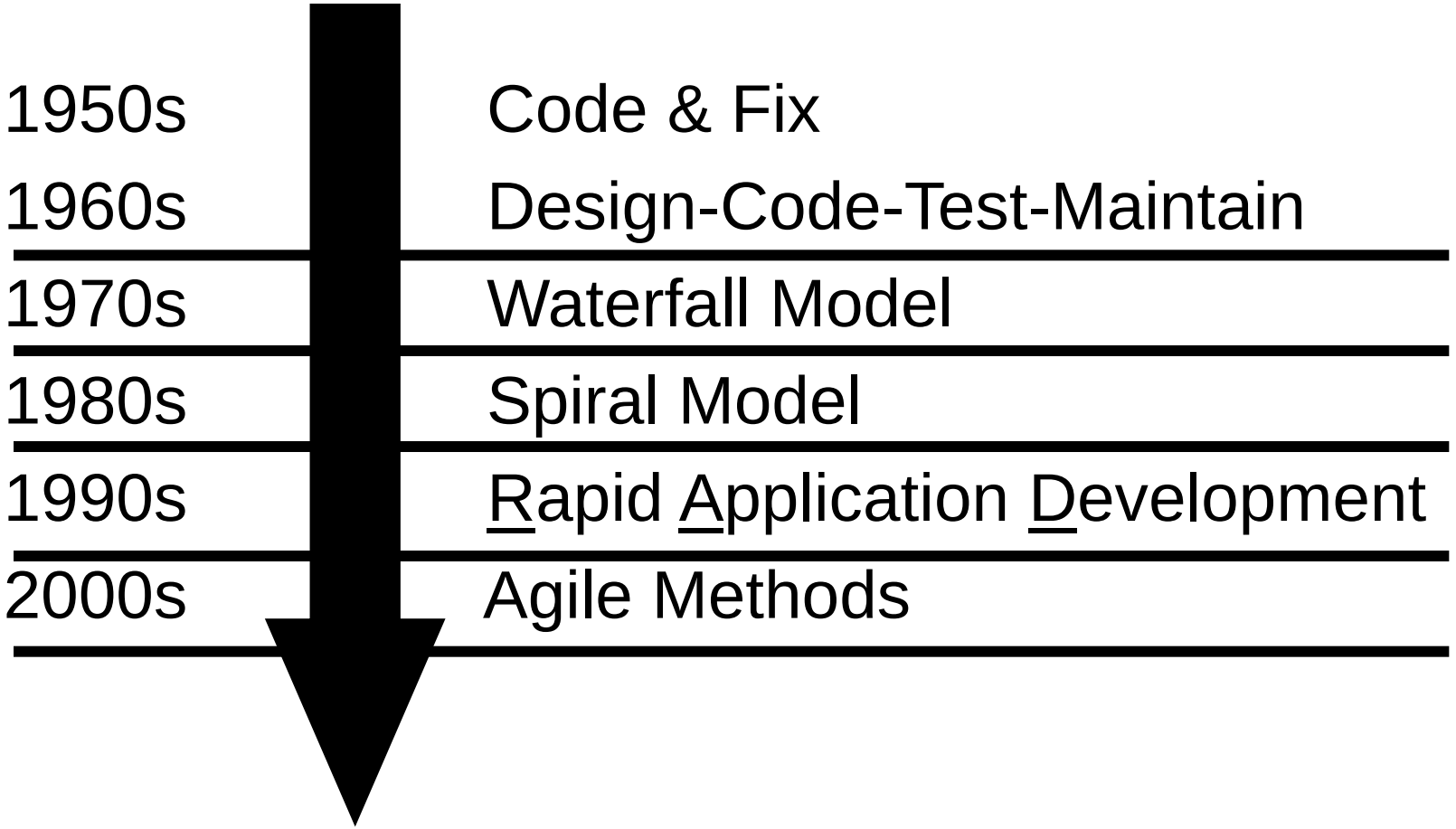
13. SOFTWARE DEVELOPMENT METHODOLOGIES 1

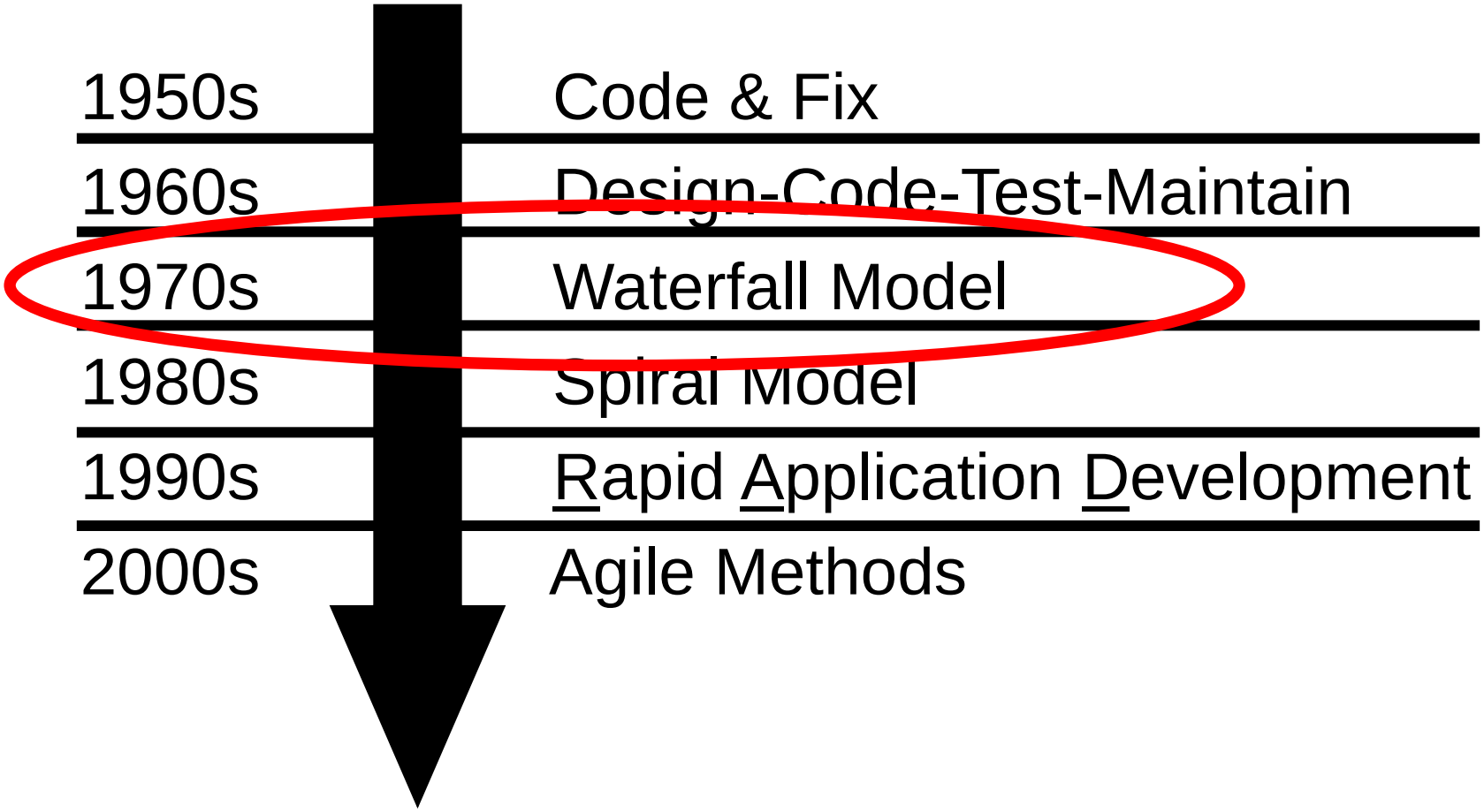
Outline of today's Lecture

- Timeline of methodologies
- Waterfall model
- Other methodologies

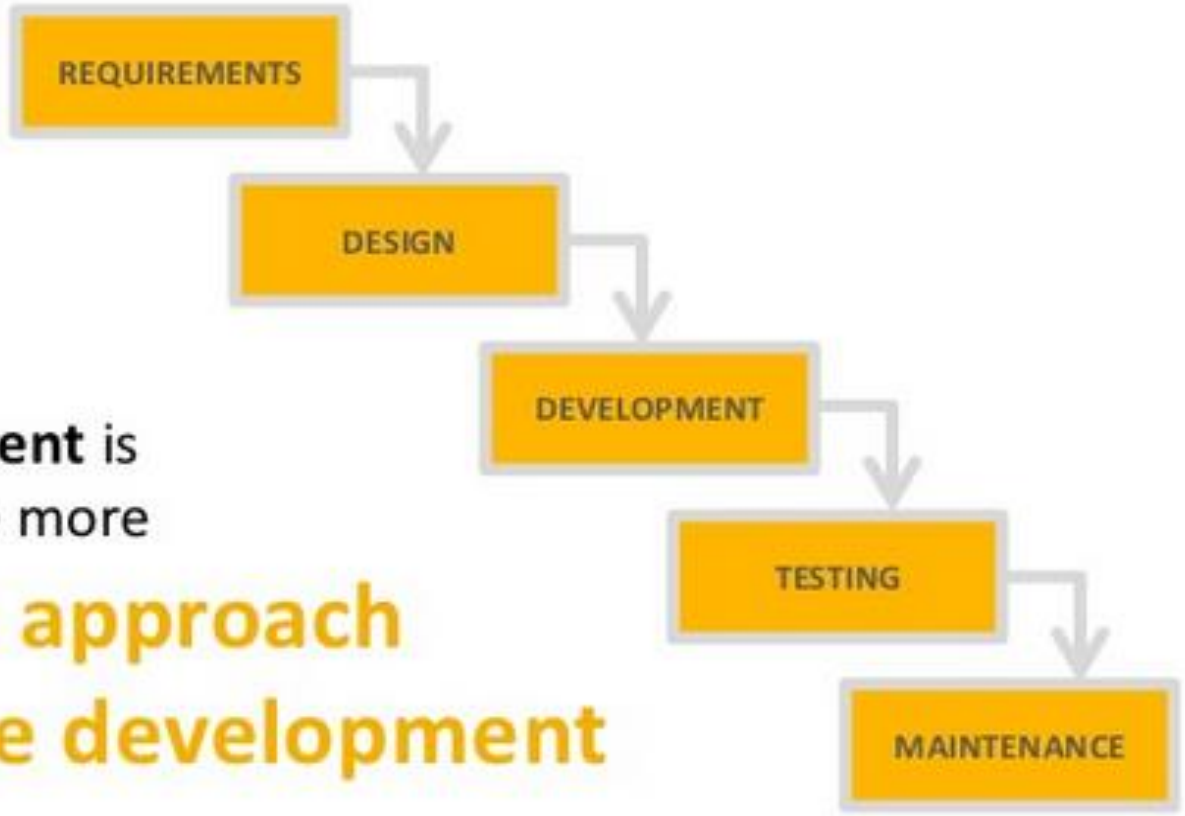
Timeline of Methodologies



Timeline of Methodologies



Waterfall Development



Waterfall Development is another name for the more

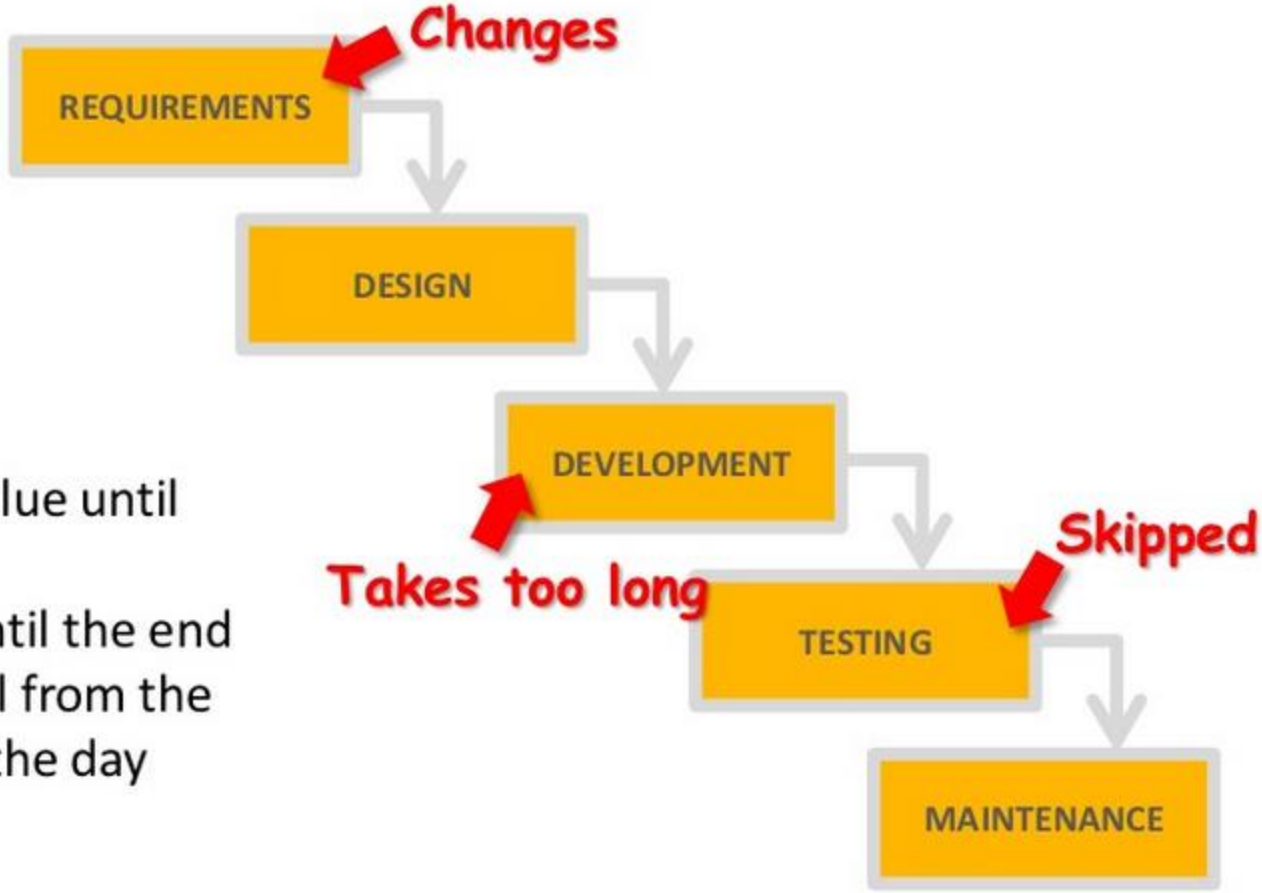
traditional approach to software development

Waterfall Development (contd..)

You **complete one phase** (e.g. design) **before** moving on to the **next phase** (e.g. development)

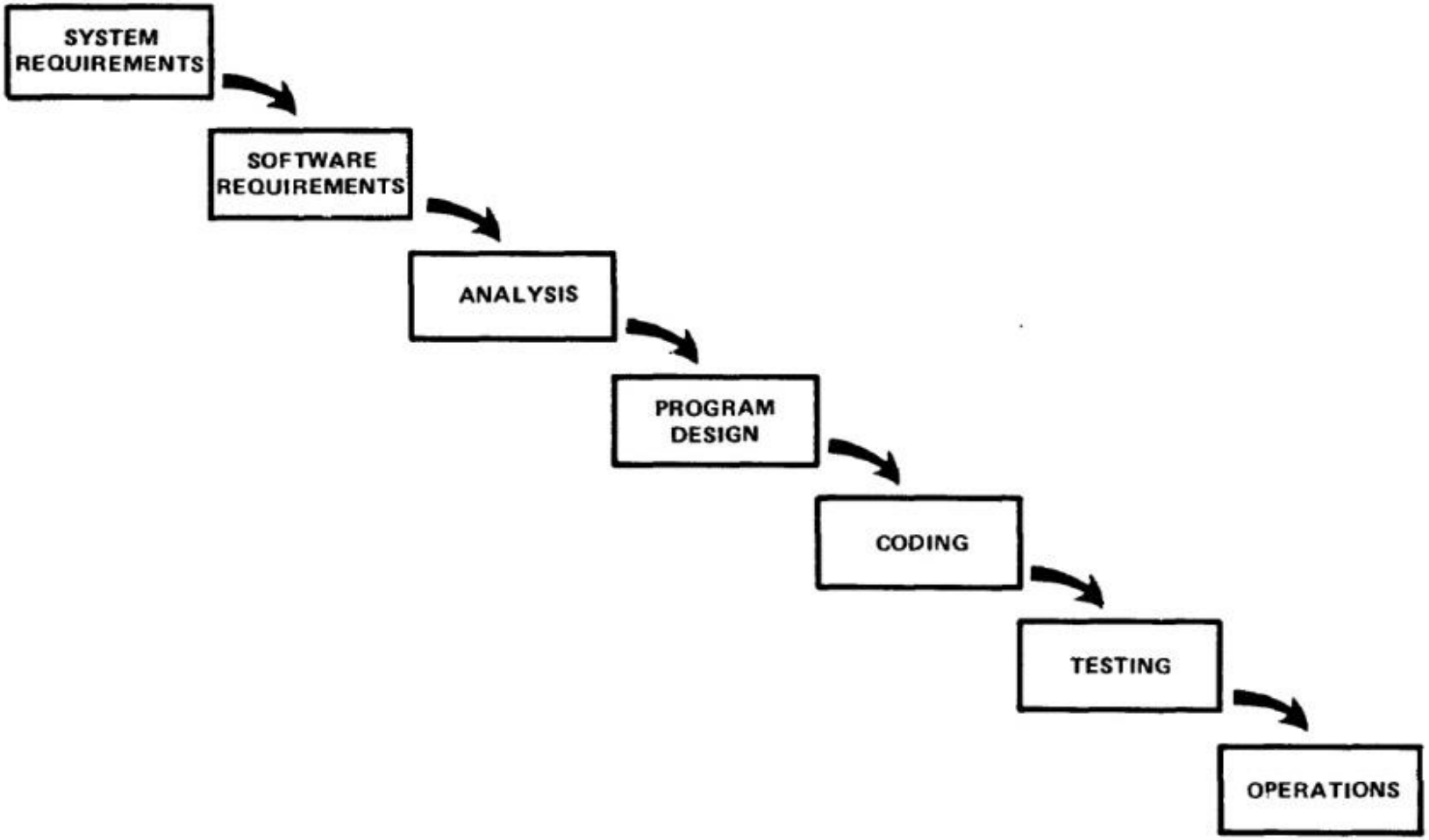
You **rarely aim to re-visit a 'phase' once it's completed**. That means, you **better get whatever you're doing right the first time!**

But...



- You don't realize any value until the end of the project
- You leave the testing until the end
- You don't seek approval from the stakeholders until late in the day

This approach is **highly risky**, often more **costly** and generally **less efficient** than **Agile** approaches



- Royce, W.W., 1970, "*Managing the Development of Large Software Systems*", Proceedings of IEEE WESCON 26 (August), pp.1–9.

Winston W. Royce

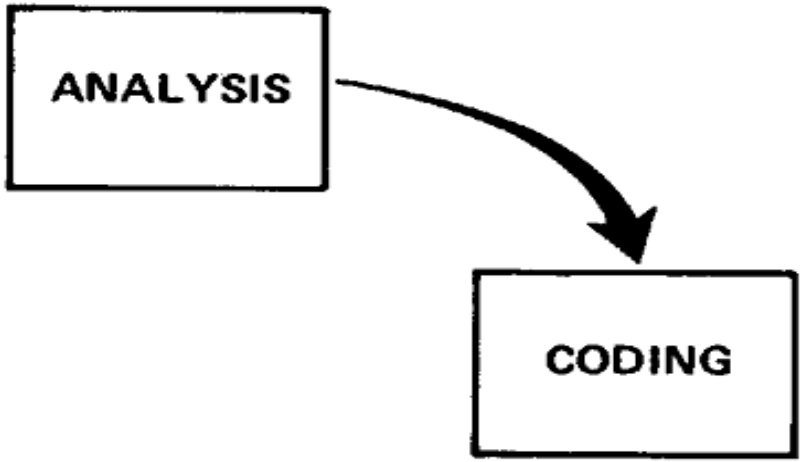
- Born in 1929.
- Died in 1995.
- An **American computer scientist**, director at Lockheed Software Technology Centre in Austin, Texas, and one of the leaders in software development in the second half of the 20th century.
- He was the **first person to describe the “Waterfall model” for software development**, although Royce did not use the term "waterfall" in that article, nor advocated the waterfall model as a working methodology.

Introduction

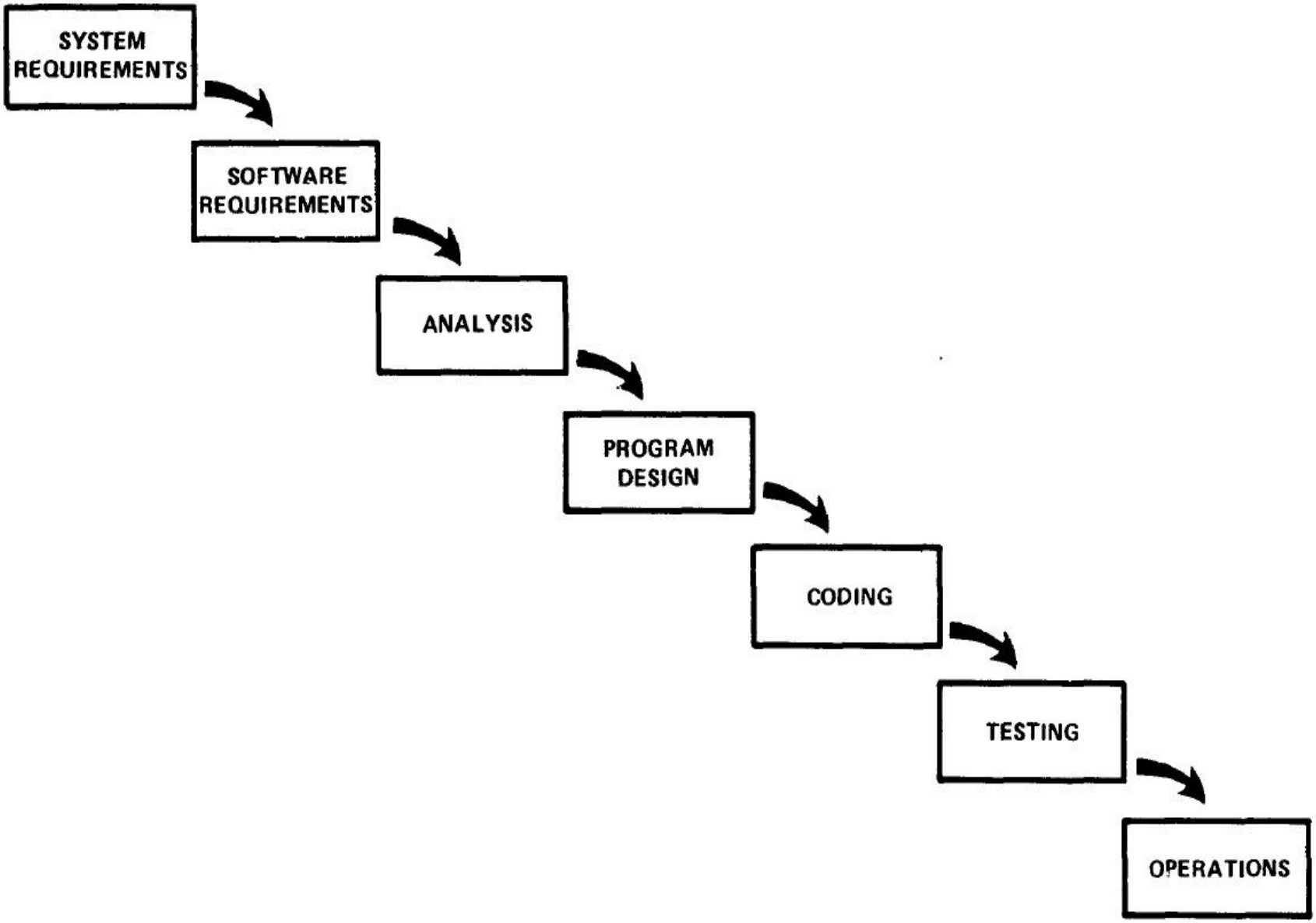
- *"I am going to describe my personal views about managing large software developments.*
- *I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis.*
- *In these assignments I have experienced **different degrees of success** with respect to arriving at an operational state, on-time, and within costs.*
- *I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.*"

Small Developments

- For a small development, you only need the following steps
  - Typically done for programs for internal use



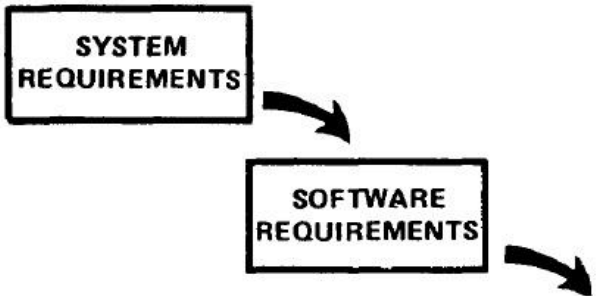
Large Developments



Large Developments

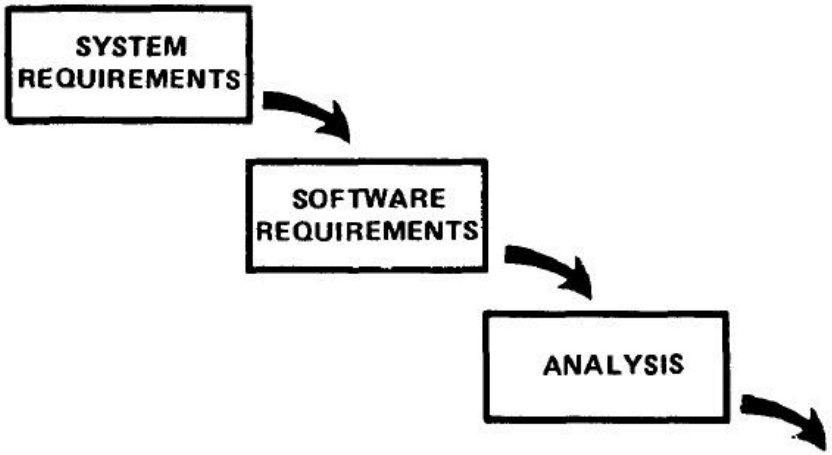
- **System Requirements:** Identify, select and document functional, scheduling and financial requirements.

Large Developments



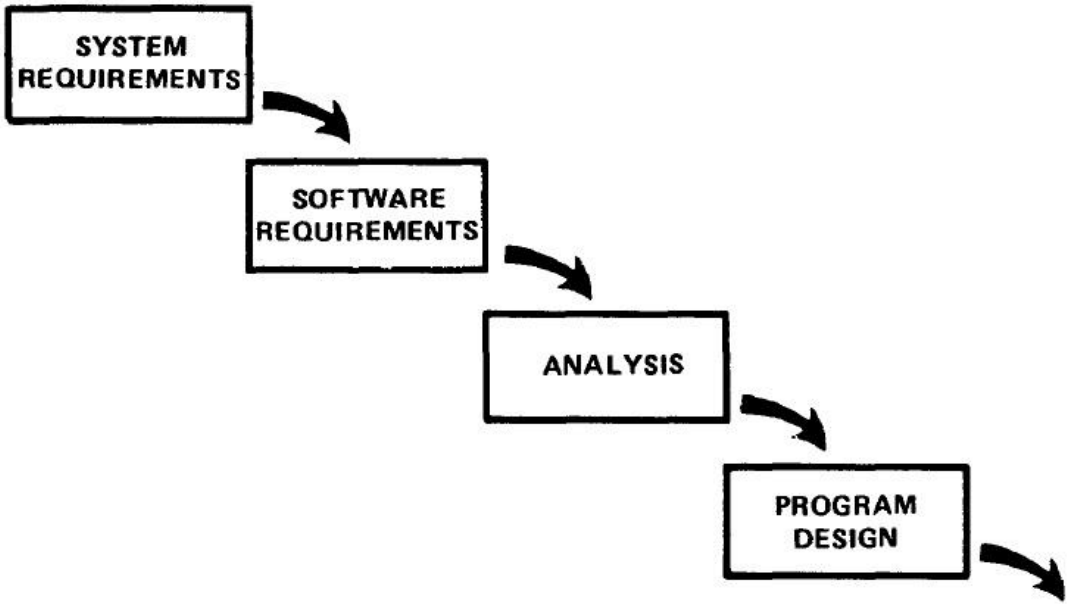
- **Software Requirements:** Identify, select and document the software features necessary to satisfy the system requirements.

Large Developments



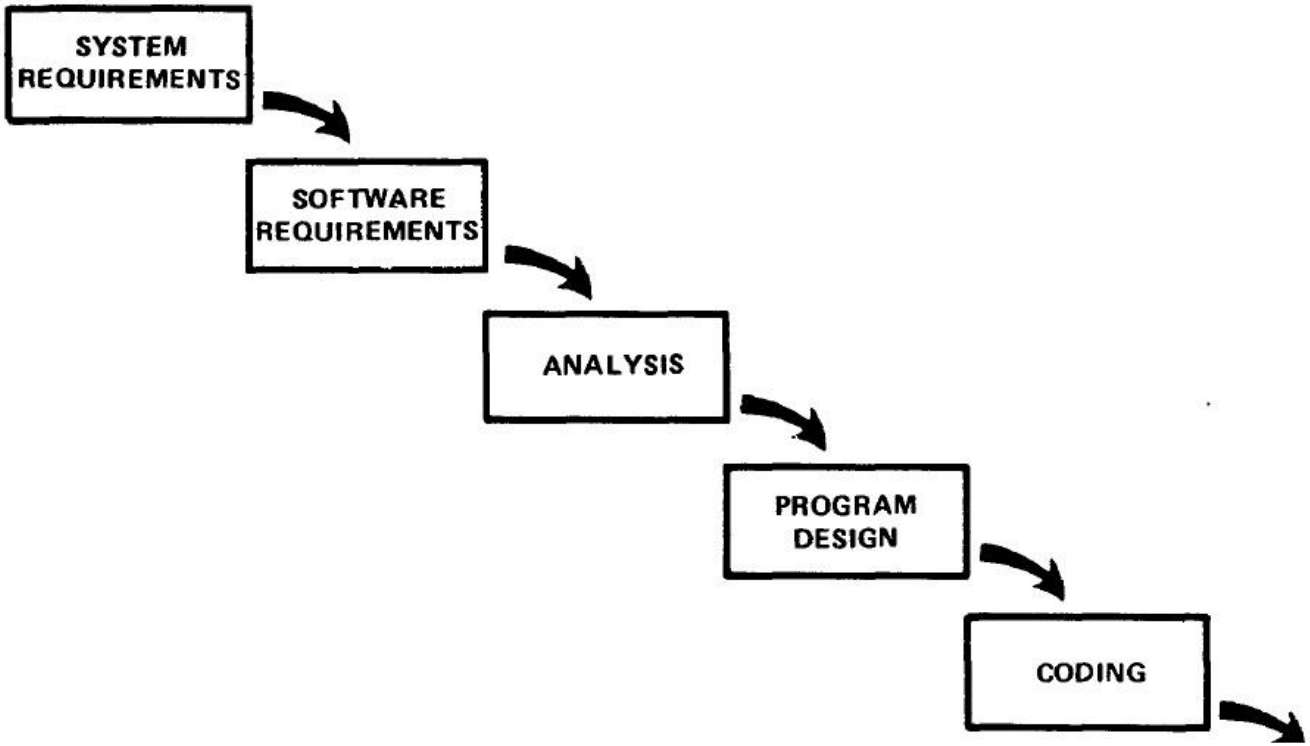
- **Analysis:** Methodically work through the details of each requirement.

Large Developments



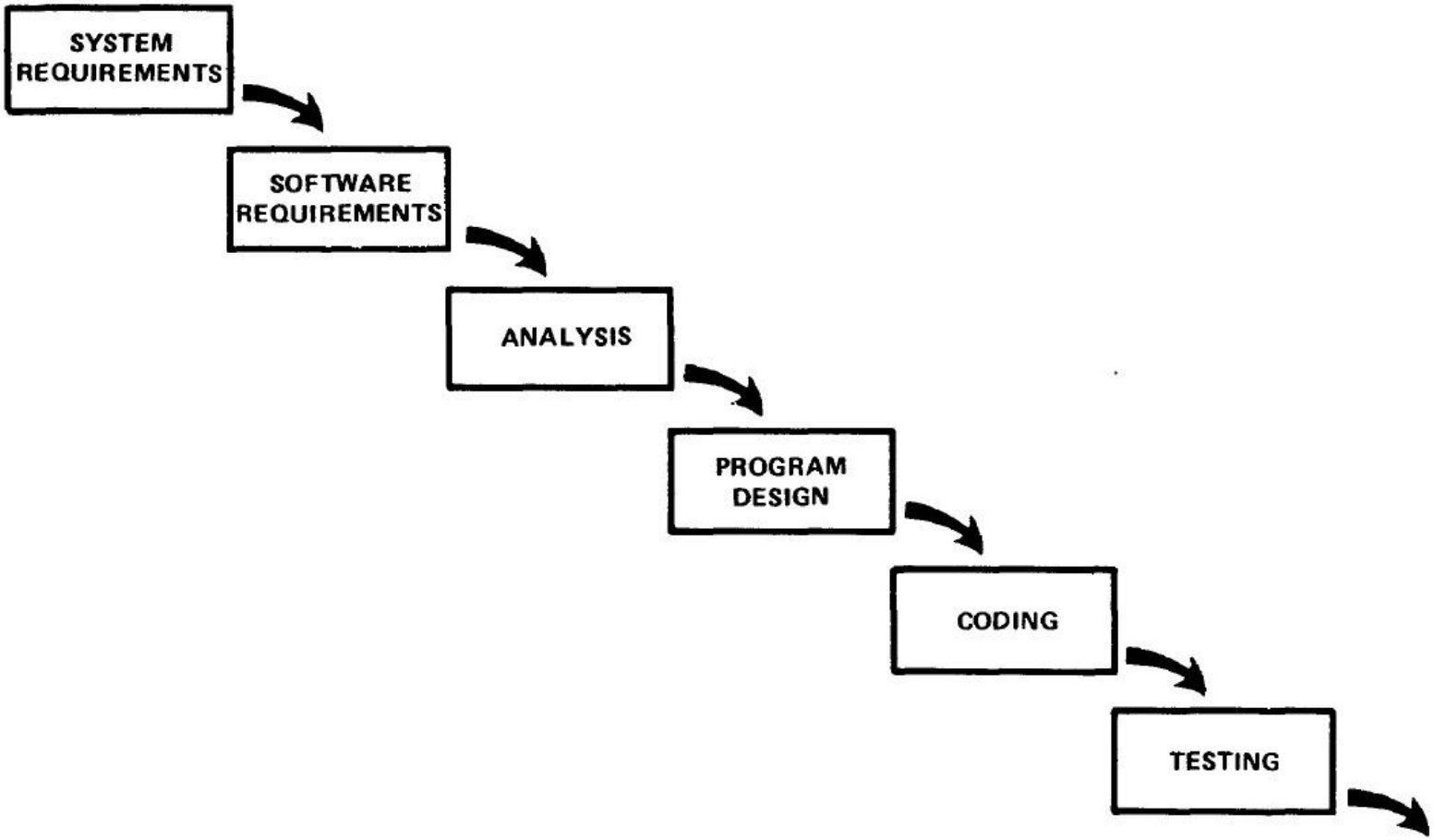
- **Program Design:** Use programming techniques to design software and hardware within the constraints and objectives set in the earlier stages.

Large Developments



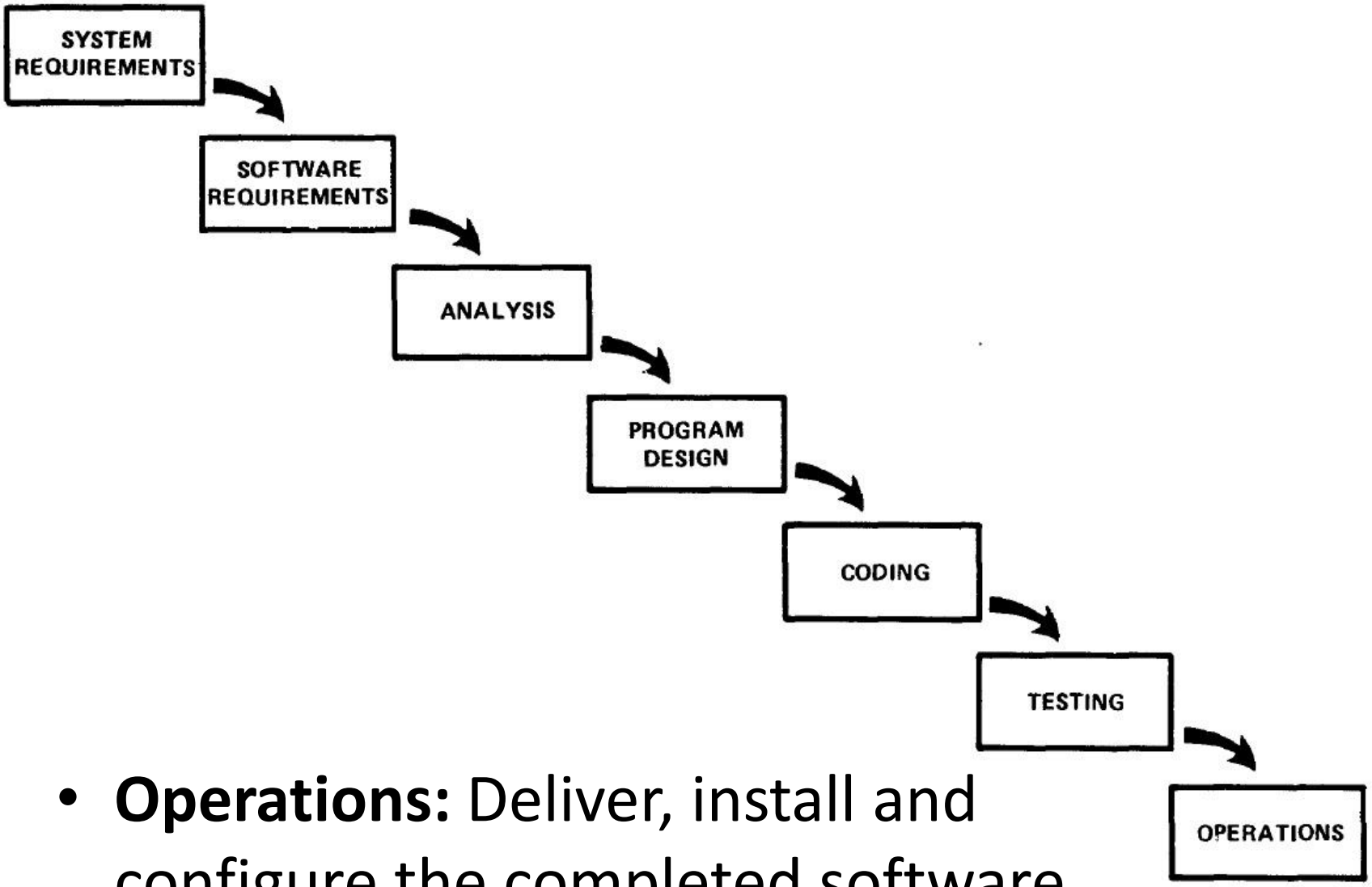
- **Coding:** Implement the program as designed in the earlier stages.

Large Developments



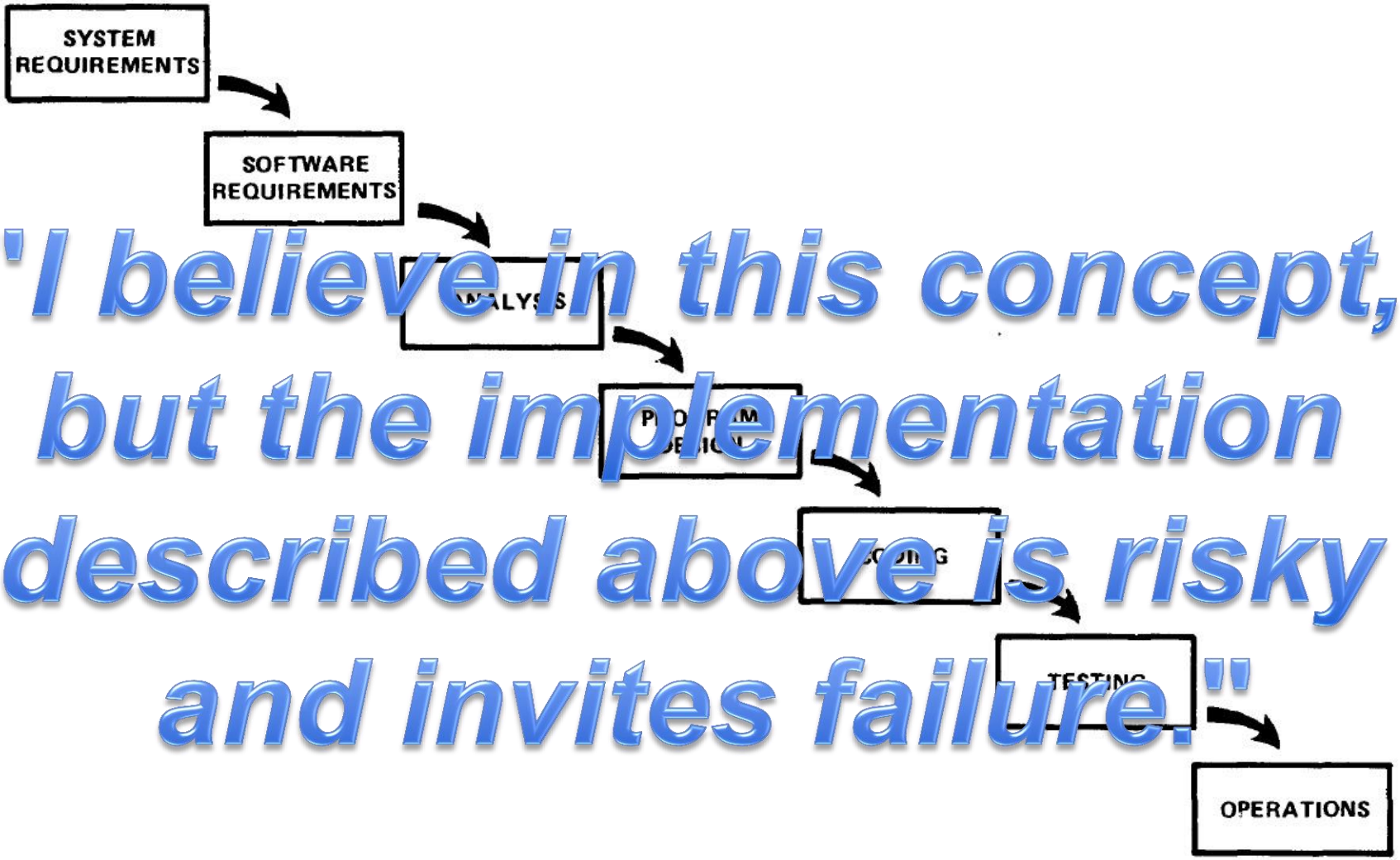
- **Testing:** Test the software and record the results.

Large Developments



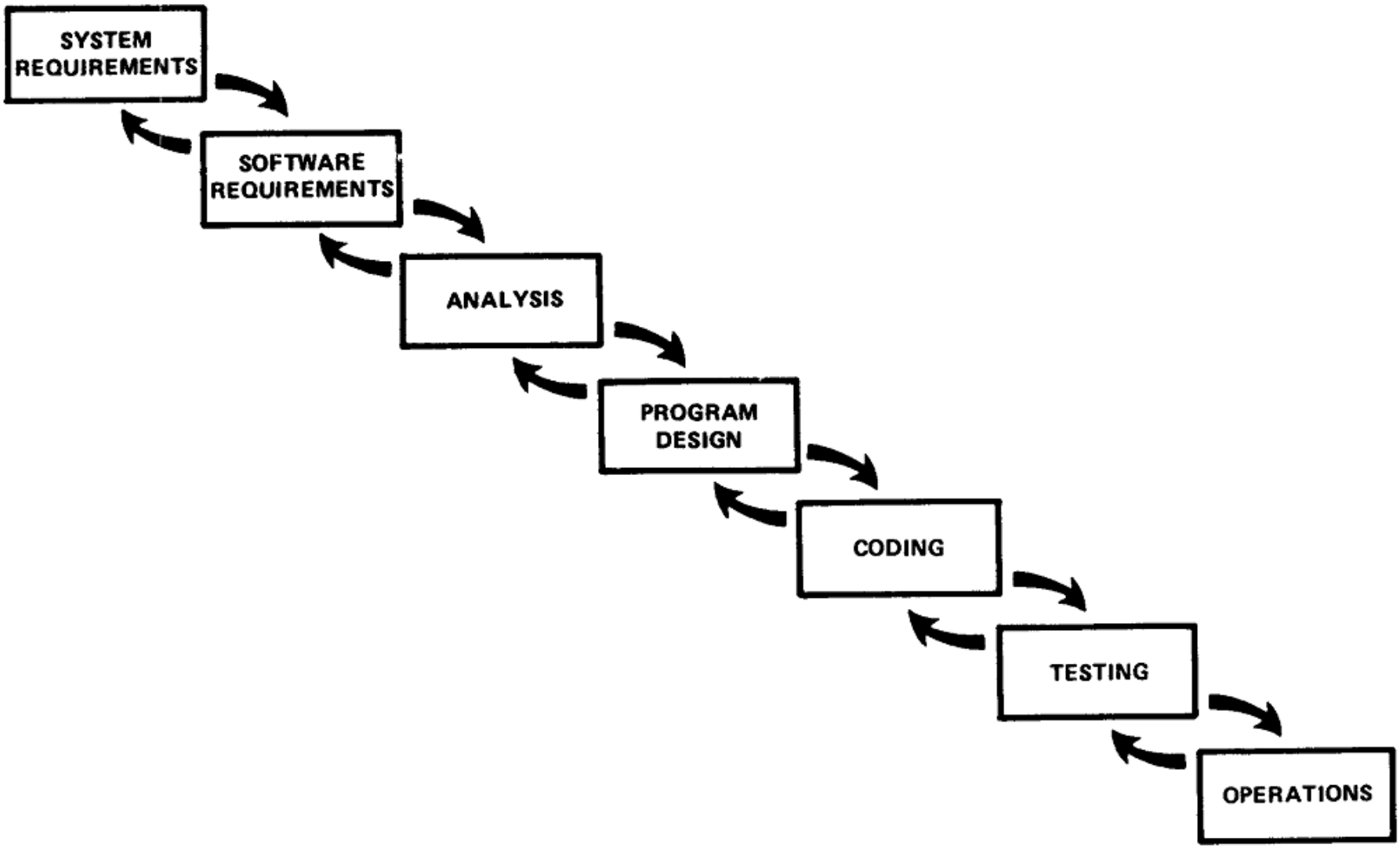
- **Operations:** Deliver, install and configure the completed software.

Large Developments



"I believe in this concept, but the implementation described above is risky and invites failure."

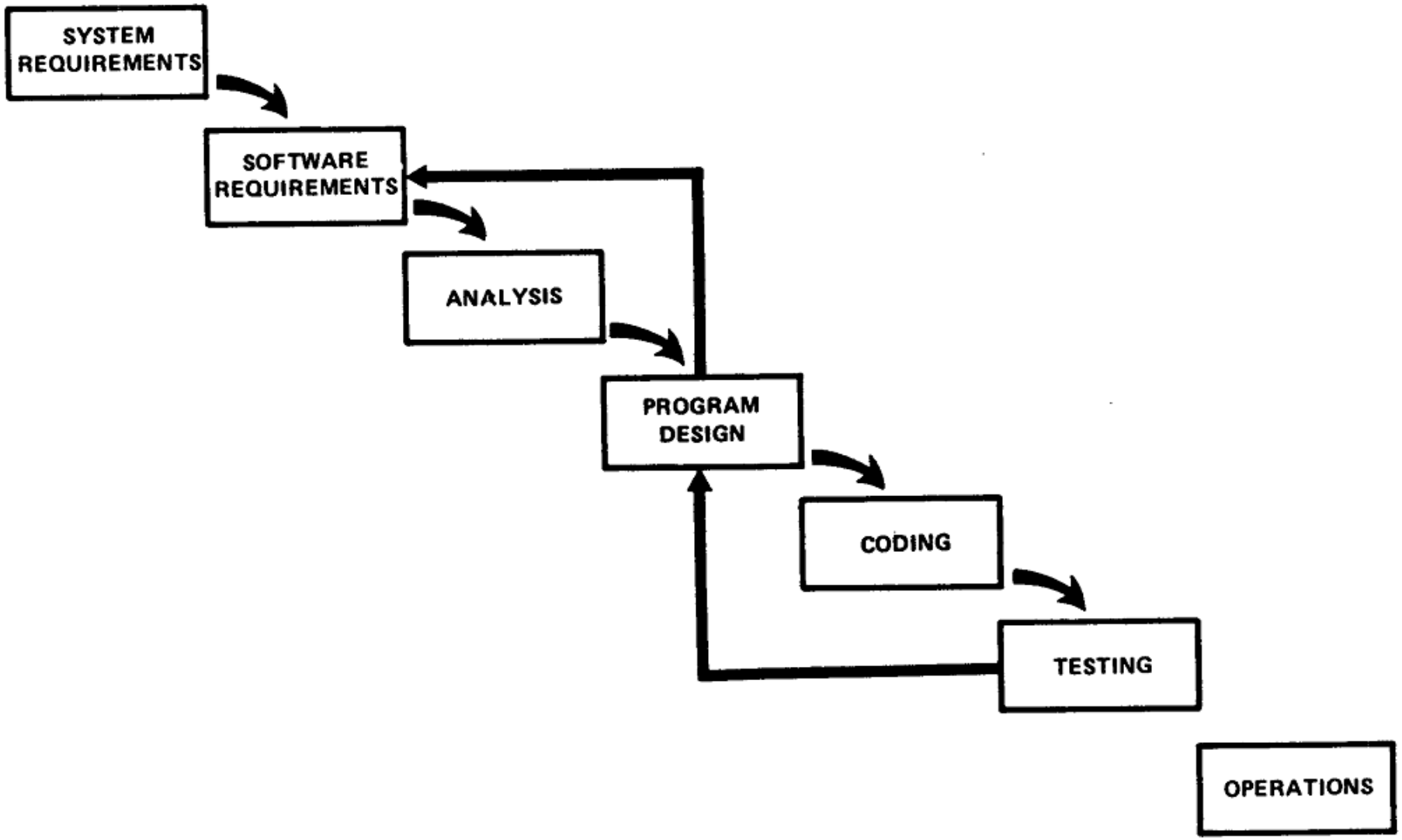
Iterative Relationship between Successive Development Phases



Iterative Relationship between Successive Development Phases

- Each step progresses and the design is further detailed, there is an **iteration with the preceding and succeeding steps** but rarely with the more remote steps in the sequence.
- The virtue of all of this is that as the design proceeds the change process is scoped down to manageable limits.

Unfortunately the design iterations are never confined to the successive steps



Unfortunately the design iterations are never confined to the successive steps

- **Unfortunately the design iterations are never confined to the successive steps**
- The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed.
- These phenomena are not precisely analyzable.
- Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required.

How do we fix this?

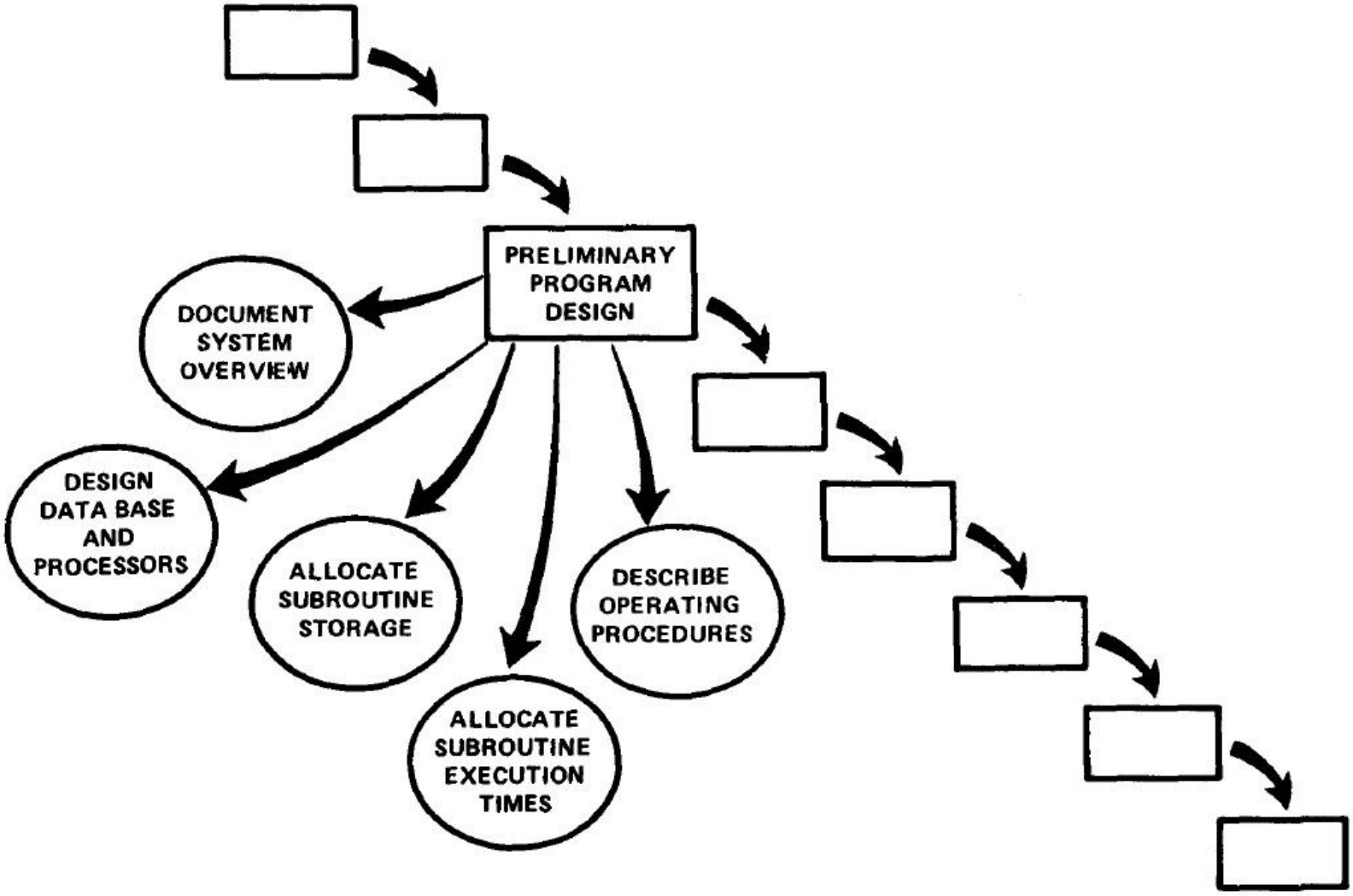
Five Steps – from the 1970's !!

1. Program Design comes first
2. Document the Design
3. Do it twice
4. Plan, Control and Monitor Testing
5. Involve the Customer

Step 1. Program Design comes first

A preliminary program design phase has been inserted between the Software Requirements Generation phase and the Analysis phase.

Step 1. Program Design comes first – cont.



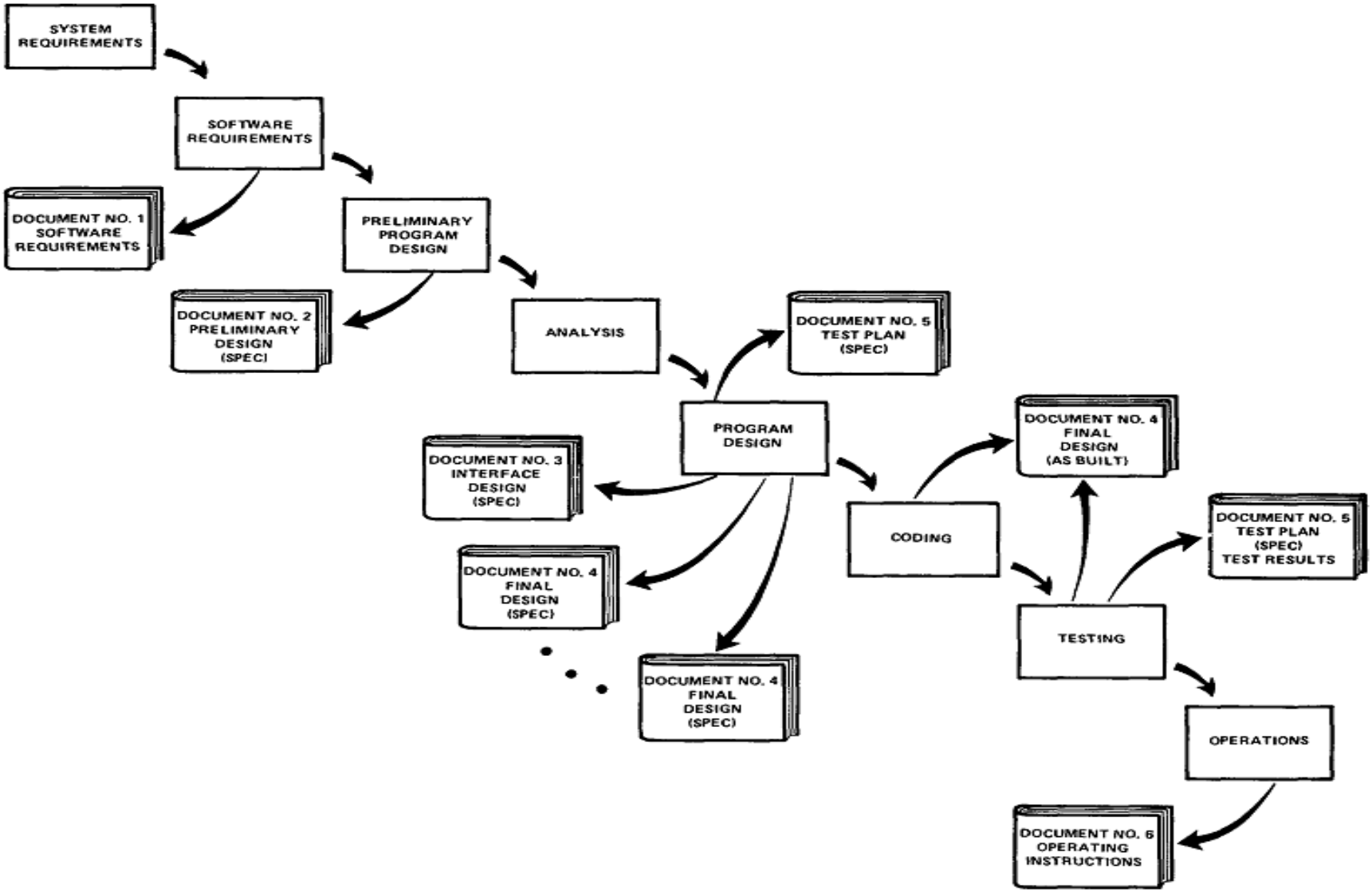
Step 1. Program Design comes first – cont.

- The following steps are required:
  1. Begin the design process with program designers, not analysts or programmers.
  2. Design, define and allocate the data processing modes.
  3. Write an overview document that is understandable, informative and current.

Step 2. Document the Design

- “How much documentation?"
- “Quite a lot"
- **More than most programmers, analysts, or program designers are willing to do if left to their own devices.**
- The first rule of managing software development is ruthless enforcement of documentation requirements.

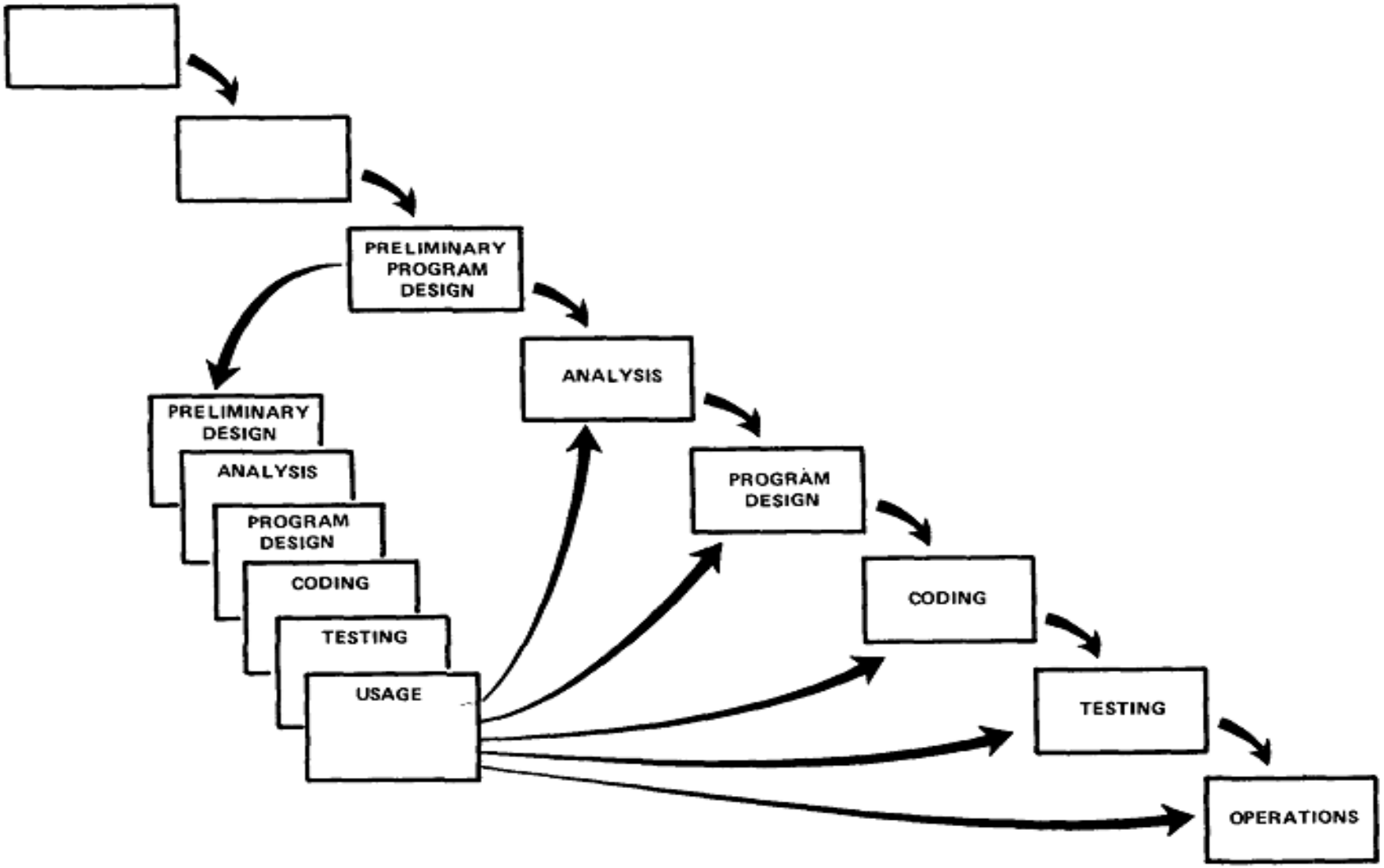
Step 2. Document the Design – cont.



Step 3. Do It Twice

- Create a **pilot** study
- If the computer program in question is being developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned.

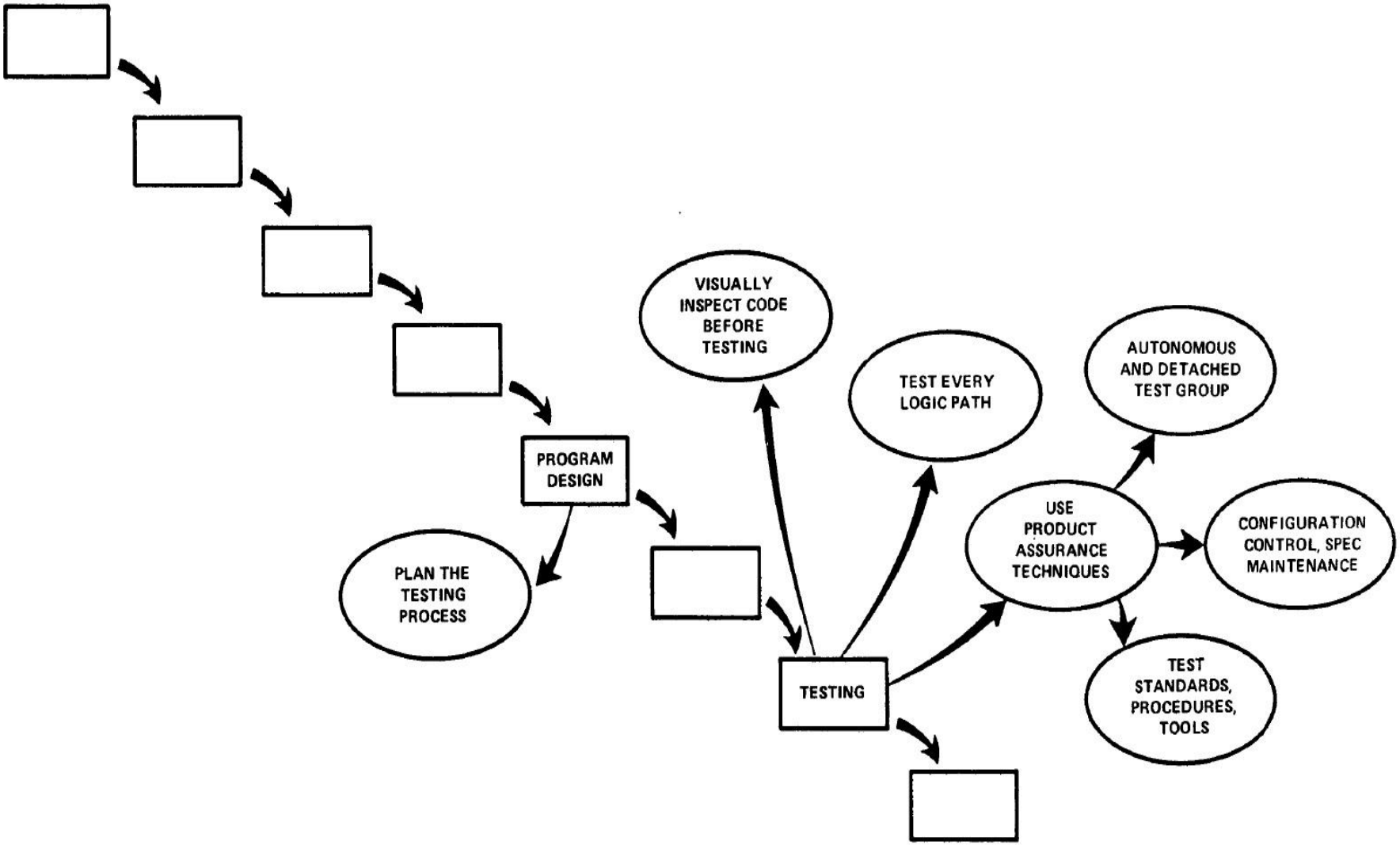
Step 3. Do It Twice – cont.



Step 4. Plan, Control and Monitor Testing

Without question the biggest user of project resources, whether it be manpower, computer time, or management judgment, is the test phase. It is the phase of greatest risk in terms of dollars and schedule.

Step 4. Plan, Control and Monitor Testing – cont.



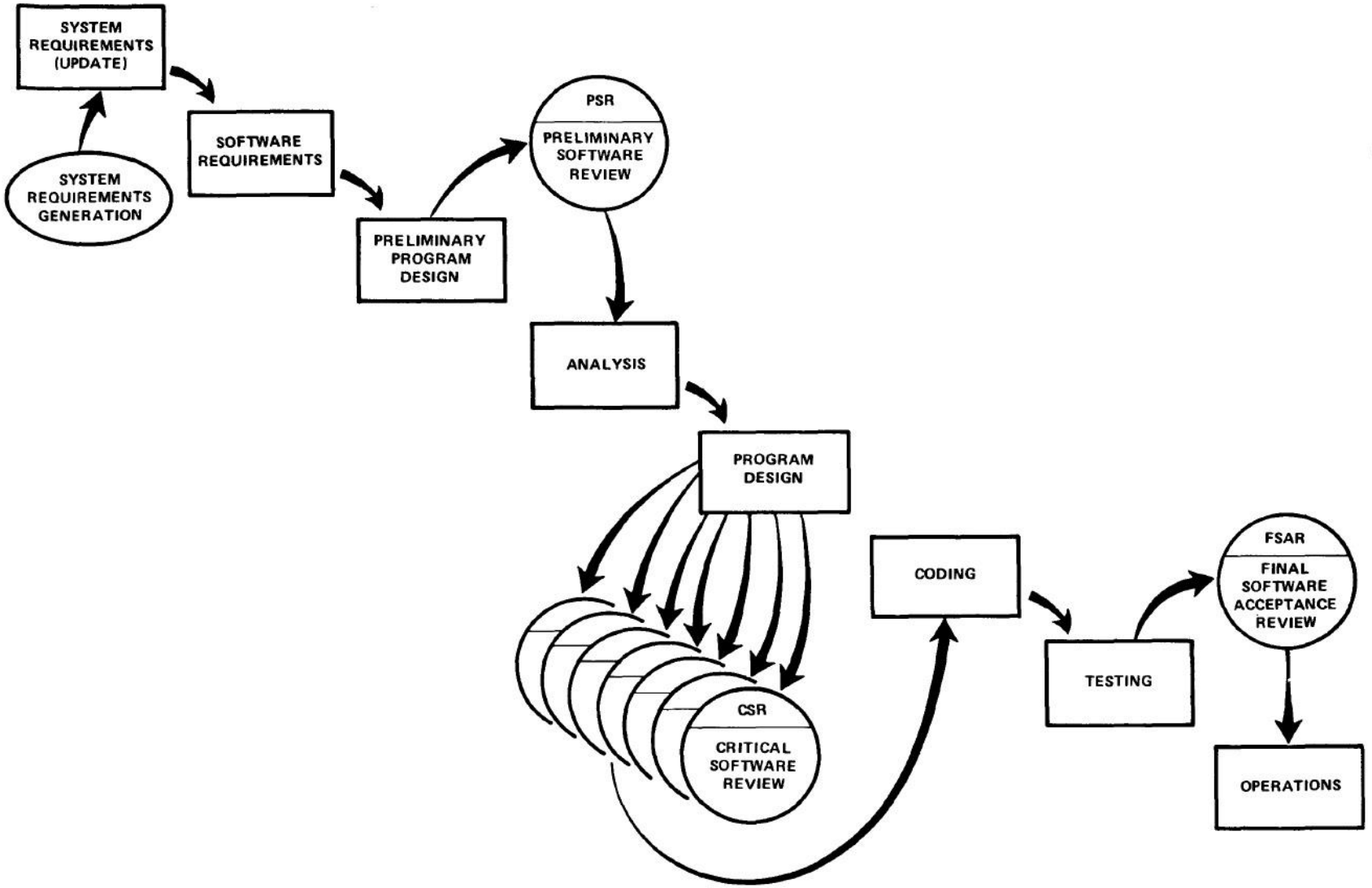
Step 4. Plan, Control and Monitor Testing – cont.

1. Many parts of the test process are best handled by test specialists who did not necessarily contribute to the original design.
2. Most errors are of an obvious nature that can be easily spotted by visual inspection.
3. Test every logic path in the computer program at least once with some kind of numerical check.
4. After the simple errors (which are in the majority, and which obscure the big mistakes) are removed, then it is time to turn over the software to the test area for checkout purposes.

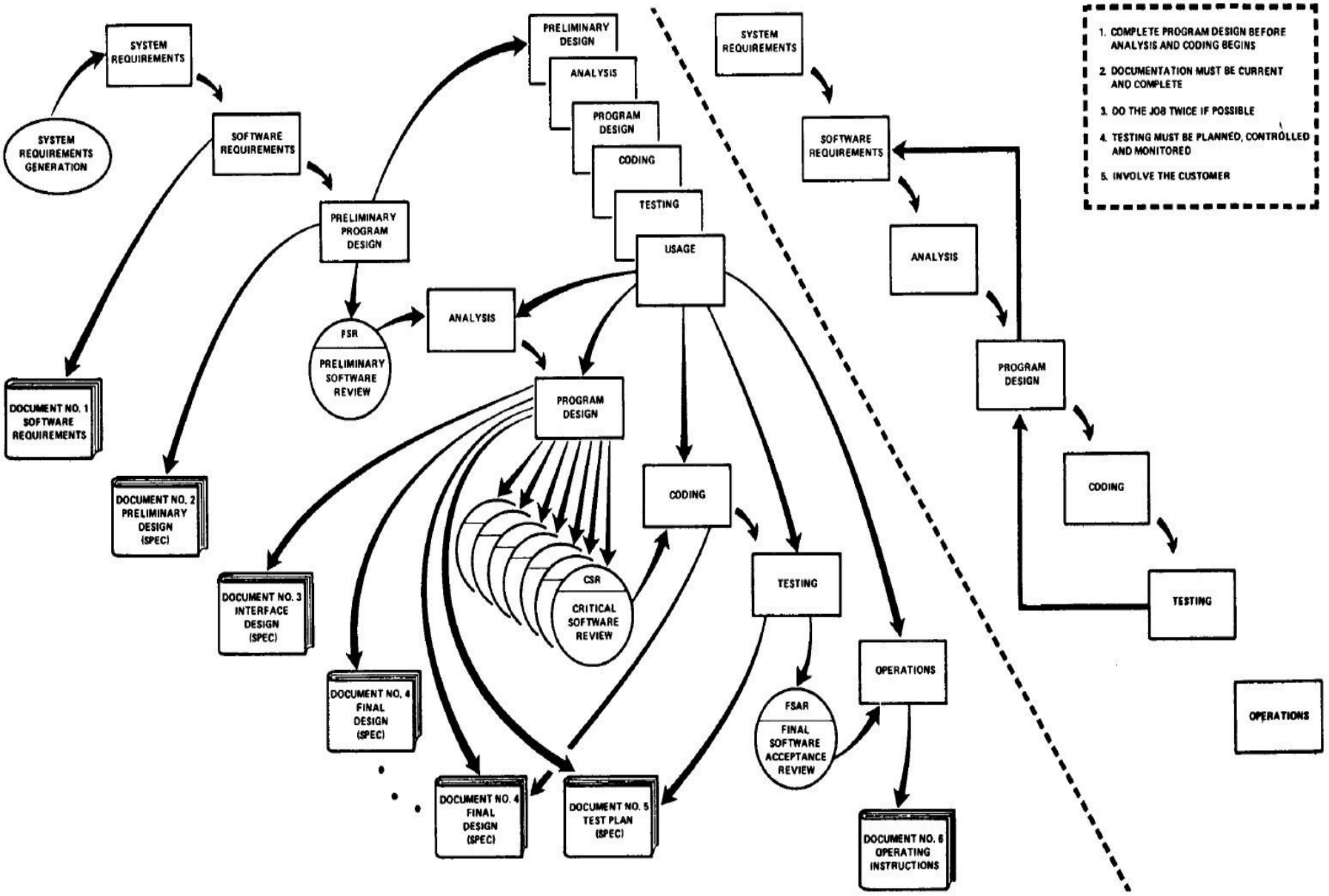
Step 5. Involve the Customer

- For some reason what a software design is going to do is subject to wide interpretation even after previous agreement.
- It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery.
- To give the contractor free rein between requirement definition and operation is inviting trouble.

Step 5. Involve the Customer – cont.



Summary


SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 5 TEST PLAN (SPEC)
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
FSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
CSR
CRITICAL SOFTWARE REVIEW
CODING
TESTING
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
1. COMPLETE PROGRAM DESIGN BEFORE ANALYSIS AND CODING BEGINS
2. DOCUMENTATION MUST BE CURRENT AND COMPLETE
3. DO THE JOB TWICE IF POSSIBLE
4. TESTING MUST BE PLANNED, CONTROLLED AND MONITORED
5. INVOLVE THE CUSTOMER

Other methodologies

- Agile Software Development
- Crystal Methods
- Dynamic Systems Development Model (DSDM)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Joint Application Development (JAD)
- Lean Development (LD)
- Rapid Application Development (RAD)
- Rational Unified Process (RUP)
- Scrum
- Spiral
- Systems Development Life Cycle (SDLC)