# COMP1531

# 🎨 Design - Software Complexity

## Lecture 9.2

Author(s): Hayden Smith

(Download as PDF)

# In This Lecture

- **Why? 🤔**
  - We need a material way to be able to understand and have conversations about how complex software is
- **What? 📰**
  - Accidental V Essential Complexity
  - Cyclomatic Complexity Measurements

# 🧠 What Is Software Complexity?

Any ideas?

# 🧠 What Is Software Complexity?

- A famous paper from 1986:
  - *No Silver Bullet – Essence and Accident in Software Engineering* by Fred Brooks
- Described software complexity by dividing it into two categories *essential* and *accidental*.
- Further conclusions of the paper are much debated

## 😎 Essential

Complexity that is inherent to the problem.

For example, if the user or client requires the program to do 30 different things, then those 30 things are essential.

## 😳 Accidental

Complexity that is not inherent to the problem.

For example, generating or parsing data in specific formats.

## 😎 Essential

Fundamentally can't be removed, but can be managed with good *software design*.

## 😳 Accidental

Can be somewhat mitigated by engineering decisions; e.g. smart use of libraries, standards, etc.

Hard to remove entirely.

# 🙉 Open Questions

- Is there a concrete process for distinguishing accidental and essential complexity?
- How much of the complexity of modern software is accidental?
- To what degree has or will accidental complexity be removed in future?

# 🥲 How Can We Measure Software Complexity?

# 🥲 How Can We Measure Software Complexity?

- Coupling
- Cohesion
- Cyclomatic Complexity

# 👯 Coupling

- A measure of how closely connected different software components are.
- Usually expressed as a simple ordinal measure of "loose" or "tight".
- For example, web applications tend to have a frontend that is loosely coupled from the backend.
- **Loose coupling is good**

#  Cohesion

- The degree to which elements of a module belong together.
- Elements belong together if they're somehow related.
- Usually expressed as a simple ordinal measure of "low" or "high".
- **High cohesion is good**
- Read more here

#  Cyclomatic Complexity

- A measure of the branching complexity of functions.
- Computed by counting the number of linearly-independent paths through a function.

#  Cyclomatic Complexity

To compute:

1. Convert function into a control flow graph
2. Calculate the value of the formula

$$V(G) = e - n + 2$$

where e is the number of edges and n is the number of nodes

# Cyclomatic Complexity

## Example 1

```
1 function foo() {
2   if (A()) {
3     B();
4   } else {
5     C();
6   }
7 }
```

$$V(G) = 4 - 4 + 2 = 2$$

#  Cyclomatic Complexity

## Example 2

```
1 function foo() {
2   if (A()) {
3     B();
4   } else {
5     if (C()) {
6       D():
7     }
8   }
9 }
```

$$V(G) = 6 - 5 + 2 = 3$$

# 👩‍🔬 Cyclomatic Complexity

## Example 3

```
1 function foo() {
2   while (A()) {
3     B();
4   }
5   C();
6 }
```

$$V(G) = 3 - 3 + 2 = 2$$

#  Cyclomatic Complexity

## Example 4

```
 1 function day_to_year(days) {
 2   let year = 1970
 3
 4   while (days > 365) {
 5     if (is_leap_year(year)) {
 6       if (days > 366) {
 7         days -= 366;
 8         year += 1;
 9       }
10     } else {
11       days -= 365;
12       year += 1;
13     }
14   }
15
16   return year;
17 }
```

$$V(G) = 8 - 6 + 2 = 4$$

#  Cyclomatic Complexity

## Example 5

```
 1 function day_to_year(days) {
 2   let year = 1970
 3
 4   while (days > 0) {
 5     if (is_leap_year(year)) {
 6       days -= 366;
 7     } else {
 8       days -= 365;
 9     }
10     year += 1;
11   }
12
13   return year - 1;
14 }
```

$$V(G) = 7 - 6 + 2 = 3$$

# 🔧 Usage

A simple understandable measure of function complexity.

Some people argue 10 should be the maximum cyclomatic complexity of a function where others argue for 8.

# 😨 Drawbacks

- Assumes non-branching statements have no complexity.
- Keeping cyclomatic complexity low encourages splitting functions up, regardless of whether that really makes the code more understandable.

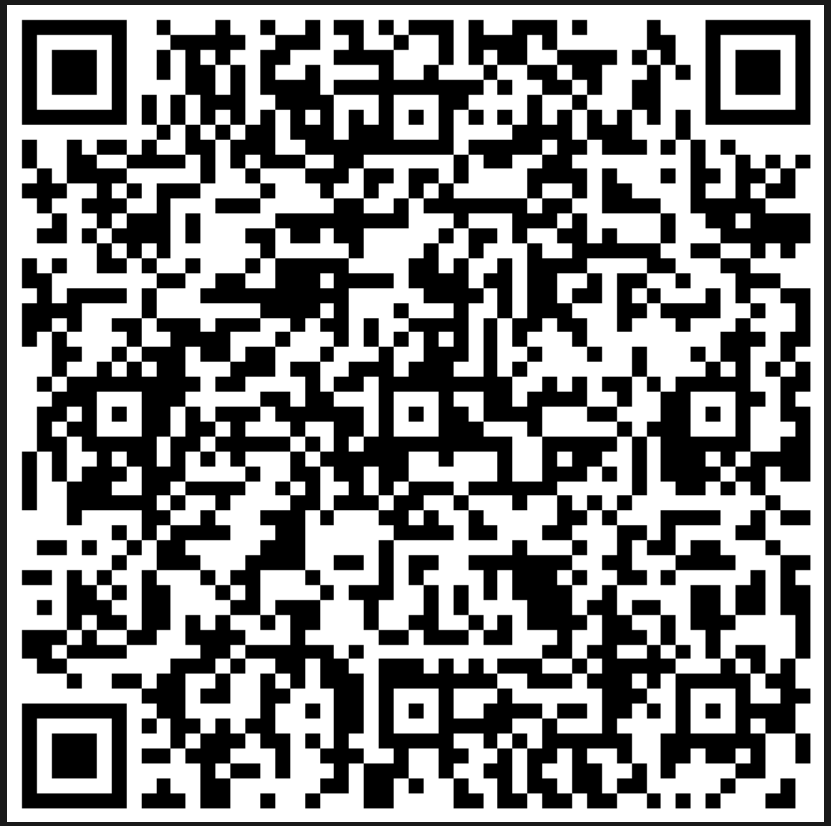
# 💻 Automatic Calculation

Depending on the programming language, sometimes there are tools that exist to automatically calculate it.

# 📚 Further Reading

- The original No Silver Bullet paper:
  - http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/no-silver-bullet.pdf
- A more modern description:
  - https://stevemcconnell.com/articles/software-engineering-principles/
- A recent rebuttal:
  - https://blog.ploeh.dk/2019/07/01/yes-silver-bullet/

# 👂 Feedback



Or go to the form here.