Complexity

Nick Sumner
wsumner@sfu.ca
Laying a foundation

- Our goal for this lecture is pretty abstract.
  - We want to talk about goals for software
  - But we aren’t going to look at much code
Laying a foundation

- Our goal for this lecture is pretty abstract.
  - We want to talk about goals for software
  - But we aren’t going to look at much code

- Instead, I want to lay a foundation that you should keep in mind consistently as we consider code throughout the course.
Why do we care about software complexity?

- What even is software complexity?
Why do we care about software complexity?

- What even is software complexity?
- What is the goal of a software engineer?
Why do we care about software complexity?

- What even *is* software complexity?
- What is the goal of a software engineer? [Steve Tockey, Construx]
  
  Engineering = Scientific Theory + Practice + Engineering Economy
Why do we care about software complexity?

- What even is software complexity?

- What is the goal of a software engineer?

    Engineering = Scientific Theory + Practice + Engineering Economy
    (Software) Engineering = Computer Science + Practice + Engineering Economy

[Steve Tockey, Construx]
Why do we care about software complexity?

- What even is software complexity?

- What is the goal of a software engineer? [Steve Tockey, Construx]
  
  Engineering = Scientific Theory + Practice + Engineering Economy
  (Software) Engineering = Computer Science + Practice + Engineering Economy
Why do we care about software complexity?

- What even is software complexity?
- What is the goal of a software engineer?  
  
  Engineering = Scientific Theory + Practice + Engineering Economy  
  (Software) Engineering = Computer Science + Practice + Engineering Economy  

- A good engineer needs to develop economical solutions.
Why do we care about software complexity?

- What even is software complexity?

- What is the goal of a software engineer? [Steve Tockey, Construx]
  
  Engineering = Scientific Theory + Practice + Engineering Economy
  
  (Software) Engineering = Computer Science + Practice + Engineering Economy

- A good engineer needs to develop economical solutions.
  - ↓ maintenance costs
  - ↓ defect rates
  - ↓ legal liabilities
  - ↑ extensibility & reuse for new requirements
Why do we care about software complexity?

- What even is software complexity?

- What is the goal of a software engineer? [Steve Tockey, Construx]
  
  Engineering = Scientific Theory + Practice + Engineering Economy
  
  (Software) Engineering = Computer Science + Practice + Engineering Economy

- A good engineer needs to develop economical solutions.
  - ↓ maintenance costs
  - ↓ defect rates
  - ↓ legal liabilities
  - ↑ extensibility & reuse for new requirements

- Our intuition may capture these, but software complexity is nuanced
Good engineers must exercise judgment

- Every problem has multiple solutions
Good engineers must exercise judgment

- Every problem has multiple solutions
- Good software engineering requires *evaluating* several forms of costs across many *different solutions* and *choosing* a cost effective solution
Good engineers must exercise judgment

- Every problem has multiple solutions
- Good software engineering requires **evaluating** several forms of costs across many **different solutions** and choosing a cost effective solution
- Different solutions may be **functionally** equivalent but the **nonfunctional** attributes can determine what is appropriate for a specific problem
Good engineers must exercise judgment

- Every problem has multiple solutions
- Good software engineering requires evaluating several forms of costs across many different solutions and choosing a cost effective solution
- Different solutions may be functionally equivalent but the nonfunctional attributes can determine what is appropriate for a specific problem
  - May differ radically in performance, maintainability, etc.
  - A good solution for one problem may be disastrous for another
Good engineers must exercise judgment

- Every problem has multiple solutions
- Good software engineering requires *evaluating* several forms of costs across many *different solutions* and choosing a cost effective solution
- Different solutions may be *functionally* equivalent but the *nonfunctional* attributes can determine what is appropriate for a specific problem
  - May differ radically in performance, maintainability, etc.
  - A good solution for one problem may be disastrous for another
  - Need to perform cost/benefit analysis of different solutions
Good engineers must exercise judgment

- Every problem has multiple solutions
- Good software engineering requires evaluating several forms of costs across many different solutions and choosing a cost effective solution
- Different solutions may be functionally equivalent but the nonfunctional attributes can determine what is appropriate for a specific problem
  - May differ radically in performance, maintainability, etc.
  - A good solution for one problem may be disastrous for another
  - Need to perform cost/benefit analysis of different solutions
- A modern classic example is monolith vs microservices
The ravages of time

- Worse still, costs must be considered *over time*
The ravages of time

- Worse still, costs must be considered *over time*
  - A low cost immediate solution may be expensive to live with
  - As much as we try to avoid it, requirements evolve and change
The ravages of time

• Worse still, costs must be considered **over time**
  – A low cost immediate solution may be expensive to live with
  – As much as we try to avoid it, requirements evolve and change

• **But can’t our process include refactoring and redesign?**
  – In theory
  – In practice, to a limit
  – Much of the code in a bad design must be lived with & worked around
The ravages of time

- Worse still, costs must be considered *over time*
  - A low cost immediate solution may be expensive to live with
  - As much as we try to avoid it, requirements evolve and change

- But can’t our process include refactoring and redesign?
  - In theory
  - In practice, to a limit
  - Much of the code in a bad design must be lived with & worked around

- **Good judgment involves writing code that can cope with evolution**
The complexities we will not consider

- Complexity has many sources.
The complexities we will not consider

- Complexity has many sources.
  - Design and code is only one of them, but it will be our focus
  - Just as important (maybe more) are requirements
  - Clients often say they want A when they want B
The complexities we will not consider

- Complexity has many sources.
  - Design and code is only one of them, but it will be our focus
  - Just as important (maybe more) are requirements
  - Clients often say they want A when they want B

- Requirements engineering & elicitation are more out of scope for us
  - Supposedly CMPT 475 dives into those?
The complexities we will not consider

- Complexity has many sources.
  - Design and code is only one of them, but it will be our focus
  - Just as important (maybe more) are requirements
  - Clients often say they want A when they want B

- Requirements engineering & elicitation are more out of scope for us
  - Supposedly CMPT 475 dives into those?

- But I will still change requirements on you deliberately
So what *is* complexity?

- If we want to judge and assess it, it would be nice to define it but...
So what *is* complexity?

- If we want to judge and assess it, it would be nice to define it but... we don’t have a single good answer. It is openly researched & debated.
So what is complexity?

- If we want to judge and assess it, it would be nice to define it but... we don’t have a single good answer. It is openly researched & debated.

- The goal is to capture the idea that software is hard to work with.
So what is complexity?

- If we want to judge and assess it, it would be nice to define it but... we don’t have a single good answer. It is openly researched & debated.
- The goal is to capture the idea that software is hard to work with.
- There are some classic definitions & even tools to check them.
A classic measure available in tools is McCabe or cyclomatic complexity.
Classic McCabe & Halstead measures

- A classic measure available in tools is McCabe or cyclomatic complexity
  - Idea: complexity may be about the number of independent behaviors
A classic measure available in tools is McCabe or cyclomatic complexity.

- Idea: complexity may be about the number of independent behaviors.
- So count the *linearly independent paths* through a program. (Each path has at least one unique edge.)
A classic measure available in tools is **McCabe** or **cyclomatic complexity**

- Idea: complexity may be about the number of independent behaviors
- So count the **linearly independent paths** through a program. (each path has at least one unique edge)

```
if c1:
  ...(x)
else:
  ...(y)
if c2:
  ...(z)
```

Consider the **control flow graph**
Classic McCabe & Halstead measures

- A classic measure available in tools is McCabe or cyclomatic complexity
  - Idea: complexity may be about the number of independent behaviors
  - So count the *linearly independent paths* through a program.
    (each path has at least one unique edge)

```python
if c1:
    ...(x)
else:
    ...(y)
if c2:
    ...(z)
```

- Consider the *control flow graph*
A classic measure available in tools is *McCabe* or *cyclomatic complexity*

- Idea: complexity may be about the number of independent behaviors
- So count the *linearly independent paths* through a program.
  (each path has at least one unique edge)

Consider the *control flow graph*
Classic McCabe & Halstead measures

- A classic measure available in tools is *McCabe* or *cyclomatic complexity*
  - Idea: complexity may be about the number of independent behaviors
  - So count the *linearly independent paths* through a program. (each path has at least one unique edge)

```
if c1:
  ...(x)
else:
  ...(y)
if c2:
  ...(z)
```

- Consider the *control flow graph*
A classic measure available in tools is *McCabe* or *cyclomatic complexity*

- Idea: complexity may be about the number of independent behaviors
- So count the *linearly independent paths* through a program. (each path has at least one unique edge)

```
if c1:
    ...(x)
else:
    ...(y)
if c2:
    ...(z)
```

- Consider the *control flow graph*
- \( M = \text{Edges} - \text{Nodes} + 2 \times \text{Connected Components} \)
A classic measure available in tools is McCabe or cyclomatic complexity

- Idea: complexity may be about the number of independent behaviors
- So count the linearly independent paths through a program. (each path has at least one unique edge)

Consider the control flow graph

\[ M = \text{Edges} - \text{Nodes} + 2 \times \text{Connected Components} \]

\[ M = 7 - 6 + 2 \times 1 = 3 \]
A classic measure available in tools is McCabe or cyclomatic complexity
- Idea: complexity may be about the number of independent behaviors
- So count the *linearly independent paths* through a program. (each path has at least one unique edge)

**Halstead complexity instead applies physics metaphors over**
- Distinct # operators
- Distinct # operands
- Total # operators
- Total # operands
Classic McCabe & Halstead measures

- A classic measure available in tools is *McCabe* or *cyclomatic* complexity
  - Idea: complexity may be about the number of independent behaviors
  - So count the *linearly independent paths* through a program.
    (each path has at least one unique edge)

- Halstead complexity instead applies physics metaphors over
  - Distinct # operators
  - Distinct # operands
  - Total # operators
  - Total # operands

- These are easily automated & some companies use them. Are they good?
  - Well, not really
Classic McCabe & Halstead measures

- McCabe & Halstead metrics *mostly just measure function size*
  - There is a bit more going on, but its utility is not considered cost effective
Classic McCabe & Halstead measures

- McCabe & Halstead metrics mostly just measure function size
  - There is a bit more going on, but its utility is not considered cost effective
- They also have counterintuitive scenarios

```c
void foo() {
    if (c1) { m } else { n }
    if (c2) { o } else { p }
    if (c3) { q } else { r }
    if (c4) { s } else { t }
    return;
}
```

\[ M = 16 - 13 + 2*1 = 5 \]
Classic McCabe & Halstead measures

- McCabe & Halstead metrics mostly just measure function size
  - There is a bit more going on, but its utility is not considered cost effective

- They also have counterintuitive scenarios

```c
void foo() {
    if (c1) { m } else { n }
    if (c2) { o } else { p }
    if (c3) { q } else { r }
    if (c4) { s } else { t }
    return;
}

M = 16 - 13 + 2*1 = 5
```

```c
void mn() { if (c1) { m } else { n } }
void op() { if (c1) { o } else { p } }
void qr() { if (c1) { q } else { r } }
void st() { if (c1) { s } else { t } }

M = 4 - 4 + 2*1 = 2
```
Classic McCabe & Halstead measures

- McCabe & Halstead metrics *mostly just measure function size*
  - There is a bit more going on, but its utility is not considered cost effective
- They also have obvious *counterintuitive scenarios*
- **In practice just using whitespace & the shape of code**
  - is as effective
  - is more intuitive for people
Classic McCabe & Halstead measures

- McCabe & Halstead metrics *mostly just measure function size*
  - There is a bit more going on, but its utility is not considered cost effective
- They also have obvious *counterintuitive scenarios*
- In practice just using whitespace & the shape of code
  - is as effective
  - is more intuitive for people
- *This is still clearly limited in meaning, so it isn’t on the track we want*
More philosophical definitions

- Being too specific may get in the way of defining a general concept
More philosophical definitions

- Being too specific may get in the way of defining a general concept
- Instead, we can start to consider it by its intuitive effects
  - Complexity grows with size
  - It also grows as pieces of a system are connected or woven together
More philosophical definitions

- Being too specific may get in the way of defining a general concept
- Instead, we can start to consider it by its intuitive effects
  - Complexity grows with size
  - It also grows as pieces of a system are connected or woven together
More philosophical definitions

- Being too specific may get in the way of defining a general concept

- Instead, we can start to consider it by its intuitive effects
  - Complexity grows with size
  - It also grows as pieces of a system are connected or woven together
  - It grows as individual clarity is muddled by the bigger picture

[Watch “Simple Made Easy” for more on this perspective]
More philosophical definitions

- Being too specific may get in the way of defining a general concept
- Instead, we can start to consider it by its intuitive effects
  - Complexity grows with size
  - It also grows as pieces of a system are connected or woven together
  - It grows as individual clarity is muddled by the bigger picture

[Watch “Simple Made Easy” for more on this perspective]

- We also have some general forms of complexity to consider
  - Inherent (essential) complexity
  - Incidental (accidental) complexity
Refining these for code

- We can consider more specific symptoms for code [Ousterhout 2018]
Refining these for code

- We can consider more specific symptoms for code [Ousterhout 2018]
  - Change Amplification
    An apparently simple change requires modifying many locations
Refining these for code

- **We can consider more specific symptoms for code** [Ousterhout 2018]
  - *Change Amplification*
    An apparently simple change requires modifying many locations
  - *Cognitive Load*
    The developer needs to know a great deal in order to complete a task
Refining these for code

- We can consider more specific symptoms for code [Ousterhout 2018]
  - *Change Amplification*
    An apparently simple change requires modifying many locations
  - *Cognitive Load*
    The developer needs to know a great deal in order to complete a task
  - *Unknown unknowns*
    Portions of code to modify for a task may be hard to identify
Refining these for code

- We can consider more specific symptoms for code [Ousterhout 2018]
  - *Change Amplification*
    An apparently simple change requires modifying many locations
  - *Cognitive Load*
    The developer needs to know a great deal in order to complete a task
  - *Unknown unknowns*
    Portions of code to modify for a task may be hard to identify

- We can then look for common causes to attack them
Refining these for code

- We can consider more specific symptoms for code [Ousterhout 2018]
  - *Change Amplification*
    An apparently simple change requires modifying many locations
  - *Cognitive Load*
    The developer needs to know a great deal in order to complete a task
  - *Unknown unknowns*
    Portions of code to modify for a task may be hard to identify

- We can then look for common causes to attack them
  - *Dependencies*
    Code cannot be understood in isolation because of relationships to other code.
Refining these for code

- We can consider more specific symptoms for code [Ousterhout 2018]
  - Change Amplification
    An apparently simple change requires modifying many locations
  - Cognitive Load
    The developer needs to know a great deal in order to complete a task
  - Unknown unknowns
    Portions of code to modify for a task may be hard to identify

- We can then look for common causes to attack them
  - Dependencies
    Code cannot be understood in isolation because of relationships to other code.
  - Obscurity
    Important information about code is not obvious.
Signs of complexity

- These may present themselves in many ways
Signs of complexity

- These may present themselves in many ways
  - Coupling
Signs of complexity

- These may present themselves in many ways
  - Coupling
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content (accessing implementation of another component)
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data

```c
int global = ...

... = global

global = ...

... = global
```
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data

Singletons have these constraints and worse.
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing

We will spend a day in the future on this.
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing
    - Temporal
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing
    - Temporal

```java
Cat cat = new Cat;
...
delete cat;
```
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing
    - Temporal

```java
Cat cat = new Cat;
...
delete cat;
```

```java
Process p;
p.doStep1();
p.doStep2();
p.doStep3();
```
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing
    - Temporal

```java
Cat cat = new Cat;
...
delete cat;
```

```java
Process p;
p.doStep1();
p.doStep2();
p.doStep3();
```

```java
Process p;
p.foo();
p.bar();
p.baz();
```

This is more insidious!
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing
    - Temporal
    - Passing data to/from each other

```python
def foo(a, b):
...
x = foo(1, 2)
```
Signs of complexity

- These may present themselves in many ways
  - Coupling
    - Content
    - Common global data
    - Subclassing
    - Temporal
    - Passing data to/from each other
    - Independence
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out

Do you agree? Why?
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
  - Layers & stratification

& a consistent, self contained view per level
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
  - Layers & stratification
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
  - Layers & stratification
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
  - Layers & stratification
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
  - Layers & stratification

What impact does this have on invariants & types?
Signs of complexity

These may present themselves in many ways

- Coupling
- Fan in vs fan out
- Layers & stratification
- Cohesion
Signs of complexity

- These may present themselves in many ways
  - Coupling
  - Fan in vs fan out
  - Layers & stratification
  - Cohesion

- These are only some of the signals. In fact you can analyze your workflow to search for other signs!
(Some) ways to seek out complexity [Tornhill 2015]

- Analyzing your version control logs
  - Which files tend to change together?
  - Which files change frequently?
(Some) ways to seek out complexity [Tornhill 2015]

- Analyzing your version control logs
  - Which files tend to change together?
  - Which files change frequently?

- Whitespace analysis & visual complexity
(Some) ways to seek out complexity [Tornhill 2015]

- Analyzing your version control logs
  - Which files tend to change together?
  - Which files change frequently?

- Whitespace analysis & visual complexity

- Visualizing static coupling to assess potential risk
(Some) ways to seek out complexity [Tornhill 2015]

- Analyzing your version control logs
  - Which files tend to change together?
  - Which files change frequently?
- Whitespace analysis & visual complexity
- Visualizing static coupling to assess potential risk
- More guidance can be found in “Your Code as a Crime Scene”
Technical Debt

- *Temporarily* allowing complexity can be useful in order to provide more value along another dimension
  - Perhaps it is to enable progress and exploration before refinement
  - Perhaps efficiency requirements are not well understood yet
  - ...
Technical Debt

- Temporarily allowing complexity can be useful in order to provide more value along another dimension
  - Perhaps it is to enable progress and exploration before refinement
  - Perhaps efficiency requirements are not well understood yet
  - ...

- Making a temporarily bad choice that you know will have to be changed later is known as \textit{technical debt}
Technical Debt

• Temporarily allowing complexity can be useful in order to provide more value along another dimension
  – Perhaps it is to enable progress and exploration before refinement
  – Perhaps efficiency requirements are not well understood yet
  – ...

• Making a temporarily bad choice that you know will have to be changed later is known as technical debt

• Just like financial debt, it can be a useful tool, but the longer it goes unpaid, the greater the damages can be
Technical Debt

- Temporarily allowing complexity can be useful in order to provide more value along another dimension
  - Perhaps it is to enable progress and exploration before refinement
  - Perhaps efficiency requirements are not well understood yet
  - ...
- Making a temporarily bad choice that you know will have to be changed later is known as technical debt
- Just like financial debt, it can be a useful tool, but the longer it goes unpaid, the greater the damages can be
  - And sometimes you may have unintended debts!
Technical Debt

- Temporarily allowing complexity can be useful in order to provide more value along another dimension
  - Perhaps it is to enable progress and exploration before refinement
  - Perhaps efficiency requirements are not well understood yet
  - ...

- Making a temporarily bad choice that you know will have to be changed later is known as technical debt

- Just like financial debt, it can be a useful tool, but the longer it goes unpaid, the greater the damages can be
  - And sometimes you may have unintended debts!
  - Teams that *deliberately* manage it may become 50% faster. [Gartner]
Where we will go

- Much of this semester will involve applying programming skills to explore these issues
Where we will go

- Much of this semester will involve applying programming skills to explore these issues
  - We presented things abstractly here, but we will talk about concrete code.
  - You must be comfortable with concrete code.
Where we will go

- Much of this semester will involve applying programming skills to explore these issues
  - We presented things abstractly here, but we will talk about concrete code.
  - You must be comfortable with concrete code.
- You will end up making trade offs and having regret
Where we will go

- Much of this semester will involve applying programming skills to explore these issues
  - We presented things abstractly here, but we will talk about concrete code.
  - You must be comfortable with concrete code.
- You will end up making trade offs and having regret
- Regret is part of the point. It indicates that you learned something along the way.
Summary

- You should have an intuition about *classic* & *modern* notions of complexity
Summary

• You should have an intuition about classic & modern notions of complexity

• You should understand the high level challenges with complexity that we will be trying to address going forward
Summary

- You should have an intuition about *classic* & *modern* notions of complexity.
- You should understand the high level challenges with complexity that we will be trying to address going forward.
- You should understand that software engineering will involve *judgments* about trade-offs and how to balance such objectives over time.