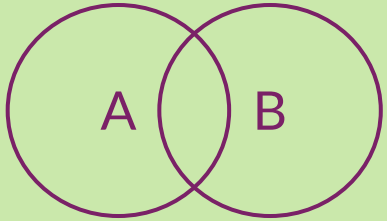
Software as Investment

Reconciling added value and nesting



Dconf 2024 London — Guillaume Piolat

What is Auburn Sounds?

- Was inspired by Walter's posts: from C++ developer to D entrepreneur in 2015
- First product in late 2015 made just 2 sales.

10 years later

- still just me
- 9 launches, 6 products
- OK-ish market-fit
- A normal boring business

resellers
sales
support
marketing
blue button or red button
invoices
competition

Why D for a B2C product business?

Why D for a B2C product business?

1. Iteration is good for product success

Why D for a B2C product business?

1. Iteration is good for product success
2. D nice at iterations

Why D for a B2C product business?

1. Iteration is good for product success
2. D nice at iterations
3. That makes D good for product success

What is different in solodev?

- Programming still the top activity (labor-intensive)

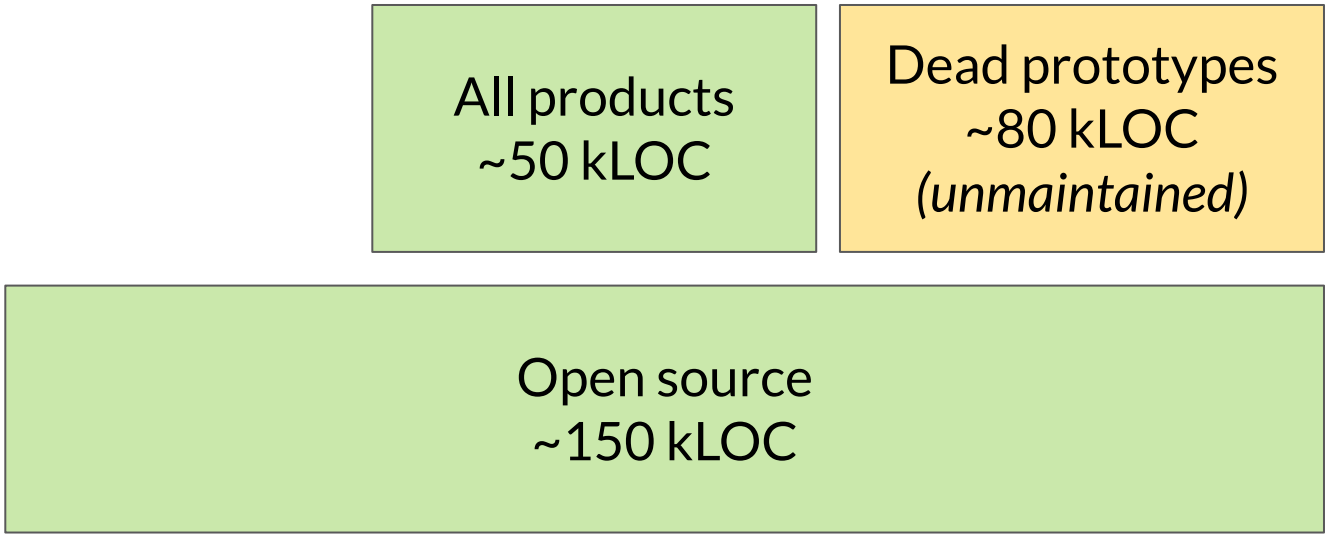
Yes, that's also your job.

What is different in solodev?

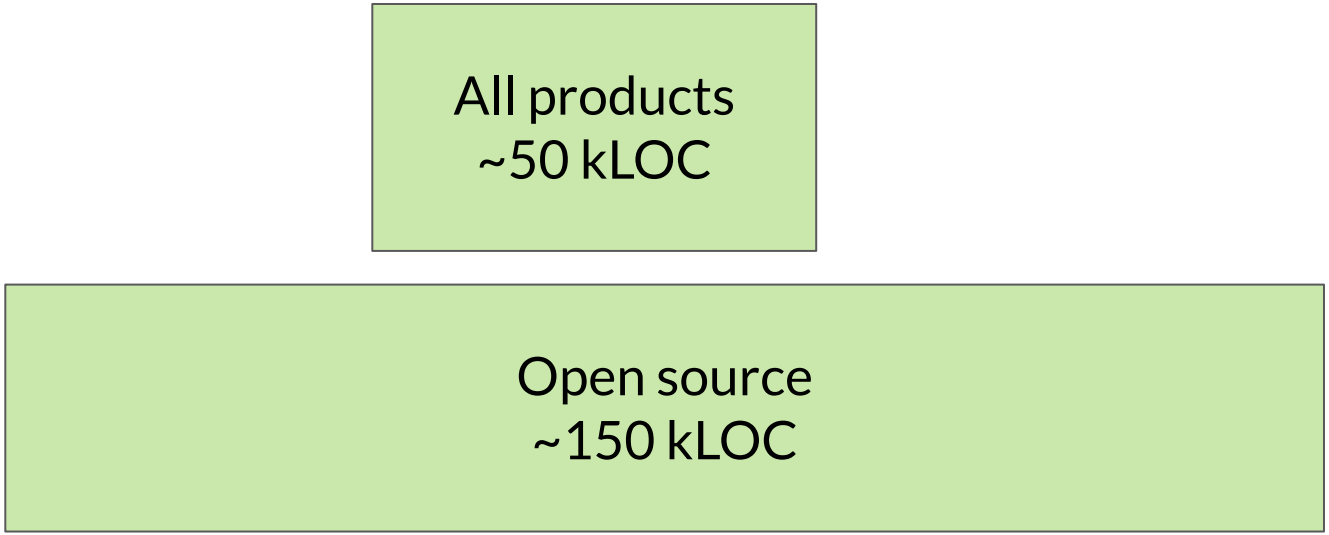
- Programming still the top activity (labor-intensive)
- What is different is seeing its effect on Added Value

Yes, that's also your job.

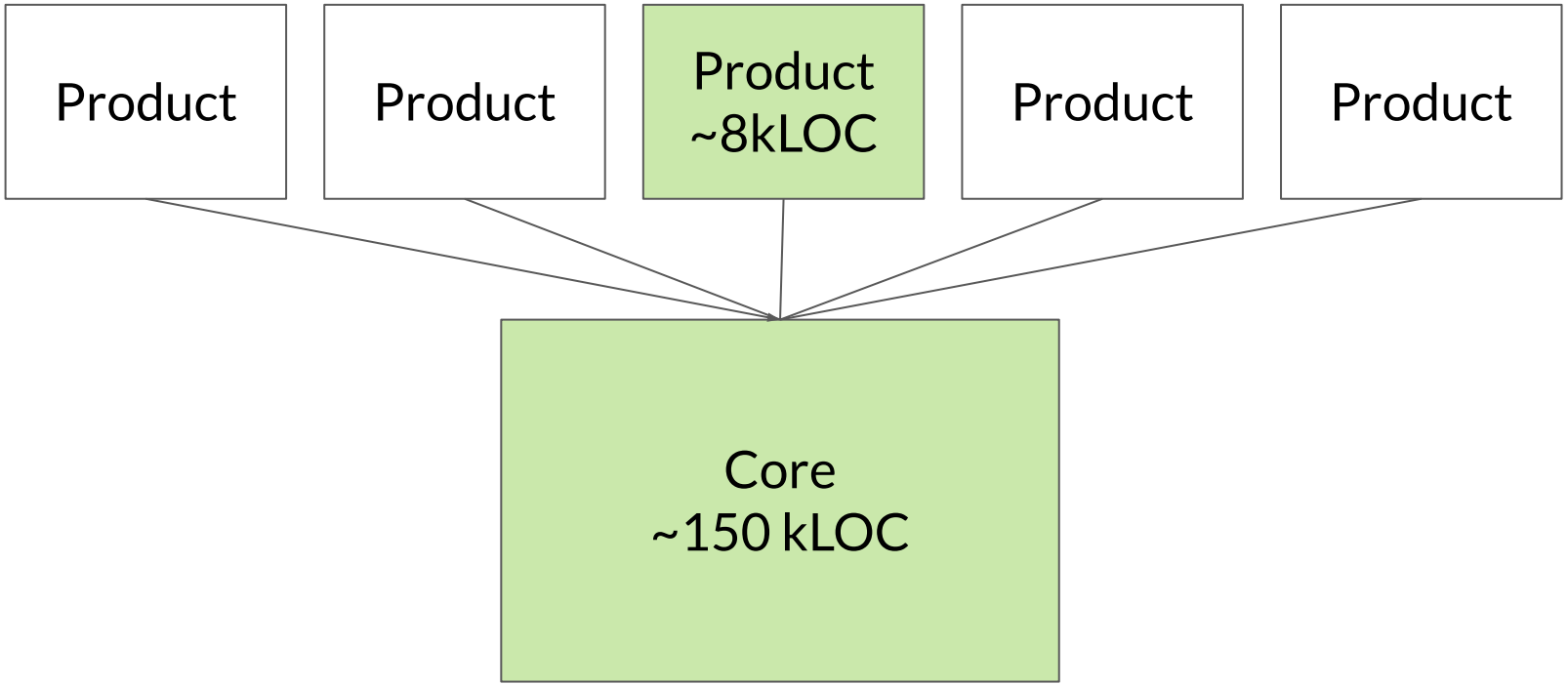
Code quantity at Auburn Sounds



Code quantity at Auburn Sounds

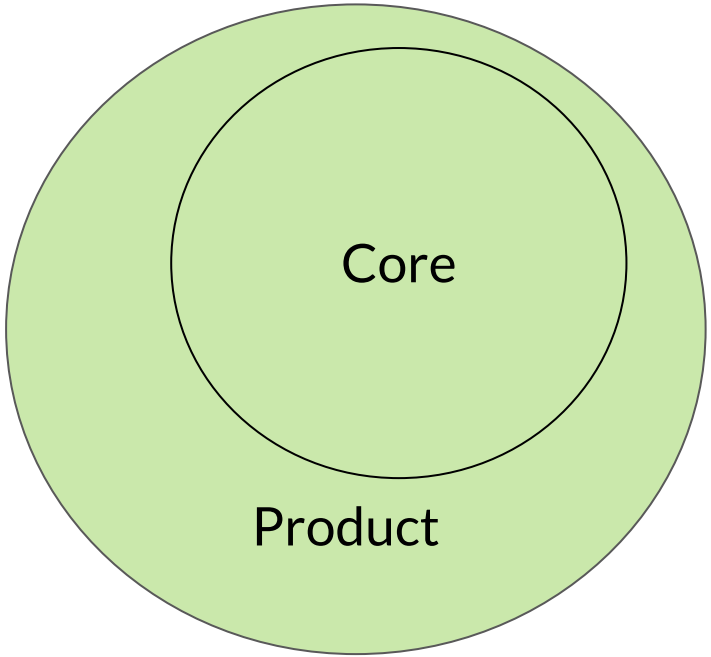


Code quantity at Auburn Sounds



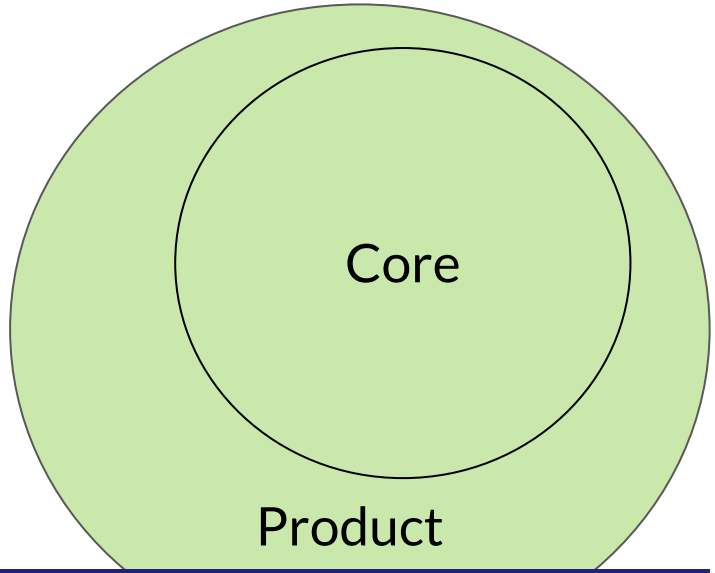
Code quantity at Auburn Sounds

“That one product cost $34k to make and brought $21k over 5 years.”



Code quantity at Auburn Sounds

"That one product cost $34k to make and brought $21k over 5 years."

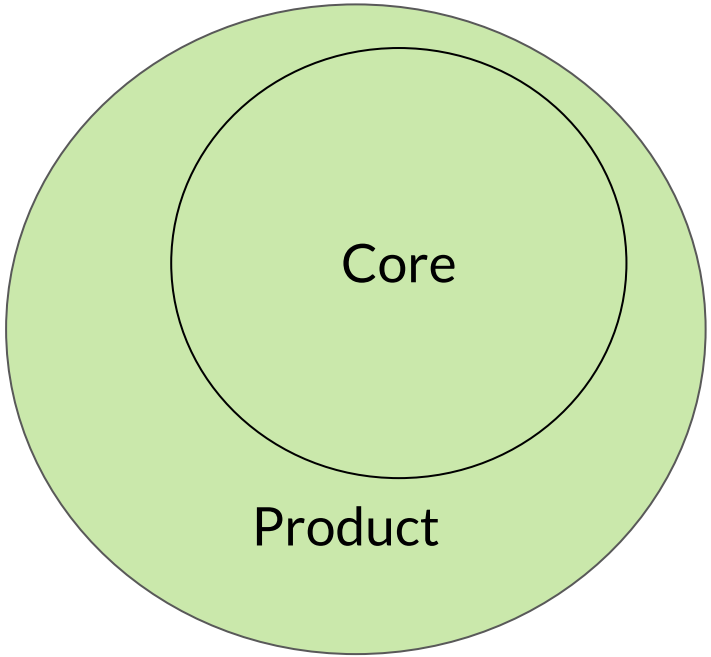


Estimated value is:
ṽ(Product) = $21000 - $34000

That was easy!

Code quantity at Auburn Sounds

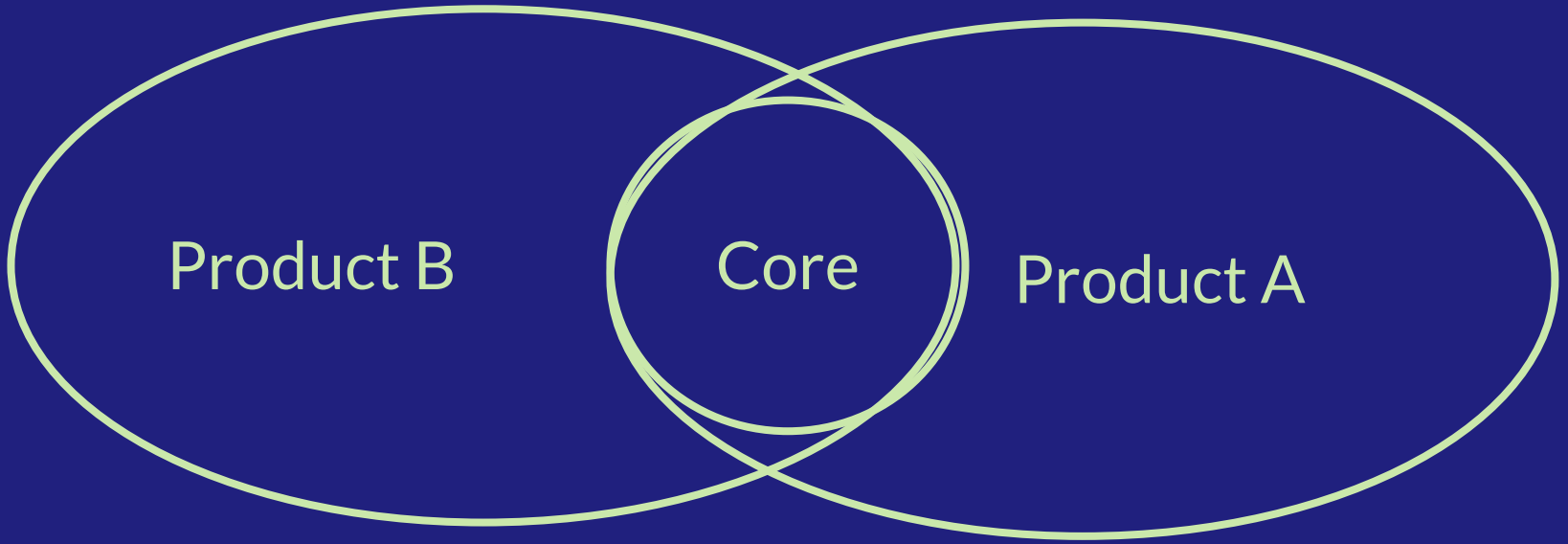
"That one product cost $34k to make and brought $21k over 5 years."

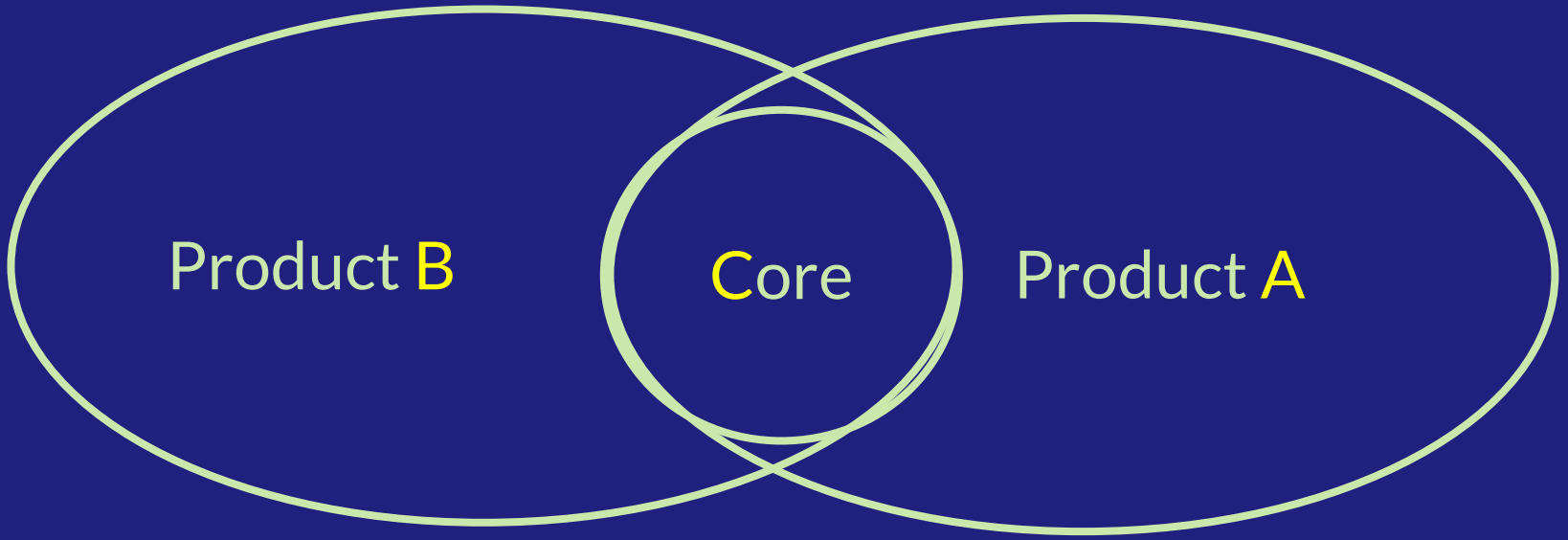


"...and our Core library cost $Z and made $W"

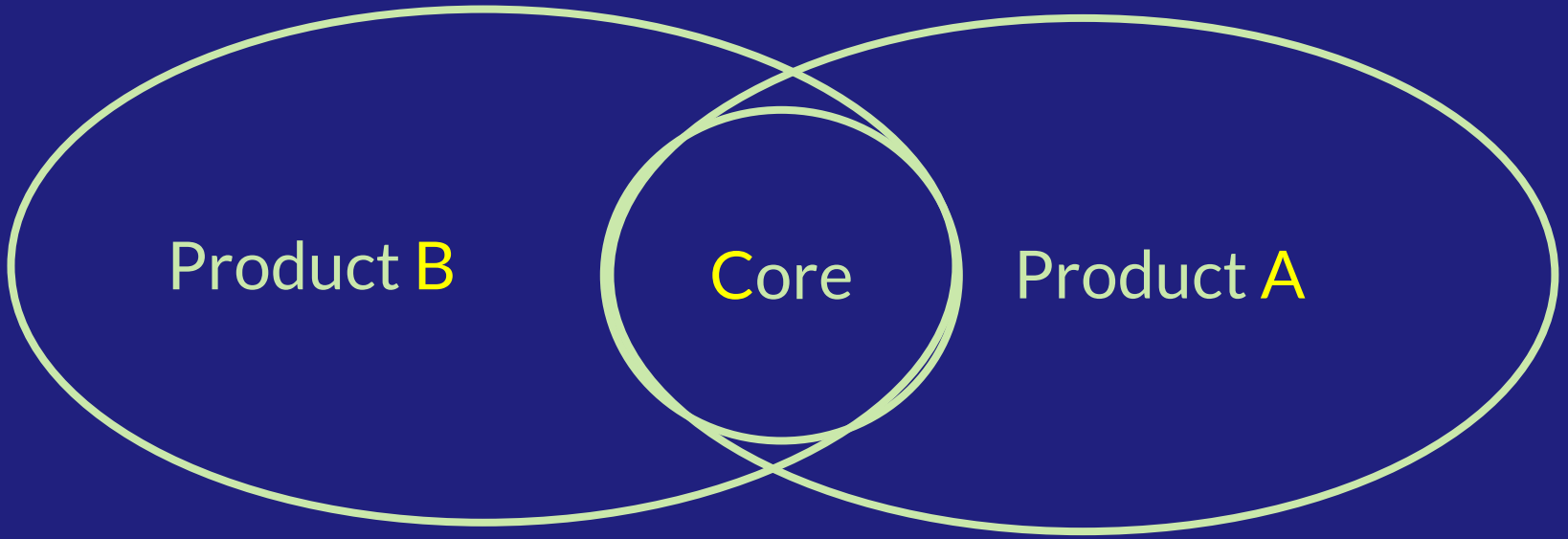
Said no one ever.

What about this case?

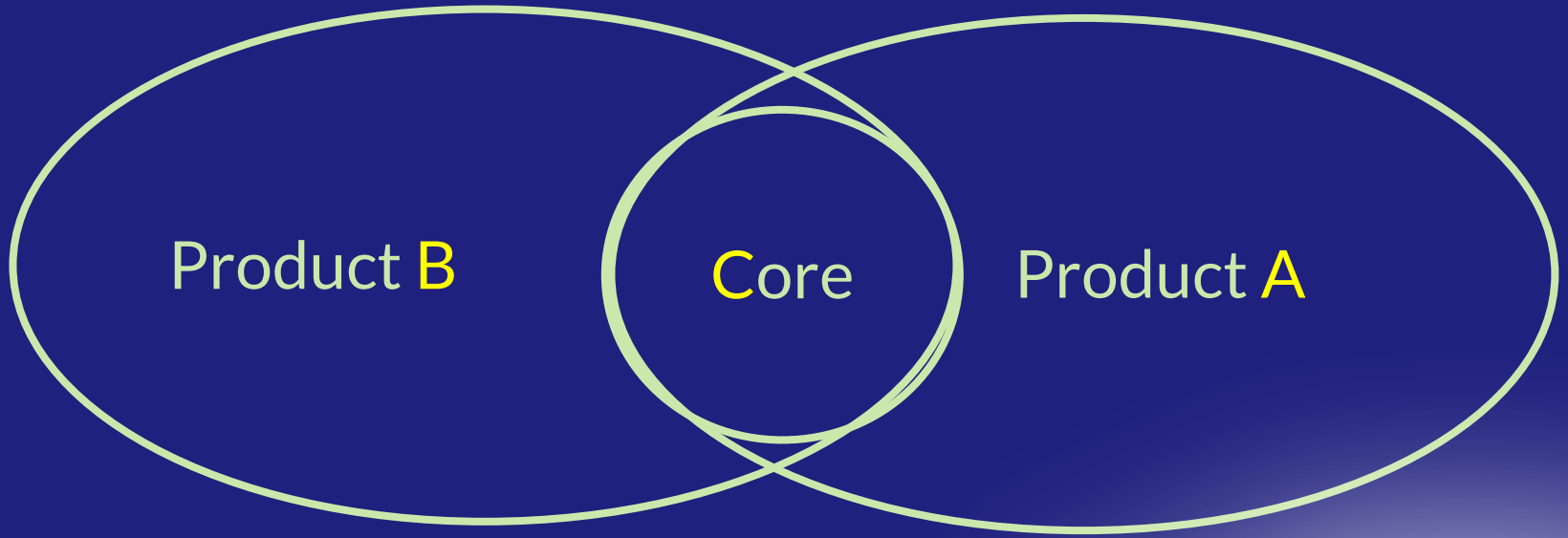




We can estimate the value of A = $\tilde{v}(A)$
We can estimate the value of B = $\tilde{v}(B)$

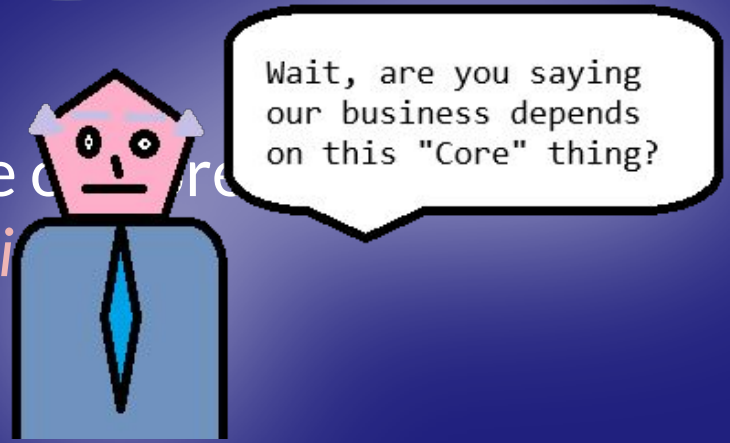


But what is the estimated value of Core $\tilde{v}(C)$
and how long should we work on it?



But what is the estimated value o re

and how long should we work on i



Let’s create a toy theory
that can express nested value.

Software artifacts

Let's reason on any of the following:

- an octet
- a for loop
- a function
- a class object
- a whole product
- a product portfolio

Marked as capital letters in this talk:
A, B, C

```
f
for
class F { }
module xxxx;
etc…
```

wholefile.d

wholeproject/

Software artifacts: Extent

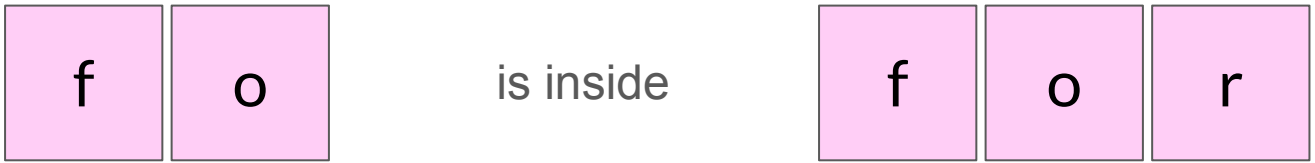
Given an artifact **A :**

- **extent(A)** is its sequence of bytes/unit symbols
  - **A** and **B** are equal if **extent(A) = extent(B)**

If two companies have the same artifact
we'll count its value together

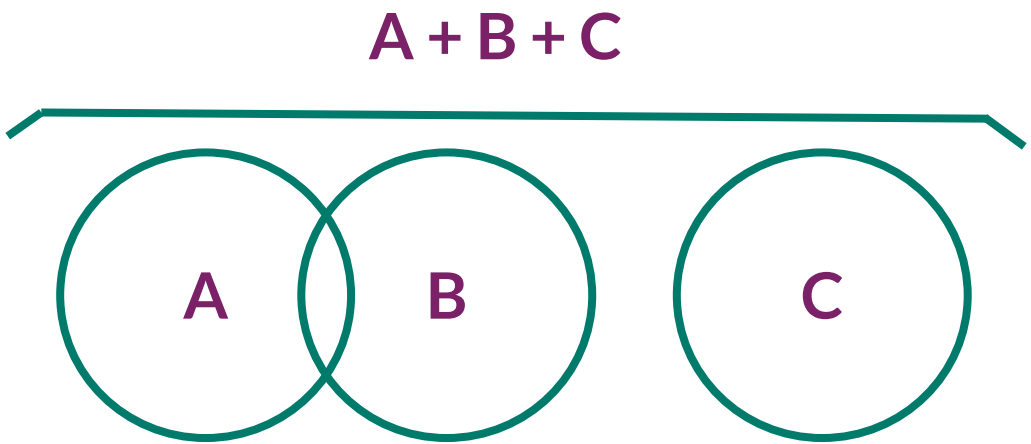
Software artifacts: Subsequence

- **B** is inside **A** if it's a subsequence.



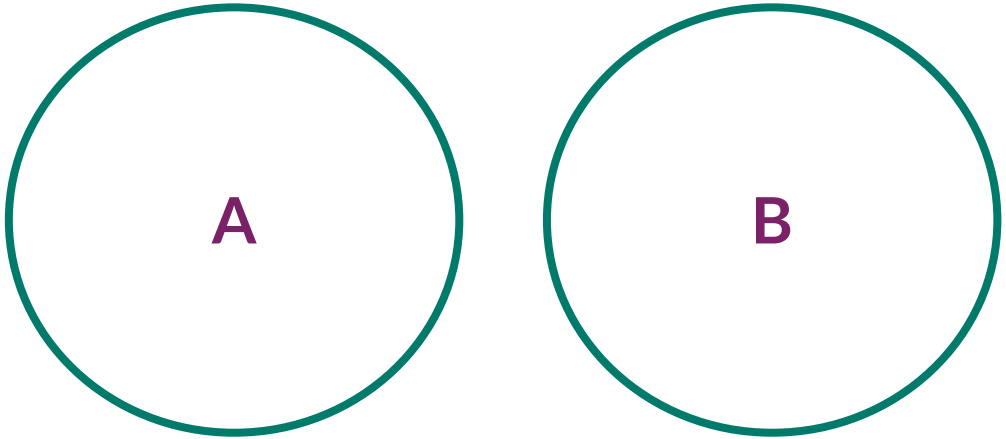
Software artifacts: Sum

- **A + B** is the juxtaposition of artifacts
  - **eg: 2 products A** and **B** = a portfolio **A + B**
  - possibly overlapping



Software artifacts: Postulate of nested value

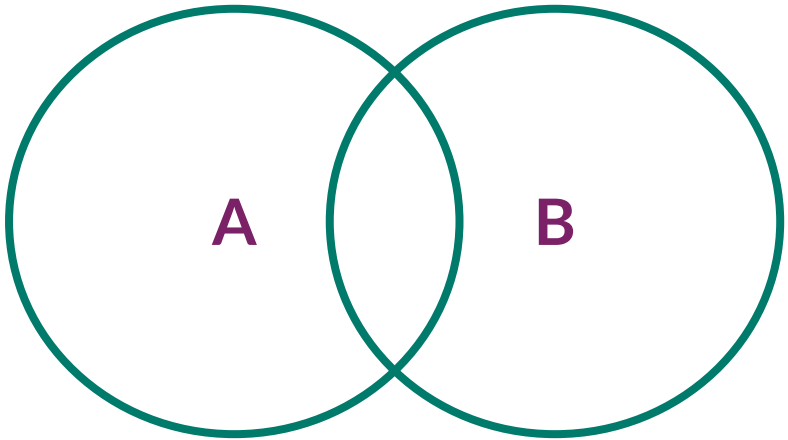
$$value(A + B) = value(A) + value(B)$$

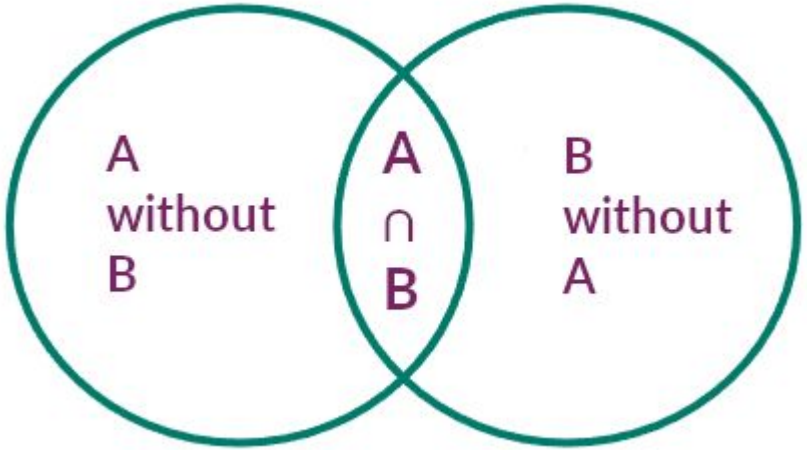


If and only if **A** and **B** are independent.

Software artifacts: Postulate of nested value

$$v(A + B) = 2.v(A \cap B) + v(A \setminus B) + v(B \setminus A)$$





The postulate being that A without B and B without A cleanly exist.

Software value: Definition

Given a software artifact **A :**

- **v(A)** is “all value derived from **A** existing”

Software value: Definition

Given a software artifact **A :**

- **v(A)** is “all value derived from **A** existing”

> “So in fact, the value of the code is the present value of all its future costs and benefits.
>
> John Colvin, Dconf 2022

Software value: Definition

Given a software artifact **A :**

- **v(A)** is “all value derived from **A** existing”
- If **v(A) > 0** then it’s an **asset**
- If **v(A) < 0** then it’s a **liability**

Software value: Definition

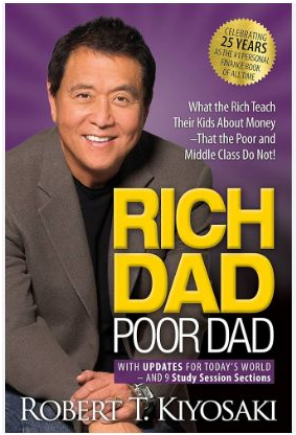
Given a software artifact **A :**

- **v(A)** is “all value derived from **A** existing”
- If **v(A) > 0** then it’s an **asset**
- If **v(A) < 0** then it’s a **liability**
- To have this result, we consider time, money and attention as interchangeable.

Software value: Successful software 📈🚀

If maximizing profit:

- Create artifacts with positive v
- Do NOT create artifacts with negative v



*Much like a stock, or **financial equity**, something below zero isn't good.*

Case study: Accidental Space Age

In 300 years, someone unearth your JSON9 parser DUB package and build a 100T$ venture out of it, propelling humanity into a new space age.

Case study: Accidental Space Age

In 300 years, someone unearth your JSON9 parser DUB package and build a 100T$ venture out of it, propelling humanity into a new space age.

When the DUB registry stops responding, a hero emerges that will change the destiny of civilization…

Case study: Accidental Space Age

In 300 years, someone unearth your JSON9 parser DUB package and build a 100T$ venture out of it, propelling humanity into a new space age.

J = a JSON9 parser

Let **J** be the JSON9 parser.

v(J) = *100T$*

» **v(J)** is unknowable without seeing the deep future.

Case study: Accidental Space Age

In 300 years, someone unearth your JSON9 parser DUB package and build a 100T$ venture out of it, propelling humanity into a new space age.

» **v(J)** is unknowable without seeing the deep future.

» We'll call **ṽ** the *estimate of* **v** and move on

Case study: Accidental Space Age

In 300 years, someone unearth your JSON9 parser DUB package and build a 100T$ venture out of it, propelling humanity into a new space age.

$v(J) = 100T\$ \quad but \quad \tilde{v}(J) = 0$

What goes into v(A)?

Value of artifact A for entity E called

$$v_E(A)$$

What goes into v(A)?

v(A) is all the value of A, for ever, for *everyone*

- Gain and losses from **A** existing for the business
  (eg: sales, velocity, opportunities…)
- Gain and losses from *working* on **A**
  (eg. wages, pleasure and pain, work opportunities, expertise)
- Gain and losses from *operating* the software
  (utility, price paid, brain damage…)
- etc…

What is v(A)? Other definitions.

v(A) is a single monetary number, expressed in $$$

$$v(A) = \sum_{everyone} \int_{t=now}^{\infty} v_{entity}(A, t)\, dt$$

Case study: Totometrics and Ben the Contractor

"Please fix a bug in our JSON parser for $500"

Case study: Totometrics and Ben the Contractor

*"I will do it instantly,
gain $500,
learn nothing from it,
and we'll never meet
again."*

TotoMetrics
the Scrapping Company

Case study: Totometrics and Ben the Contractor

J = a JSON parser

F = the JSON parser fix, independenf of **J**

TotoMetrics
the Scrapping Company

T = TotoMetrics
the Scrapping Company

*"Please change **J** to **J+F** for $500"*

B = Ben the contractor

Case study: Totometrics and Ben the Contractor

J = a JSON parser

F = the JSON parser fix, independenf of **J**

TotoMetrics
the Scrapping Company

T = TotoMetrics
the Scrapping Company

"We thought at the meeting that:

$$\tilde{v}_T(F) > 0$$

because it sure looks like -$500 right now"

Case study: Totometrics and Ben the Contractor

Decision for the business: $\tilde{v}_T(F) > 0$

Considering: $v = \sum_{E=Entities} v_E$ n $\tilde{v} = \tilde{v}_T + \tilde{v}_B$

So:

$$\tilde{v}(F) - \tilde{v}_B(F) > 0$$

Case study: Totometrics and Ben the Contractor

J = a JSON parser

F = the JSON parser fix, independenf of **J**

"In Software as Investment™ terms, I've no real ***exposure*** to **J**, only **F**.

$$\tilde{v}_B(J) = 0$$
$$\tilde{v}_B(F) = \$500$$

Glad I went to that training."

B = Ben the contractor

Case study: Totometrics and Ben the Contractor

$$\tilde{v}_B(F) = \$500$$

$$\tilde{v}(F) - \tilde{v}_B(F) > 0 \longrightarrow \tilde{v}(F) > \$500$$

…which makes sense since only TotoMetrics has exposure to the changed software.

Main Equation of Software Change (ESC)

Most decision problems converge on:

$$\tilde{v}_{whoever\ asks\ the\ question}(Software\ change) > 0$$

Main Equation of Software Change (ESC)

Most decision problems converge on:

$$\tilde{v}_{whoever\ asks\ the\ question}(Software\ change) > 0$$

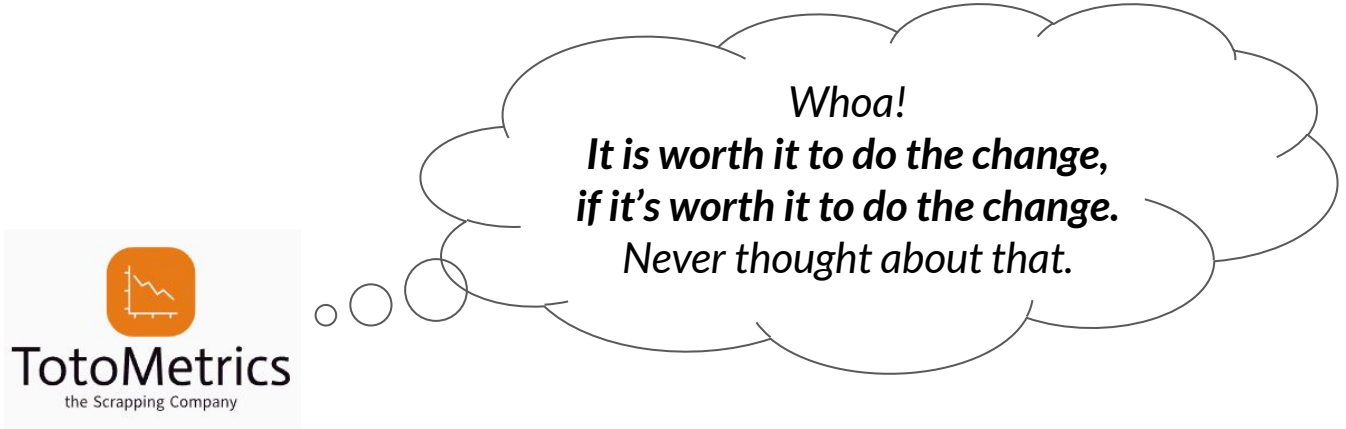
A *"build versus buy"* decision could be:

$$v_T(Dependency\ created\ and\ maintained) > v_T(Dependency\ used)$$

Main Equation of Software Change (ESC)

Most decision problems converge on:

$$\tilde{v}_{whoever\ asks\ the\ question}(Software\ change) > 0$$



Main Equation of Software Change (ESC)

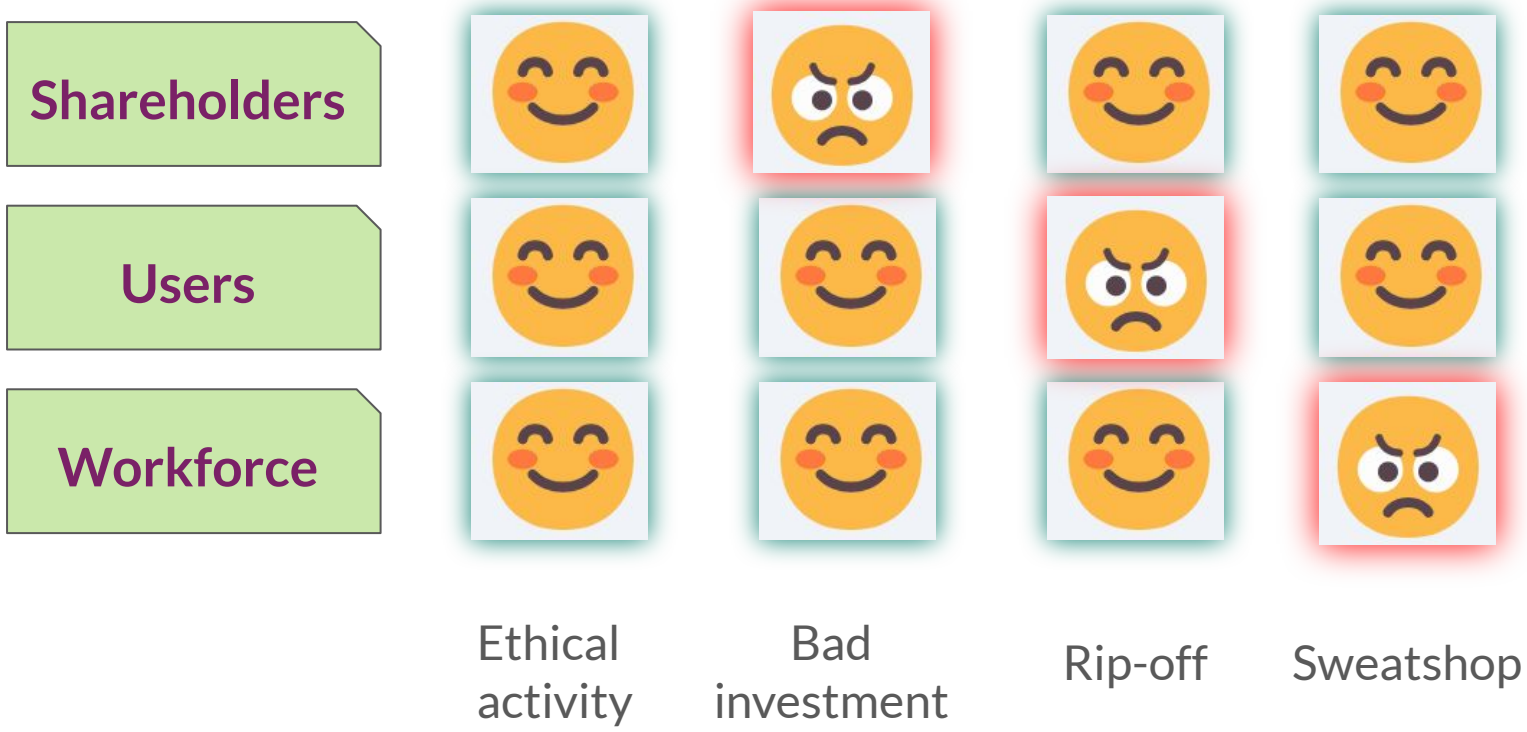
Most decision problems converge on:

$$\tilde{v}_{whoever\ asks\ the\ question}(Software\ change) > 0$$

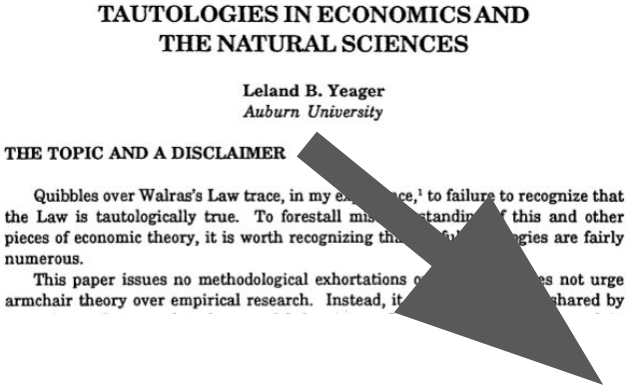
- **tautological** theory that shuffles numbers around *(true by definition)*
- the value of all this (if any) would be… posing the right **terms** and then using a cost model eventually
- Has the non-selfish version uses?
- Does it even map reality well?

$$\tilde{v}(Software\ change) > 0$$

Types of software: Uneven value split



Tautologies are what economists do instead of lunch



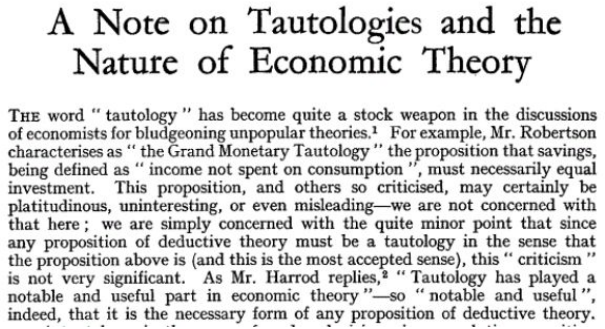
TAUTOLOGIES IN ECONOMICS AND THE NATURAL SCIENCES

Leland B. Yeager
Auburn University

THE TOPIC AND A DISCLAIMER

Quibbles over Walras's Law trace, in my e[illegible]ce,[1] to failure to recognize that the Law is tautologically true. To forestall mis[illegible]standin[illegible] this and other pieces of economic theory, it is worth recognizing th[illegible]gies are fairly numerous.

This paper issues no methodological exhortations [illegible]es not urge armchair theory over empirical research. Instead, it [illegible]hared by



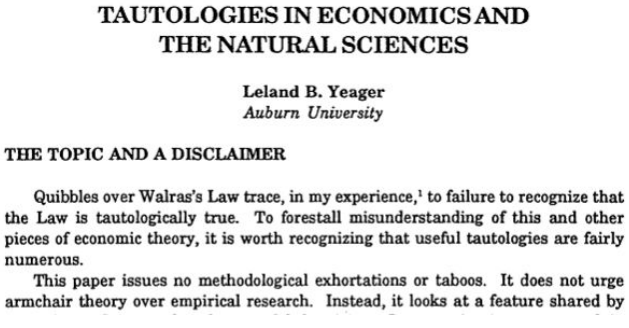
A Note on Tautologies and the Nature of Economic Theory

THE word " tautology " has become quite a stock weapon in the discussions of economists for bludgeoning unpopular theories.[1] For example, Mr. Robertson characterises as " the Grand Monetary Tautology " the proposition that savings, being defined as " income not spent on consumption ", must necessarily equal investment. This proposition, and others so criticised, may certainly be platitudinous, uninteresting, or even misleading—we are not concerned with that here ; we are simply concerned with the quite minor point that since any proposition of deductive theory must be a tautology in the sense that the proposition above is (and this is the most accepted sense), this " criticism " is not very significant. As Mr. Harrod replies,[2] " Tautology has played a notable and useful part in economic theory "—so " notable and useful ", indeed, that it is the necessary form of any proposition of deductive theory.

Abstract

Propositions true by interlocking definitions or by convention occur fairly often. Recognizing their nature and usefulness helps forestall misunderstanding and quibbles. Examples include Walras's Law, the equation of exchange, microeconomic tautologies, the money-multiplier formula, the government budget constraint, the principle of comparative advantage, and certain strands of balance-of -payments analysis. Comparisons are drawn with examples from mathematics, classical mechanics, electricity, and other fields.

Tautologies are what economists do instead of lunch



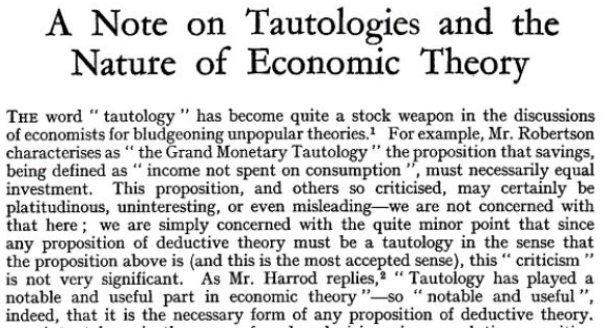
TAUTOLOGIES IN ECONOMICS AND THE NATURAL SCIENCES

Leland B. Yeager
Auburn University

THE TOPIC AND A DISCLAIMER

Quibbles over Walras's Law trace, in my experience,[1] to failure to recognize that the Law is tautologically true. To forestall misunderstanding of this and other pieces of economic theory, it is worth recognizing that useful tautologies are fairly numerous.

This paper issues no methodological exhortations or taboos. It does not urge armchair theory over empirical research. Instead, it looks at a feature shared by



A Note on Tautologies and the Nature of Economic Theory

THE word " tautology " has become quite a stock weapon in the discussions of economists for bludgeoning unpopular theories.[1] For example, Mr. Robertson characterises as " the Grand Monetary Tautology " the proposition that savings, being defined as " income not spent on consumption ", must necessarily equal investment. This proposition, and others so criticised, may certainly be platitudinous, uninteresting, or even misleading—we are not concerned with that here ; we are simply concerned with the quite minor point that since any proposition of deductive theory must be a tautology in the sense that the proposition above is (and this is the most accepted sense), this " criticism " is not very significant. As Mr. Harrod replies,[2] " Tautology has played a notable and useful part in economic theory "—so " notable and useful ", indeed, that it is the necessary form of any proposition of deductive theory.

Viewed as a language, theory has no substantive content; it is a set of tautologies. Its function is to serve as a filing system for organizing empirical material and facilitating our understanding of it;

Friedman - 1966

Case study: Bayesian extension

O’Kernel, the signal processing researcher

Case study: Bayesian extension

O’Kernel, the signal processing researcher

A promising image resizing method.

Case study: Bayesian extension

R = the **R**esizing method

K = O'Kernel

the Signal Processing Researcher

"I wonder if $\tilde{v}_K(R) > 0$

Case study: Bayesian extension

R = the **R**esizing method

K = O'Kernel
the Signal Processing Researcher

"I wonder if $\tilde{v}_K(R) > 0$

The new thing might be a right bungle of a disaster."

Case study: Bayesian extension

$v(R \mid e)$ Value of **R** if event **e** happens

$v(R \mid \neg e)$ Value of **R** if event **e** does **not** happen

Case study: Bayesian extension

R = the Resizing method

Event b = the new method is better

K = O'Kernel
the Signal Processing Researcher

$\tilde{v}_K(R) > 0$ and by application of the Bayesian rule:

$$\tilde{v}_K(R) = \tilde{v}_K(R \mid b) * prob(b) + \tilde{v}_K(R \mid \neg b) * (1 - prob(b))$$

Case study: Bayesian extension

Introducing an example **cost model** for real world use.

R = the Resizing method

Event b = the new method is better

"My cost model be:
software cost = writin' it + maintainin' it
There be treasure to be found, if the research turns out bountiful!"

K = O'Kernel
the Signal Processing Researcher

Case study: Bayesian extension

Introducing an example **cost model** for real world use.

R = the Resizing method

Event b = the new method is better

"My cost model be:
software cost = writin' it + maintainin' it
There be treasure to be found, if the research turns out bountiful!"

K = O'Kernel
the Signal Processing Researcher

$$\tilde{v}_K(R \mid b) = writing(R) + maintaining(R) + algorithmicgain(R)$$

$$\tilde{v}_K(R \mid \neg b) = writing(R)$$ ← no need to keep R if failure

Case study: Bayesian extension

R = the Resizing method

Event b = the new method is better

$$\tilde{v}_K(R \mid b) = writing(R) + maintaining(R) + algorithmicgain(R)$$

$$\tilde{v}_K(R \mid \neg b) = writing(R)$$

$$\tilde{v}_K(R) = \tilde{v}_K(R \mid b) * prob(b) + \tilde{v}_K(R \mid \neg b) * (1 - prob(b))$$

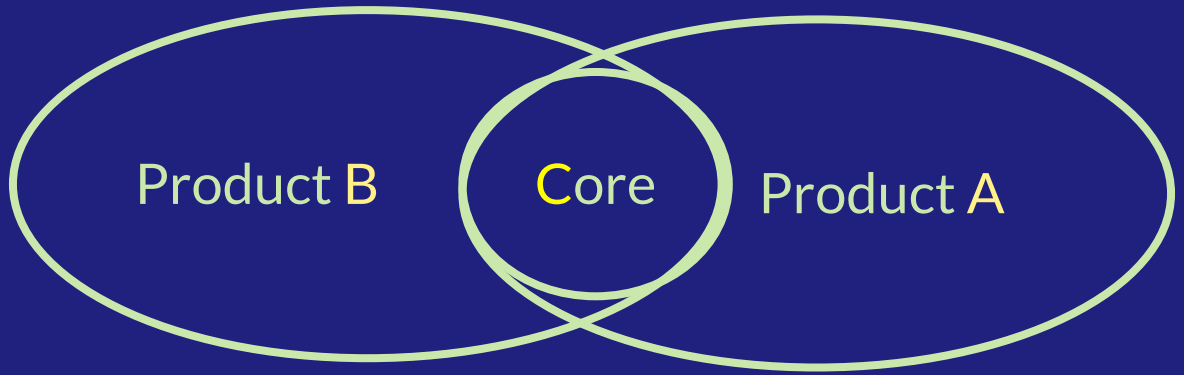
Within this cost model, the formula for **unsure reward** is:

K = O'Kernel

the Signal Processing Researcher

$$\tilde{v}_K(R) = writing(R) + (maintaining(R) + algorithmicgain(R)) * prob(b)$$

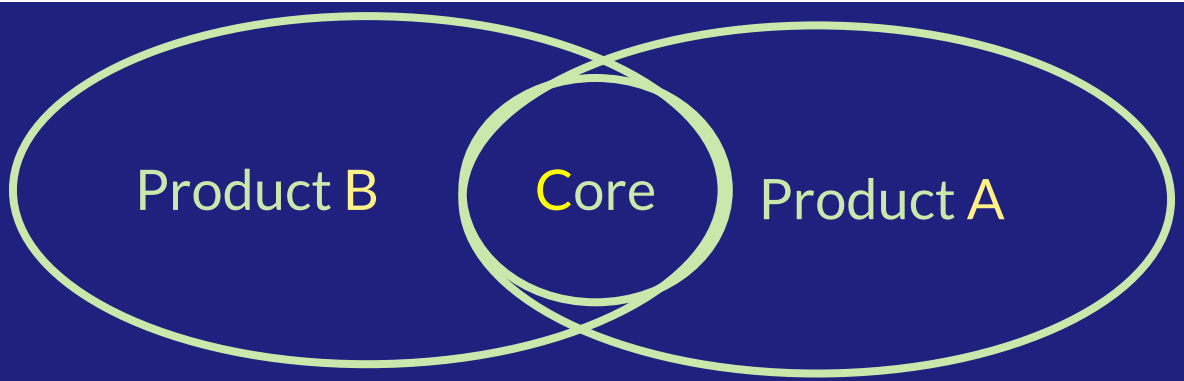
Pricing dependencies



Pricing dependencies

To compute the value of C an existing artifact…
estimate the -value of C being deleted.

$$\tilde{v}(C \textit{ will be created}) = -\tilde{v}(C \textit{ exists and will be deleted})$$



That Take-away slide

- COCOMO method considers the **cost** of software.

The COCOMO (Constructive Cost Model) is a widely used model that estimates the effort, time, and cost associated with software development projects.

That Take-away slide

- COCOMO method considers the **cost** of software.

  The COCOMO (Constructive Cost Model) is a widely used model that estimates the effort, time, and cost associated with software development projects.

- You can instead consider its **value** on a 1D axis, also nestable, using such a language *(conflating time, effort, and cost)*

That Take-away slide

- COCOMO method considers the **cost** of software.

  The COCOMO (Constructive Cost Model) is a widely used model that estimates the effort, time, and cost associated with software development projects.

- You can instead consider its **value** on a 1D axis, also nestable, using such a language *(conflating time, effort, and cost)*

- Accountants consider IP as **equity** *(assets minus liabilities)*, **owned**, but doesn't nest its value nor consider it as commons.

That Take-away slide

- COCOMO method considers the **cost** of software.

  The COCOMO (Constructive Cost Model) is a widely used model that estimates the effort, time, and cost associated with software development projects.

- You can instead consider its **value** on a 1D axis, also nestable, using such a language *(conflating time, effort, and cost)*

- Accountants consider IP as **equity** *(assets minus liabilities)*, **owned**, but doesn't nest its value nor consider it as commons.

- But "exposure" **v(A)** might well be a more useful concept than "ownership"

That Take-away slide

- COCOMO method considers the **cost** of software.

  The COCOMO (Constructive Cost Model) is a widely used model that estimates the effort, time, and cost associated with software development projects.

- You can instead consider its **value** on a 1D axis, also nestable, using such a language *(conflating time, effort, and cost)*

- Accountants consider IP as **equity** *(assets minus liabilities)*, **owned**, but doesn't nest its value nor consider it as commons.

- But "exposure" **v(A)** might well be a more useful concept than "ownership"

- You'll certainly want a cost model to make that useful

Questions?

That Bonus slide

- We do have language support for value annotations!

```
// TODO
```

means: *“this probably has > 0 value, since it’s worth it to work on it further”*

That Bonus slide

- We do have language support for value annotations!

```
// TODO
```

means: *“this probably has > 0 value, since it’s worth it to work on it further”*

```
deprecated
```

means: *“this probably has < 0 value, since it’s worth it to delete it”*