Approaches that are Problematic

- Natural Language
- Disciplined Natural Language
- Code

They have their advantages, but serious disadvantages too

Using Different Models to Represent Different Aspects of Software

- Model represents certain specific features
- Others are suppressed, not represented
- Aim is to clarify some issues by not having to worry about the others
- But this leads to different representations of reality
- How to see/infer the whole picture?

Plato’s Allegory of the Cave

The question of how to synthesize multiple views of “reality” is at least 2000 years old

Plato’s Allegory of the Cave

What is most tangible is least “real”
What is least tangible is most “real”
Plato could have been (was?) a great software engineer
Graphs as Visualization Aids

- A graph is a picture of a relation
- Graphs are mathematical structures with obvious visualizations that seem often to help many stakeholders visualize these relations.
- A graph's edges visually represent the ordered pairs that compose the relation.
- If the pairs in $E$ are ordered, then $G$ is a directed graph, and its edges are depicted with arrowheads.
- If not, the graph is called an undirected graph.

Example: Elevator Controller

- What is it supposed to do?
  - Stop on every floor it is called to
  - Maybe not so easy: multiple elevators
  - Service users "first come, first served"
  - May conflict with optimal strategies
- These are hard to be precise about, reason about in "plain English"
- Code helps with some of these, but not all
  - What should it never do?
    - Allow elevator to move with doors open
  - Graphs are frequently used to specify this kind of thing
    - Many kinds of graphs for many purposes
    - A graph is a picture of a relation

Data Flow Graph

Control Flow Graph

Finite State Machine

Petri Net
Pictures are not enough

- Want to be able to reason about them
- Answer (stakeholder) questions
  - Can this data item ever reach that statement?
  - Can these two events ever happen in sequence?
  - What is the maximum time to execute that sequence?
- Pictures can leave ambiguous impressions
  - How to be sure what they say?
- Graphs, representing mathematical relations, can support deriving definitive answers
  - About the relation

What is This Graph Specifying?

What’s wrong with this diagram

Annotations Provide Intuitions

What’s wrong with this diagram

Back to Basics

- Review fundamental Finite Mathematics
  - Set theory
  - Graph theory
  - Predicate Calculus
  - Etc.
- But with an emphasis on why this is of interest in software engineering
Relations:
A RELATION, R, over a set, S = \{s\} is a set of tuples
\[ R = (r), \text{ where } r = (s_1, s_2, \ldots, s_n) \]
An n-ary relation is a relation where all the tuples are n-tuples
A Binary relation is a relation where all the tuples are 2-tuples
If \((s, s)\) is an element of R, then we often write \(s R s\)

Another view of relations:
The relation, R, over the set S can be defined as:
\[ R = \{ (s, s) \mid \text{PRED}(s, s) = \text{True, for some predicate, PRED} \} \]

If the tuples are ordered, the relation is called an ordered relation
If the tuples, \(s_1, s_2, \ldots, s_n\), are unordered, the relation is an unordered relation

Some Properties of Relations
Some familiar properties of ordered binary relations, R, over the set \(S = \{s\}\):
Symmetry: \(s R s \iff s R s\) for all pairs, \(s\) and \(s\) in \(S\)
Reflexivity: \(s R s\) for all \(s\) in \(S\)
Transitivity: \(s R s\) and \(s R s\) \(\implies\) \(s R s\) for all \(s, s, s\) in \(S\)

A relation that is symmetric, reflexive and transitive is called an equivalence relation

If \(R = \{(s, s)\}\) is transitive, then \(C = \{(s, s)\}\) there exists a sequence, \(i_1, i_2, \ldots\), in \(S\), such that \(s_{i_1} \in R \land s_{i_2} \in R \land \ldots \land s_{i_n} \in R\)

Antisymmetry: \(s R s \iff \neg(s R s)\) for all pairs, \(s\) and \(s\) in \(S\)
Irreflexivity: \(\neg R s\) for all \(s\) in \(S\)

Examples
If \(S = \{\text{all subroutines written in Fortran}\}\) \(s R s\) if and only if \(s\) calls \(s\), then \(R\) is an irreflexive relation

Let \(PS = \{c, \text{ all the statements in a program that consists of a set of modules, } M = \{m_1\}\}\),
\[ \text{INMOD} = \{(c, c) \mid c \text{ and } c \text{ appear in the same module } m_1\} \]
\(\text{INMOD}\) is an equivalence relation

The relation \(\text{ImmFol}\) (earlier slide) is not transitive
Change \(\text{ImmFol}\) to \(\text{Fol}\), by defining \(\text{Fol} = \{(L_1, L_2) \mid \text{the execution of } L_2 \text{ may follow the execution of } L_1 \text{ for some execution of } P\}\)

Graphs
A Graph, \(G = (N, E)\) is an ordered pair, consisting of a node set, \(N\), and an edge set, \(E = \{(n, n)\}\)

If \(OG = (N, E)\) is an ordered graph with \(E = \{(n, n)\}\) then its unordered version, \(UG = (N, U)\), where \(U = \{<n, n>\}\)
Graphs as Visualization Aids

- Graphs are mathematical structures with obvious visualizations that seem often to help many stakeholder communities to visualize key relations.
- A graph’s edges visually represent the ordered pairs that compose the relation
- If the pairs in E are ordered, then G is a directed graph, and its edges are depicted with arrowheads
- If not, the graph is called an undirected graph

Paths

A path, P through an ordered graph G = (N, E) is a sequence of edges, (n_1, n_2), (n_2, n_3), ..., (n_k, n_1)

A path, UP, thru an unordered graph UG = (N, U) is a sequence of edges, (n_1, n_2), (n_2, n_3), ..., (n_k, n_1)

If the pairs in E are ordered, then G is a directed graph
If not, the graph is called an undirected graph

Trees

A cycle in a graph G is a path whose start node and end node are the same.

A simple cycle in a graph G is a cycle such that all of its nodes are different (except for the start and end nodes).

If a graph G is connected and has no path through it that is a cycle, then the graph is called acyclic.

An acyclic, connected, unordered graph is called a tree.

A collection of trees is called a forest.

If the unordered version of an ordered connected graph is acyclic, the graph is called a directed tree.

If the unordered version of an ordered graph has cycles, but the ordered graph itself has no cycles, then the graph is called a Directed Acyclic Graph (DAG).

DATA FLOW Graph (Again)

- Capture system functionality: What does system do? How?
- Basic components of a data flow diagram:
  - Nodes, represented by circles (boxes), are functional units
  - Edges, represented by arrows, are data flows between units
  - Both augmented by separate annotation relations
  - Boxes (sometimes circles), represent I/O data

Other Types of Graphs

A Multigraph MG is an ordered pair MG = (N, C) where N is a set of nodes (n) and C is a collection of pairs of nodes (edges) with repetitions allowed (ie. C can be a multiset).

A Hypergraph HG is an ordered pair HG = (N, T) where N is a set of nodes (n) and T is a set of t-tuples of nodes, where t > 2.

A Hypermultigraph is a hypergraph where the set of t-tuples can be a multiset.

A bipartite graph BG is an ordered pair, BG = (BN, E) where BN is a node set that is the union of two disjoint subsets, N: U N_b, and no edge in E has both nodes in either N or N_b.

A bipartite graph is often called a 2-colorable graph.

A k-colorable graph is defined analogously, with BN being the disjoint union of k subsets.

Differences in Graphs Result from Different Relations

- Data Flow:
  - Nodes represent sets of sites where data is generated/used
  - Each edge is a (data generated, data used) node pair
- Control Flow:
  - Nodes represent units of functionality
  - (n, n) is an edge in this graph if and only if unit n executes (ImmFol relation)
- Hierarchy:
  - Models “consists of” or “is a part of”
- Key to divide-and-conquer approaches to understanding
- Finite State Machines
  - Nodes represent all possible different “execution states”
  - (s_i, s_j) is an edge if and only if it is possible for state s_i to immediately succeed s_j.
  - Called a transition from s_i to s_j.
- Edges annotated with transition condition
- Petri Nets
  - Multiple node and edge types in the same diagram
Formalizing DFGs as Relations

- \((i, j) \in \text{DataFlow}_G\) if node \(i\) creates data that node \(j\) uses in \(G\)
- \(\text{INPUT}_G = \{\text{set of nodes (input)}\) such that input is a provider of input to \(G\) from an external source
- \(\text{OUTPUT}_G = \{\text{set of nodes (output)}\) such that output is a conveyor of artifacts computed by \(G\) to an external source
- \((e, \text{operand}) \in \text{EdgeAnnotation}_G\) if \(\text{operand}\) is the name of the artifact that flows along an edge \(e \in E\), where \(G = (N, E)\)
- Preferably the data artifact is defined rigorously
- \((n, \text{text}) \in \text{NodeAnnotation}_G\) if the string \(\text{text}\) describes the functioning of node \(n \in N\), where \(G = (N, E)\)
- This may imply different semantics (e.g. abstraction)
- InputAnnotation, OutputAnnotation are similar

Questions this helps answer:

- Why create this data? Who uses this data? What results does the end user see? What does the end user have to input?
- How does a node do its job?

Flowgraphs

Let \(S = \{\text{all statements } s\) in a program, \(P\)

Let \(\text{ImmFol} = \{(s_i, s_j) \mid \text{The execution of } s_j \text{ immediately follows the execution of } s_i \text{ for some execution of } P\}\)

Then: If \(FG = (S, \text{ImmFol})\), \(FG\) is called the flowgraph of \(P\)

\(FG\) is an ordered graph

Every execution sequence (i.e., the sequence in which the statements of \(P\) are executed for a given execution of \(P\)) corresponds to a path in \(FG\).

However—the converse is not true. A path through \(FG\) may not correspond to an execution sequence for \(P\)

A loop in \(P\) appears as a cycle in \(FG\)

Callgraphs

Let \(PROC = \{\text{procedures } S\) that the program \(P\) comprises\)

Let \(\text{CALLS} = \{(S_i, S_j) \mid S_j \text{ is directly invoked from } S_i \text{ during some execution of } P\}\)

Then \(CG = (PROC, CALLS)\) is called the Call Graph of \(P\)

\(CG\) is a directed graph

If \(P\) is written in a language that does not allow recursion, then \(CG\) will be acyclic

A cycle in \(CG\) indicates that the nodes along the cycle participate in a recursive calling chain

NOTE: DEPICTIONS OF THESE GRAPHS MAY BE SUPERIMPOSED OVER EACH OTHER TO CLARIFY (?!)

Consistency is a principal concern

- Are the diagrams consistent with each other?
- Top view consistent with elaborations?
  - Arrows consistent
  - Data flows consistent
  - Other semantics?
- Are we seeing different shadows of the same object?
- Invitation to subtle errors

Hierarchy

- Enables incrementally adding detail
- Increased precision too
- Draws upon innate human mental capability
  - Abstraction
  - Encapsulation
- A typical solution to the problem of needing detail, but needing to avoid overload
- But creates potential problems

Plato’s Allegory of the Cave

The question of how to synthesize multiple views of “reality” is at least 2000 years old
Plato’s Allegory of the Cave

What is most tangible is least “real”
What is least tangible is most “real”
Plato could have been (was?) a great software engineer

Simple Hierarchical Elaboration

Something a Little More Complex

Hierarchical Elaboration (of “Select New Floor”)

Hierarchical Elaboration (of “Select New Floor”)

Is this consistent with its parent?

Inconsistencies with parent
Hierarchical Elaboration (of “Select New Floor”)

(Some) Consistency definitions

- Let \((n, \text{somenode}) \in \text{NodeAnnotation}_{\text{ParentGraph}}\) for some \(n, \text{somenode} \in N\), where \(\text{ParentGraph} = (N, E)\)
- Let \(\text{somenode} = (N', E')\)
  - This is the DFG elaborating on “somenode”
- Some consistency properties
  - \(\text{If } ((m, \text{somenode})) \neq \emptyset, m, \text{somenode} \in N, \text{then } \text{Cardinality}(\text{INPUT(somenode)}) \neq 0\)
  - \(\text{If } ((\text{somenode}, m)) \neq \emptyset, m, \text{somenode} \in N, \text{then } \text{Cardinality}(\text{OUTPUT(somenode)}) \neq 0\)
- Maybe some others(?)
  - \(\text{If } \text{Cardinality}((\text{m, somenode})) = k, m, \text{somenode} \in N, \text{then } \text{Cardinality}((\text{INPUT(somenode)})) = k\)
  - \(\text{If } \text{Cardinality}((\text{(somenode, m)})) = k, m, \text{somenode} \in N, \text{then } \text{Cardinality}((\text{OUTPUT(somenode)})) = k\)

Data needs precise specification too

- DFG’s focus on functionality, using data as a vehicle
- Data shown as unstructured atomic units--usually unrealistic
- Complex functions cannot be adequately defined without delving into the details of how they handle structured data
- Sub-DFG’s can show how the high level data that high level DFG’s deal with is decomposed
  - But this is implicit data definition
  - Can be hard to read/inconsistent
- Data specification is worth doing explicitly, carefully
- Usually using Disciplined Natural Language--eg. Templates
- Formal specification of data is an important future topic

Elaboration of “Select New Floor” Suggests Artifact Issues

IDEF0

- Commercial DFG formalism
- Some formality and rigor behind it
- Primarily pictorial
- In wide use
- Additional semantics in IDEF1, IDEF2, etc.
Note: The edges here do not comprise a set; they comprise a collection.

Broadening DFG Semantics

- Node cannot begin until data arrives along all in-edges

Let \( \text{DFG} = (N, E) \), where \( N \) is set of nodes, \((n)\). If \( n \) is a function, then it must be defined on the set of all artifacts \( A = \{a_i\} \), such that \( (n, n_i, i, a_i) \in \text{EdgeAnnotation}_{\text{DFG}} \).

- How to adapt this for
  - any semantics?
  - for output semantics?
  - exactly one Output node
  - Etc.

These sorts of constraints can support additional types of reasoning:

Eg. about parallelism
More Broadening
- Use of "open boxes" to indicate data store
  - A different set, with different semantics
  - Not a computation function
  - Methods are: put, get, search(?)

Still More Broadening
- Different shapes of boxes
- Different pictures instead of boxes
- Different colors
- Different lines
  - .......

Central questions: What are the semantics?
Does this really help? Or confuse?

Scientific Workflow Graph

Attempt to make this more precise

But iteration complicates matters

Kepler--Another DFG Technology
- Data Flow Graph notation
- Has hierarchical decomposition
- Capability for specifying DFG semantics
  - For each diagram
  - Can be different at each level of hierarchy (!)
- Based on Ptolemy II system
A Kepler Example

Kepler Elaboration

What kinds of questions are well addressed by DFGs?

• Overall structure of functional capabilities
  – What does this piece do?
• System outputs and inputs
• How might changes be made?
• What functions create what data entities

Given that Precision is essential

• What about the other three dimensions?
• Detail
  – Gained from hierarchical elaboration
• Breadth
  – Comes from different (sub)types of DFG
• Clarity
  – Seems to be reduced by increased Detail and Breadth
  – With the need for Precision

Considerable Appeal, but Limited Value, to most stakeholders

• Users think they have sufficient understanding
  – But have trouble being able to see easy things (iteration)
• Developers have same problem
• Managers may only care to see easy things (!)
  – Although they should be interested in more
• Bystanders may be shown only easy things
  – Which could be a real problem

Final observations

• Very primitive representation
  – Very limited semantics
• But actually more a family of model types
  – Different sets of semantics
• The actual relation(s) are rarely made clear and precise
• Powerful aid to intuition and efficiency of communication
  – Clear advantages over natural language
• But is intuition misled by ambiguity, misinterpretation?
• Does not help explain HOW things get done