Engineering Large-Scale Software-Intensive Systems

Geoff Dromey
Software Quality Institute
ARC Centre for Complex Systems

© R.G.Dromey, Griffith University, 2007
Plato’s Advice

“The beginning is the most important part of the work”

Applies very much to Systems & Software Engineering
Large Projects ... Greatest Risks

• Failure to squarely address the problems of **scale** and **complexity**

• Failure to resolve the **imperfect knowledge** associated with large sets of requirements for systems.

• Failure to build the “**right**” system.

• Failure to keep team **productive**.
• Failure to squarely address the problems of **scale** and **complexity**

• Failure to resolve the **imperfect knowledge** associated with large sets of requirements for systems.

• Failure to build the “**right**” system.

• Failure to keep team **productive**.
Threats to Project Success

- Complexity/Scale
- Deficient Requirements
- Satisfying Requirements

These Problems are all interdependent
Problem 1 - Complexity

4.4.8.6 Report XXXXXXX

XXXX health information is requested to aid in planning required XXXX maintenance.

CG2) The XCS shall process each HR (XXXX health request) command message received from the YCS.

CG2.1) An HR command message shall be the first message received after the initiation of each "Manage XXXX" transaction.

CG2.2) The XCS may receive a XXXX health request message anytime during a "XXXX " transaction. (Describes order during transaction.)

CG2.3) A XXXX health request command message will only be accepted by the XCS during an active "XXXX " transaction. (Describes condition under which an HR may be received.)

CG3) The XCS shall prepare and send an HA (XXXX health acknowledgment) message to the YCS in response to an HR (XXXX health request) command message.
“Deficient Requirements are the single biggest cause of project failure”
Inconsistencies among stakeholders – major issue

Problem 2 – Deficient Requirements

Imperfect Knowledge

- Stakeholder-01
  - Vocabulary-01
  - Understanding-01
  - Assumptions-01
  - Needs-01
  - Requirements-01
- Stakeholder-02
  - Vocabulary-02
  - Understanding-02
  - Assumptions-02
  - Needs-02
  - Requirements-02
- Stakeholder-0N
  - Vocabulary-0N
  - Understanding-0N
  - Assumptions-0N
  - Needs-0N
  - Requirements-0N

Brooks’ Tarpit

Composition Tree → Behavior Trees

Shared Conceptualization

- Requirements
- Integrated Views
- Shared Vocabulary
- Shared Understanding
- Shared Assumptions
Problem 3 - Satisfying Requirements - Example

Use-case

- Receive suggested dates for course
- Send out fliers for the course
- Wait for a predetermined length of time
- When time is up, assess the response
- If the response is sufficient, send details to attendees
- If response is insufficient, cancel the course

Vocabulary Changes
1. “Course” omitted
2. “Participants” introduced
3. “Attendees” omitted
4. “Predetermined length of time” omitted
5. “Setting up” introduced
6. “Entry” introduced
7. “Do” introduced
8. “Call for” introduced

State-Chart

Loss of original intention – major issue

Claimed to be “equivalent” to text version on page .226
“The hardest single part of building a software system is deciding what to build, … No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later”

F.P. Brooks
The Greatest Challenge

Scale & Complexity

Imperfect Knowledge

4.4.8.6 Report XXXXXXX

XXXXX health information is requested to aid in planning required XXXXX maintenance.

CG2) The XCS shall process each HR (XXXX health request) command message received from the YCS.

CG2.1) An HR command message shall be the first message received after the initiation of each "Manage XXXXX" transaction.

CG2.2) The XCS may receive a XXXXX health request message anytime during a "XXXXX" transaction. (Describes order during transaction.)

CG2.3) A XXXXX health request command message will only be accepted by the XCS during an active "XXXXX" transaction. (Describes condition under which an HR may be accepted.)

CG3) The XCS shall prepare and send an HA (XXXXX health acknowledgment) message to the YCS in response to an HR (XXXXX health request) command message.

Large Teams Involved

The Right System
Build system to satisfy the requirements

- **Imply**: other methods require a “miracle” to go from requirements to code (p.611)

- **Claim**: “Catalysis reduces such magic” but you need to read 688 pages to find out

- **Advocate**: “Treat your system as a single object, define the type of any system implementation that would meet the requirements” (p.596)

Comment – Perhaps a “smaller” miracle
Current Software Engineering - Strategy

Build system to satisfy the requirements

This is TOO Hard

Current SE Methods — Fail to consistently deliver
Recent Failures – June Verner (NICTA)

Most organizations try to hide their failures
A recent “Hall of Shame” IEEE Spectrum, Nov. 2005:
(in $US millions)

- FBI 100
- UK Inland Revenue 33
- Ford Motor Company 400
- Sainsbury’s 527
- Sydney Water Corp 32.2

Recent Australian problems include:

- National Australia Bank’s ERP project,
- RMIT’s Academic Management System,
- Victoria State’s Infrastructure Management System,
- Federal Government’s new sea cargo import reporting system
- But there are many more……..

Current methods buckle under complexity
We need an Alternative way of Thinking
“All mathematics exhibits in its conclusions only what is already implicit in its premises -- all mathematical derivation can be viewed simply as change in representation, making evident what was previously true but obscure.”

- Herbert Simon
Build system OUT OF its requirements

REPRESENTATION – is the key to doing this
Build system
OUT OF
its requirements

Implies can deal with
ONE Requirement
At a time

REPRESENTATION – is the key to doing this
Deficient Requirements

Quality Software

Complexity

Deficient Requirements

Satisfying Requirements

Deal with ONE Requirement at a time

Build System OUT OF Requirements

Requirements Translation & Requirements Integration

PROBLEMS

Strategy

Tackling Complexity Head-on
Behavior Engineering

PROBLEMS

Complexity

Deficient Requirements

Satisfying Requirements

Strategy

Build System OUT OF Requirements

Deal with ONE Requirement at a time

Requirements Translation & Requirements Integration

Tackling Verification & Validation Head-on
Behavior Engineering

PROBLEMS

- Complexity
- Deficient Requirements
- Satisfying Requirements

Strategy

- Build System OUT OF Requirements
- Deal with ONE Requirement at a time

Quality Software

Tackling Deficient Requirements Head-on
Tackling Software Engineering’s Problems Head-on

PROBLEMS
- Complexity
- Deficient Requirements
- Satisfying Requirements

Quality Software

Strategy/Process
- Build System OUT OF Requirements
- Deal with ONE Requirement at a time
- Requirements Translation & Requirements Integration

Integrated Views
- Integrated Behavior Tree (IBT)
- Integrated Composition Tree (ICT)

Behavior Engineering
Tackling Complexity
The stumbling block with complexity is the limitations of Human Short-term Memory.
People don’t mind dealing with complexity provided it is Localised.
The Starting Point
Requirements in Natural Language

- Large numbers of requirements overflow our short-term memory
- Ambiguity, and many other types of defects are not “visible” in sequential text
- Can’t grasp what system does as a “whole”
- How do we organize teams to work productively?
• **Accuracy** – How to preserve original intent?
• **Validation** – Understandable by stakeholders?
• **Complexity** – Avoid short term memory overflow
• **Defects** – To make defects “visible”
• **Comprehending** – To see as a “whole”
• **Dividing up the work** – Productive teams?

Formalization Strategy ???
How do we do all this?
Build system out of its requirements
The Link – Build systems out of requirements
Two Types of Behavior

- **Component behavior** - component acting
- **Network behavior** - components interacting

The soccer player behavior versus soccer team behavior
Formalization Using Behavior Trees

Component behavior - component acting

Realizing a STATE

<table>
<thead>
<tr>
<th>AIRCRAFT [ Landed ]</th>
</tr>
</thead>
</table>

Changing STATEs

<table>
<thead>
<tr>
<th>AIRCRAFT [ Taking_Off ]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AIRCRAFT [ Airborne ]</th>
</tr>
</thead>
</table>
Network behavior - components interacting

DOOR [Open]

LIGHT [On]

Passing Control or Data
“Whenever the door becomes open it causes the light to go on”

Components?

States?

Behavior Tree
Step 1. Requirements Translation
When a car arrives, if the gate is open the car proceeds, otherwise if the gate is closed, when the driver presses the button it causes the gate to open.
When a car arrives, if the gate is open the car proceeds, otherwise if the gate is closed, when the driver presses the button it causes the gate to open.

Informal => Formal
R1. There is a single control button available for the user of the oven. If the oven is idle with the door closed and you push the button, the oven will start cooking (this is, energize the power-tube for one minute).

R2. If the button is pushed while the oven is cooking it will cause the oven to cook for an extra minute.

R3. Pushing the button when the door is open has no effect (because it is disabled).

R4. Whenever the oven is cooking or the door is open the light in the oven will be on.

R5. Opening the door stops the cooking.

R6. Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
Requirements Translation

Original Requirements Information (Informal)

TRANSLATION Defect Detection Correction

Requirements Behavior Tree (Formal)

GOALS of Translation:
- To Preserve meaning
- To Clarify meaning
- Not to Add anything
- Not to leave out anything
- Not to modify anything
- Not to change vocabulary
Requirements Translation - Ambiguity

There are at least three interpretations of this behavior – each has a different formal representation – author must validate which one is intended.
2.2.1 Report Satellite Health
Satellite health information is requested to aid in planning required satellite maintenance.

MG2) The SCS shall process each HR (satellite health request) command message received from the GCS.

MG2.1) An HR command message shall be the first message received after the initiation of each "Manage Satellites" transaction.

MG2.2) The SCS may receive a satellite health request message anytime during a "Manage Satellites" transaction. (Describes order during transaction.)

MG2.3) A satellite health request command message will only be accepted by the SCS during an active "Manage Satellites" transaction. (Describes condition under which an HR may be received.)

MG3) The SCS shall prepare and send an HA (satellite health acknowledgment) message to the GCS in response to an HR (satellite health request) command message.

FR-01
2.2.1 Report Satellite Health
Satellite health information is requested to aid in planning required satellite maintenance.

FR-02
2.2.1 Report Satellite Health
Satellite health information is requested to aid in planning required satellite maintenance.

FR-M
2.2.1 Report Satellite Health
Satellite health information is requested to aid in planning required satellite maintenance.

Behavior Tree
Behavior Tree Notation
A Behavior Tree notation diagram is depicted, showing the sequence of events:

1. Event
2. CAR ?? Arrives ??
   - GATE ? Open ?
     - CAR [Proceeds]
   - GATE ? Closed ?
     - DRIVER ??[(Presses]Button]??
       - BUTTON [Pressed]
         - GATE [Open]
Behavior Tree Notation

**Behavior Tree Notation**

State Realization

1. **CAR**
   - ?? Arrives ??

2. **GATE**
   - ? Open ?
   - 1. **CAR** [Proceeds]

3. **GATE**
   - ? Closed ?
   - 1. **DRIVER**
     - ?? [Presses Button] ??
     - 1. **BUTTON**
       - [Pressed]
       - 1. **GATE**
         - [Open]
Reversion = “^”

Revert back and repeat earlier behavior

---

Behavior Tree Notation
Behavior Tree Notation

Data-flow

Sender
< Data >

Receiver
> Data <
1. The telephone is on ...  

2. When the telephone rings ...  

3. If the telephone is on ...  

4. The telephone sends message ...  

5. The telephone receives message...  

6. Telephone number assigned ...
### Behavior Tree Notation

<table>
<thead>
<tr>
<th>Component-State</th>
<th>Label</th>
<th>Semantics</th>
<th>Composition Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="#" alt="tag" /></td>
<td>Component</td>
<td><img src="#" alt="tag" /> State - Realization</td>
<td><img src="#" alt="tag" /> A [ S1 ] <img src="#" alt="tag" /> A &lt; Data &gt;</td>
</tr>
<tr>
<td><img src="#" alt="tag" /></td>
<td>Component</td>
<td><img src="#" alt="tag" /> Attribute - Assignment</td>
<td><img src="#" alt="tag" /> B [ S2 ] <img src="#" alt="tag" /> B &gt; Data &lt;</td>
</tr>
<tr>
<td><img src="#" alt="tag" /> ?</td>
<td>Component</td>
<td><img src="#" alt="tag" /> Conditional - flow of control</td>
<td><img src="#" alt="tag" /> A [ S1 ] <img src="#" alt="tag" /> C [ S3 ]</td>
</tr>
<tr>
<td><img src="#" alt="tag" /> ? ?</td>
<td>Component</td>
<td><img src="#" alt="tag" /> Event - flow of control</td>
<td><img src="#" alt="tag" /> B [ S2 ] <img src="#" alt="tag" /> C [ S3 ]</td>
</tr>
<tr>
<td><img src="#" alt="tag" /> ? ? ?</td>
<td>Component</td>
<td><img src="#" alt="tag" /> Guard - flow of control</td>
<td><img src="#" alt="tag" /> A [ S1 ] <img src="#" alt="tag" /> C [ S3 ]</td>
</tr>
<tr>
<td><img src="#" alt="tag" /></td>
<td>Component</td>
<td><img src="#" alt="tag" /> Data Output State</td>
<td><img src="#" alt="tag" /> A [ S1 ] <img src="#" alt="tag" /> C [ S3 ]</td>
</tr>
<tr>
<td><img src="#" alt="tag" /></td>
<td>COMPONENT</td>
<td><img src="#" alt="tag" /> &gt; Data_Out &lt;</td>
<td><img src="#" alt="tag" /> A [ S1 ] <img src="#" alt="tag" /> C [ S3 ]</td>
</tr>
<tr>
<td><img src="#" alt="tag" /></td>
<td>System</td>
<td><img src="#" alt="tag" /> [ State ]</td>
<td><img src="#" alt="tag" /> A [ S1 ] <img src="#" alt="tag" /> C [ S3 ]</td>
</tr>
</tbody>
</table>

**Core Elements**
The BT notation captures in a simple tree-like form of composed component-states what usually needs to be expressed in a mix of other representations.

The language elegantly captures three types of inter-process communication: *shared variable*, *synchronization* and *message passing*.

BTs have been given a formal semantics based on a low-level process algebra, BTPA.

BTs can be used to support model-checking simulation and code generation.
Where to Next?
Step 2.

Requirements Integration

Putting the pieces together
The pieces of a jigsaw puzzle (and model toy kits) have the interesting (genetic) property that:

they contain enough information to allow the pieces to all be assembled one at a time.
Creating an Integrated View

- Order is not important but the position where placed is.
- Information about “f” is spread across three pieces.
- Only see there are missing pieces when integrate pieces.
- Only see some pieces don’t fit when integrate pieces.

Same IDEAS apply with requirements.
A set of requirements also:

contain enough information to allow them to all be assembled

into an integrated view which becomes a precursor to the system design

Proviso - you need to use the right representation
Each and every functional requirement expressed as a behavior tree $\text{BT}_i$ has associated with it a precondition $\text{P}_i$ that needs to be satisfied in order for the behavior encapsulated within it to be exhibitable.
Requirements Integration

R1 CAR-SYSTEM [ Park ]

R1 DRIVER ?? [[Inserted]Key] ??

R1 KEY [Inserted]

R1 DRIVER ??[[[Turned]Key] ??

R1 KEY [Turned]

R1 + IGNITION [ on ]

R1 CAR-SYSTEM [ Started ]

R5 IGNITION [ on ]

Root Node

Matching Node
R1 DRIVER
??Inserts-Key??

R1 DRIVER
??Turns-Key??

R1 INTEGRATION
[on]

R1 CAR-SYSTEM
[Park]

R1 CAR-SYSTEM
[Started]

R5 HAND-BRAKE
?On?

R5 BRAKE-LIGHT
[On]

Point of Integration
Find where root node of one tree occurs in another tree – JOIN at that point.
How Does This Help?

If we have a large number of requirements each can be integrated into a behavior tree ONE AT A TIME

It allows us to deal with complexity
Creating an Integrated Behavioral View From Requirements
Requirements Integration
Build system out of its requirements
An Example
TRAIN-STATION PROBLEM (Sherwood Station)

Develop a system to model the behavior of a Train-Station. You need to model a train entering the station from the north and then leaving the station to the south. A crossing with boom gates and flashing red lights is located just south of the station. There is a signal to the north of the station that only allows a train to enter when the station is not occupied, that is, when the north signal is green. There is also an exit signal light that ensures the train can only leave the station when the boom gates are down. There is also a north detector that can detect the train approaching the station region from the north. And, there is an exit detector that detects when a train leaves to the south.

1. Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north a train is detected, by the north detector, which causes the north signal to turn red.

2. When the north detector detects a train it causes the crossing lights to start flashing red. At the same time, a timer starts timing and when it times out it causes the boom gates to be lowered after which the exit light turns green.

3. After the train is detected the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

4. After the doors close the train may leave the station only when and if the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train leaving, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

For the purposes of the exercise ignore trains approaching the station from the south. This additional requirement can be integrated later as a separate exercise. Also ignore situations where the train does not stop at the station - this too requires some refinements to the design.
Step 1.

Requirements Translation
REQUIREMENT-R1
Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north, a train is detected by the north detector, which causes the north signal to turn red.
**R2 – Translated Behavior Tree**

**REQUIREMENT-R2**
When the north detector detects a train it causes the crossing lights to start flashing red. At the same time a timer starts timing and when it times out, it causes the boom gates to be lowered, after which the exit light turns green.
REQUIREMENT-R3
After the train is detected by the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

Implied (+)
REQUIREMENT-R4
After the doors close the train may leave the station provided the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector, which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.
Step 2.

Requirements Integration

Putting the pieces together
Integration – Base Case

1. STATION [ NOT( Occupied ) ]

1. NORTH-SIGNAL [ Green ]

1. TRAIN# ?? Approaching / ??
   where [ from ] North

1. NORTH-DETECTOR ?? Train[Detected]??

1. NORTH-SIGNAL [ Red ]
Integration of R2 with R1
Integration of R2 with R1

Point of Integration

1. STATION [NOT (Occupied)]
2. NORTH-SIGNAL [Green]
1. TRAIN? Approaching / ??
   from North
   NORTH-DETECTOR ?? Train Detected ??
2. CROSSING-LIGHTS [Flashing-Red]
2. TIMER [Starts Timing]
2. TIMER ?? Times-out ??
2. BOOM-GATES [Lowered]
2. EXIT-LIGHT [Green]
Integration of R3 into IBT

Point of Integration

1. STATION [NOT (Occupied)]
   - NORTH-SIGNAL [Green]
   - TRAIN# ??Approaching / ??
     - where [at]
       - NORTH DETECTOR ??Train[Detected]??

2. CROSSING-LIGHTS [Flashing-Red]
   - TIMER [[Starts]Timing]
     - ??Times-out ??

3. BOOM-GATES [Lowered]
   - EXIT-LIGHT [Green]

3. TIMER ??Times-out ??
3. TRAIN ??Arrived / ??
   - where [at]
     - STATION
   - DOORS [Open]
   - PEOPLE [Disembark]
   - DOORS ??Close ??
3. STATION [Occupied]
Integration of R3 into IBT

Point of Integration

1. STATION [NOT( Occupied )]

1. NORTH-SIGNAL [Green]

2. TRAIN# ??Approaching / ??

where (from) North

1. NORTH-DETECTOR ??Train[Detected]??

1. CROSSING-LIGHTS [Flashing-Red]

2. TIMER [[Starts]Timing]

3. TRAIN# ??Arrived / ??

where (at) STATION

2. TIMER ??Times-out ??

3. DOORS [Open]

3. STATION [Occupied]

2. BOOM-GATES [Lowered]

2. EXIT-LIGHT [Green]

3. PEOPLE [Disembark]

3. DOORS ??Close ??
Integration of R4 into IBT
Integration of R4 into IBT

1. STATION [NOT Occupied]
2. CROSSING LIGHTS [Flashing Red]
3. TIMER [Starts Timing]
4. BOOM-GATES [Lowered]

2. STATION [Occupied]
3. TRAIN [Detected]
4. NORTH-SIGNAL [Green]

3. STATION [Occupied]
4. CROSING LIGHTS [Flashing Red]
5. TIMER [Times out]

4. STATION [Occupied]
5. TRAIN [Leaves Station]
6. DOORS [Open]

5. STATION [Occupied]
6. PEOPLE [Disembark]
Second Example
INT-01 (Base Case 2.1 Initialization)

Here the time period needs to be independent of any restarts that the system may have. It is set up as an independent thread it will kill off all other behavior when it reverts.

NOTE: Have assumed a System-error reversion does not kill off the periodic time-out of the system.
INT-02 (Integrating 2.2 Manage Satellites)
INT-05 (Integrating 2.2.3 Maintain BL Table)
INT-06 ( Integrating 2.3.1 Establish Uplink and Downlink Site Connection )
INT-07 (Integrating 2.3.2 Report Site Readiness)
INT-08 (Integrating 2.3.2 Uploading-Downloading Packets)
INT-09 (Integrating 2.3.3 Handling Data-packet Errors)
INT-11 (Integrating 2.3.5 Terminate Customer Time-slot)
Integrated Views
- Advantages
The process of creating an integrated behavior tree does a sophisticated re-ordering of the original textual information needed so that it can be understood deeply, accurately and quickly without taxing and exceeding the limits of our short-term memory.
A second important advantage of the IBT representation is that the process of translation and integration uncovers incompleteness, inconsistency and redundancy defects in the original text that are otherwise very hard to discover because of the limitations of our short-term memory.
**NOTE:**
First sentence – in text
Near **bottom** of Behavior Tree

**PROBLEM STATEMENT**

It is required to do an analysis and high-level design for part of the functional requirements for a computerised Car System with an automatic gear-shift. The following details apply:

The car can only be started if it is in the PARK state when the driver inserts the key in the ignition and turns it on. A dashboard light remains on if the driver’s seatbelt is not fastened when the driver is seated and the ignition is on. If the handbrake is on when the ignition is on, the brake-light turns on. The security alarm is on when the car is locked, and if anyone tries to break into by breaking a window or forcing a door the alarm will sound. When the driver, on approaching the car, presses the key-button it unlocks the door and turns the security alarm off.

Model the whole process from the driver pressing the key-button, to the driver seated, to having started the engine.
NOTE:
First sentence – in text
Near **bottom** of Behavior Tree

**PROBLEM STATEMENT**

It is required to do an analysis and high-level design for part of the functional requirements for a computerised Car System with an automatic gear-shift. The following details apply:

The car can only be started if it is in the **PARK** state when the driver inserts the key in the **ignition** and turns it **on**. A dashboard light remains on if the driver’s seatbelt is not fastened when the driver is seated and the ignition is on. If the handbrake is on when the ignition is on, the brake-light turns on. The security alarm is on when the car is locked, and if anyone tries to break into the car by breaking a window or forcing a door the alarm will sound. When the driver, on approaching the car, presses the key-button it unlocks the door and turns the security alarm off.

Model the whole process from the driver pressing the key-button, to the driver seated, to having started the engine.
Satellite Control System

Integrated View

Compared with

23-page document of functional requirements
• We have applied the processes of requirements translation and integration to a large number of systems – the largest system we have worked on had 1500 requirements.

• We have been encouraged by the results we have obtained from these trials.

• It is clear that we need a tool-set that makes it practical for teams of people to apply the method.
Integrated Views

- Emergent Properties
Integrated Behavior Tree

Result of integrating eight functional requirements

Functional Requirements are Localized in IBT
Components are **Dispersed** across requirements
Component Behavior Projection

Oven Component = System Component
System – Network of Component Interactions

Architecture Transformation

Each Component Occurs **MANY** times

All behaviors merged in together

Each Component Occurs just **ONCE**
Architecture Transformation

All behaviors merged in together

System – Network of Component Interactions
Microwave Oven - High-Level Behavior
(System States Only)

1. OVEN {Cooking}
   +-----------------+-----------------+
   | 2 | OVEN {Extra-Minute} |
   | 5 | OVEN {Cooking-Stopped} |
   +-----------------+-----------------+
   | 2 | OVEN ^ {Cooking} |
   | 7 | OVEN {Timed-Out} |
   +-----------------+-----------------+
   | 1 | OVEN {Cooking-Finished} |
   +-----------------+-----------------+
2. OVEN [Open]
3. OVEN [Idle]
4. OVEN ^ [Open]
Microwave Oven - Detailed Behavior
(System States + Component States)

System State

User "Door-Closed?"

System State

User "Door-Open?"

System State

User "Door-Open?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"

System State

User "Door-Closed?"
Car-booking System -
High-Level Behavior
(System States Only)
Car-booking System - Detailed Behavior
(System States + Component States)
Behavior – Including System States

Abstract versus expanded description of behavior
Actions on Integrated Views
Actions On

- Inspection/defect detection
- Model-checking
- Defect correction
- Refinement => Specification => Design

And then:

- Component Behavior Projections
- Component Interface Specifications
- Architecture Transformation
- Integration specification
- Code Generation
Informal Requirements

R1. There is a single control button available for the user of the oven. If the oven is idle
with the door closed and you push the button, the oven will start
cooking (this is, energize the power-tube for one minute).
R2. If the button is pushed while the oven is cooking it will cause the oven to cook for an
extra minute.
R3. Pushing the button when the door is open has no effect (because it is disabled).
R4. Whenever the oven is cooking or the door is open, the light in the oven will be on.
R5. Opening the door stops the cooking.
R6. Closing the door turns the cooking off. If the oven stops, the light and the power-tube get turned off, and then a timer
starts a count down that terminates when the timer is finished.

Requirements Integration

Building Dependable Systems
END

Part 1
Behavior Engineering

Tackling Imperfect Knowledge
Problem 2 - Imperfect Knowledge

Inconsistencies among stakeholders
R6. If you close the door, the light goes **out**. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

R7. If the oven times out, it turns **off** both the power tube and the light. It then emits a warning beep to tell you the food is ready.
R6. If you close the door, the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

R7. If the oven times out, it turns off both the power tube and the light. It then emits a warning beep to tell you the food is ready.

Implication

We could formalize each requirement independently but we would end up with an inconsistent vocabulary.

- We have to overcome this problem.
- Challenge when 100s requirements
We use a second Integrated View to solve problem
We saw earlier
The Link – Build systems out of requirements
Increase Order  =>  Remove Imperfections

Requirements Versus Systems

Requirements

Systems

Information (poorly ordered)

Information (well-ordered)
Requirements Versus Systems

- Requirements for systems contain INFORMATION.

- Systems that satisfy requirements contain INFORMATION.

The Link – Build systems out of their requirements
Where to Start => Understanding

• Confronted with a statement of requirements our job is to systematically and effectively increase our understanding of the problem to be solved.

• To increase understanding we need to create useful, usable, new order in a repeatable, constructive way.

• It turns out that constructing the system composition is probably the most effective way to do this and thereby initiate the analysis/design phase.

ROLE OF SYSTEM COMPOSITION
Composition is a concept that is widely used in a number of disciplines to provide useful summary information about an entity.
Examples

- HOUSE: 4 bedrooms, 2 bathrooms, ...
- ETHANE: $\text{C}_2\text{H}_6$
- DICTIONARY:
  Table: "A piece of furniture consisting of a flat top set horizontally on legs"

Relevant for analyzing/design of large systems
A Way to Look at Things - Chemistry

Representations

1. COMPOSITION
   - C₂H₆

2. STRUCTURE
   - H \( \text{C} \text{C} \text{H} \text{H} \text{H} \text{H} \text{H} \)

3. BEHAVIOR
   - Detailed description of behavior
What is Important About Composition

**Composition** is a fundamental property of a system.

**Composition** is a fundamental property of a set of functional requirements of a system.

Properties can be identified repeatably.
It should be UNIQUE for a given statement of requirements.
The **System Composition** plays a role in system design of comparable importance to the role **laying the foundations** plays in constructing a house – it comes first and it supports all subsequent activities.

**Complete Vocabulary ➔ Well-defined Property**
The problem we face in attempting to build an understanding of the components in a system is that in statements of requirements for a system, information about an individual component in the system is usually **widely spread** throughout the set of requirements.

Component composition $\Rightarrow$ System Composition
The Problem We Face

Integrated Behavior Tree

Result of integrating eight functional requirements

DOOR Component Mentioned in R6, R8, R5

Individual Component Info dispersed across requirements
Creating an Integrated View

Information about “f” is spread across THREE pieces

Integrated View – Component “Picture” Emerges
Composition Trees
Composition Tree Form

Component-01

- Encapsulated Component Hierarchy
- Encapsulated Information Vocabulary
- Encapsulated Behavior Vocabulary
- Component-01 Interaction Vocabulary

Component-02

Only present if Component-01 is a "system" component
Composition Tree – Station System

Component Hierarchy

System-of-Systems Integrated View
Composition Tree – Station System

All Information about a single component is in one place
Composition Trees

- Provide an integrated view of data requirements
- Provide integrated knowledge of each component
- Provide a systematic way of finding many types of defects
- Approach repeatability of construction
- Provide information that supports subsequent steps
- Provide an important perspective of the size/dimensionality of the large system
- Provide vital information that supports understanding and subsequent maintenance of the system
- Provide information that can be easily and usefully refined during later stages of development
- Identify important system architecture information
- Serve to construct the vocabulary of a system
Earlier We Saw

Text $\Rightarrow$ Behavior Trees
Now We Want to Consider

Text → Composition Trees
Requirements Translation
+
Integration
Creating an Integrated Compositonal View From Requirements
TRAIN-STATION PROBLEM (Sherwood Station)

Develop a system to model the behavior of a Train-Station. You need to model a train entering the station from the north and then leaving the station to the south. A crossing with boom gates and flashing red lights is located just south of the station. There is a signal to the north of the station that only allows a train to enter when the station is not occupied, that is, when the north signal is green. There is also an exit signal light that ensures the train can only leave the station when the boom gates are down. There is also a north detector that can detect the train approaching the station region from the north. And, there is an exit detector that detects when a train leaves to the south.

1. Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north a train is detected, by the north detector, which causes the north signal to turn red.

2. When the north detector detects a train it causes the crossing lights to start flashing red. At the same time, a timer starts timing and when it times out it causes the boom gates to be lowered after which the exit light turns green.

3. After the train is detected the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

4. After the doors close the train may leave the station only when and if the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train leaving, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

For the purposes of the exercise ignore trains approaching the station from the south. This additional requirement can be integrated later as a separate exercise. Also ignore situations where the train does not stop at the station - this too requires some refinements to the design.
REQUIREMENT-R1
Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north, a train is detected by the north detector, which causes the north signal to turn red.
Train Station System

R1

Relations

<table>
<thead>
<tr>
<th>R1</th>
<th>TRAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ Approaches ]</td>
</tr>
</tbody>
</table>

where [from]

North

<table>
<thead>
<tr>
<th>R1</th>
<th>TRAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ Detected ]</td>
</tr>
</tbody>
</table>

detected [by]

NORTH_DETECTOR

States

<table>
<thead>
<tr>
<th>R1</th>
<th>NORTH_SIGNAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ Green ]</td>
</tr>
</tbody>
</table>

Relations

<table>
<thead>
<tr>
<th>R1</th>
<th>NORTH_DETECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ Detects ]</td>
</tr>
</tbody>
</table>

what

TRAIN

States

<table>
<thead>
<tr>
<th>R1</th>
<th>NORTH_DETECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ NOT (Occupied) ]</td>
</tr>
</tbody>
</table>

Requirements Translation - One at a time
REQUIREMENT-R2
When the north detector detects a train it causes the crossing lights to start flashing red. At the same time a timer starts timing and when it times out, it causes the boom gates to be lowered, after which the exit light turns green.
Train Station System

R1

Relations

TRAIN

NORTH_SIGNAL

NORTH_DETECTOR

Relations

TRAIN

[Approaches /]

NORTH

[Green /]

Red

NORTH_DETECTOR

States

Green

Red

NORTH_DETECTOR

States

Flashing_Red

NORTH

Detect

TRAIN

States

Timer

BOOM_GATES

States

Timed_Out

[Lowered /]

[Green /]

EXIT_LIGHT

States

Not_Occupied

Integrating

R2

Requirements Translation PLUS Integration
REQUIREMENT-R3
After the train is detected by the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.
REQUIREMENT-R4
After the doors close the train may leave the station provided the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector, which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.
TWO

Integrated Views
Tackling Software Engineering’s Problems Head-on

Deficient Requirements
Quality Software
Complexity
Satisfying Requirements

Strategy/Process
Build System OUT OF Requirements
Deal with ONE Requirement at a time
Requirements Translation & Requirements Integration

Integrated Views
Integrated Behavior Tree (IBT)
Integrated Composition Tree (ICT)

Two Integrated Views
Integrated Behavior Tree - IBT

Benefits of Behavior Integration

- See system behavior as a whole
- Integration - detects requirements defects
- Can refine => Specification => Design
- Model-checking, simulation, code-gen.
- Aids component design & implementation

First Integrated View
**Integrated Composition Tree - ICT**

**Benefits of Composition Integration**
- All information about a component - is in ONE place
- Aids component design and implementation
- Integration - Helps detect inconsistent information
- Provides complete system vocabulary – all in context

---

Second Integrated View
Deal with ONE Requirement at a time

Build System OUT OF Requirements

Integrated Views

Integrated Behavior Tree (IBT)

Integrated Composition Tree (ICT)

Strategy/Process

Where We Are Up To

Integrated Composition Tree (ICT)

Integrated Behavior Tree (IBT)

Work Products

Component Behavior

Components Interactions

Integration Specification

System Vocabulary

Components' Compositions

Components Interfaces

System-of-Systems Hierarchy

Two Integrated Views ✅
Where Are We Up To

- Scale & Complexity ✔️
- Build Right System ✔️
- Imperfect Knowledge ✔️
- Productivity ❌
Tackling Team Productivity
Collaborative Editing

Development
By
Teams

Integrated Composition Tree + BTs in Parallel
Collaborative Editing - Advantages

- Team members translate subsets of requirements.
- Integrated Composition Tree provides strict progressive vocabulary consistency.
- Each team member sees dynamically how the work of others affects their work.
- Practical, transparent way to combine the work of individual team members.
- Reduces project team communication overhead.

Potential for significant productivity gains
Microwave Oven – Functional Requirements†

R1. There is a single control button available for the user of the oven. If the oven door is closed and you push the button, the oven will cook (that is, energize the power-tube) for 1 minute.

R2. If you push the button at any time when the oven is cooking, you get an additional minute of cooking time.

R3. Pushing the button when the door is open has no effect.

R4. There is a light inside the oven. Any time the oven is cooking, the light must be turned on. Any time the door is open, the light must be on.

R5. You can stop the cooking by opening the door.

R6. If you close the door, the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

R7. If the oven times out, it turns off both the power tube and the light. It then emits a warning beep to tell you the food is ready.

†After Shlaer and Mellor, Object Life Cycles, p.36
If you close the door the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

Finding Alias Defects
Finding alias defects
Work
With Industry
Behavior Engineering trials on a series of large projects with one large company consistently found 10 – 15% of requirements analyzed contained significant defects not found by their review processes.

Company is a CMMI company with mature processes.

Similar statistics on projects for other large companies and organizations.
The following table contains statistics on recent projects where we have applied the method.

<table>
<thead>
<tr>
<th></th>
<th>Recent Project- RP</th>
<th>Last 3 Projects – (RP excluded)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Pages Analysed:</strong></td>
<td>101pages</td>
<td>265</td>
</tr>
<tr>
<td><strong>Number of Requirements Analysed:</strong></td>
<td>920requirements</td>
<td>3142</td>
</tr>
<tr>
<td><strong>Major Defects Only</strong></td>
<td>128defects</td>
<td>412</td>
</tr>
<tr>
<td>Incompleteness</td>
<td>73</td>
<td>260</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>Ambiguity</td>
<td>19</td>
<td>93</td>
</tr>
<tr>
<td>Redundancy</td>
<td>31</td>
<td>13</td>
</tr>
<tr>
<td>Inaccuracy</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td><strong>Number of Queries:</strong></td>
<td>7queries</td>
<td>98</td>
</tr>
<tr>
<td><strong>Effort (Includes reporting, analysis, modeling)</strong></td>
<td>94Person-hours</td>
<td>325</td>
</tr>
<tr>
<td><strong>Effort (Includes reporting, analysis, modeling)</strong></td>
<td>94Person-hours</td>
<td>325</td>
</tr>
<tr>
<td><strong>Effort (Includes reporting, analysis, modeling)</strong></td>
<td>94Person-hours</td>
<td>325</td>
</tr>
</tbody>
</table>

What the results show is that the Behavior Engineering method consistently finds 130 **major** defects per 1000 of requirements **after** normal reviews and correction have been carried out. In addition the integrated work products **constructed to detect defects** can subsequently be corrected and refined to create an executable design.
Where have we got to?
Threats to Producing Quality Software

These Problems are all interdependent

- Complexity/Scale
- Deficient Requirements
- Satisfying Requirements
Deficient Requirements

Quality Software

Deal with ONE Requirement at a time

Build System OUT OF Requirements

Strategy

PROBLEMS

Complexity

Deficient Requirements

Satisfying Requirements

Tackling Complexity Head-on
Deficient Requirements

Quality Software

Complexity

Satisfying Requirements

Strategy

Build System OUT OF Requirements

Deal with ONE Requirement at a time

Requirements Translation & Requirements Integration

PROBLEMS

Tackling Deficient Requirements Head-on
Deficient Requirements
Complexity
Deficient Requirements
Satisfying Requirements

PROBLEMS

Quality Software

Strategy
Build System OUT OF Requirements
Deal with ONE Requirement at a time
Requirements Translation & Requirements Integration

Tackling Verification & Validation Head-on
Informal Requirements

R1. There is a single control button available for the user of the oven. If the oven is idle with the door closed and you push the button, the oven will start cooking (this is, energize the power-tube for one minute).

R2. If the button is pushed while the oven is cooking it will cause the oven to cook for an extra minute.

R3. Pushing the button when the door is open has no effect (because it is disabled).

R4. Whenever the oven is cooking or the door is open the light in the oven will be on.

R5. Opening the door stops the cooking.

R6. Closing the door turns off the light. This is the normal idle state, prior to cooking when the door has been closed and the oven is not cooking.

R7. If the oven times-out, the light and the power-tube are turned off and then a beeper emits a sound to indicate that the cooking is finished.
Simple, Scaleable Development

- **Accuracy** – individual requirements translation
- **Validation** – preserve original vocabulary
- **Complexity** – deal one requirement at a time
- **Defects** – rigorous translation, integration, MC
- **Comprehending** – requirements integration
- **Dividing up the work** – single requirement focus

Towards Quality Software
“There are two ways of acquiring knowledge … Argument reaches a conclusion and compels us to admit it, but it neither makes us certain nor so annihilates doubt that the mind rests calm in the intuition of truth, unless it finds this certitude by way of experience”

- Roger Bacon, 1268 AD
... and more information

www.behaviorengineering.org
www.accs.edu.au
I would like acknowledge the contribution of my colleagues at the ARC Centre for Complex Systems at University of Queensland and my colleagues and students at Griffith University who have contributed to this work. I would also like to thank the many people in Industry and academia who have supported and encouraged me in progressing this work over the last seven years.
“If you keep doing what you have always done, you will keep getting what you have always got”.

– W. Edwards Deming
“I believe that failure is less frequently attributable to either insufficiency of means or impatience of labour than to a confused understanding of the thing actually to be done.”

John Ruskin