dependences & dependability

Daniel Jackson, MIT
HDCP Review
Ames, June 18, 2003
dependability

dependable software
› ‘the software works’
› will it ever be a reality?

no, because for most systems
› requirements are complex
› codebase is large
› bugs are inevitable

so, change viewpoint
› dependable properties, not systems
› ‘with high probability, no catastrophes’
› example: ‘emergency stop button works’
guaranteeing properties

an approach
› identify properties & concerns
› design to encapsulate properties
› determine scope from code
› check conformance statically

other elements
› conformance monitors
› ‘software interlocks’

SDG research areas
› problem frames
› dependency model
› assumption trees

Alloy analysis

in this talk, focus on
› dependency model and assumption trees
› because funding primarily from HDCP
dependencies and decoupling

› a key aim in software design
› reduce inter-module dependences
› limit scope of modification & reasoning

standard models are binary
› dependency exists or not
› quantity, not quality

in practice
› more flexible design has more dependences
› want to trace particular properties
› so need a richer model
standard model

module A ‘uses’ module B when
- correct working of A depends on correct working of B

example: Observer

a new model

dependences mediated by specs
› module A has S-use of module C
› means A relies on C satisfying S

dependences mediated by specs
› module A has S-use of module C
› means A relies on C satisfying S

module as specification transducer
› for a given exported spec
› module relies on imported specs

example: module C
› exports S and T
› imports U and V
› transduces
  S \rightarrow U, V
  T \rightarrow V
example: observer pattern

really 2 distinct patterns: Register and Notify
assumption trees

suppose we care about property $P$
› which modules must be checked?

approach
› identify set of partial module specs for $P$
› trace dependences from these, forming a tree
› verify each node in the tree

joint work with Drew Rae
example

suppose P is established by spec R
assumption tree is:
A: R
B: T
  D: V
    Env: X
E: W

checks
A: satisfies R given T
B: satisfies T given V, W
D: satisfies V given X
Env: satisfies X

transducers
A: R->T ; S->T,U
B: T->V,W
D: V->X
application: TSAFE

› design of prototype expressed in model
› undesirable couplings led to changes

Greg Dennis’s masters thesis
application: NPTC

› northeast proton therapy center
› property: emergency stop works
› assumptions discovered
  treatment room is not room 3
  disk is not full, so logging returns
  other processes don’t hog msg queue

analysis by Drew Rae
future work

- automating dependency analysis
  - dependency extractor for Java: prototype complete
  - now working on specification discovery

- automating conformance checking
  - find relevant code within module?
  - extract transducers?

- application to CTAS (with Notkin, Kotov)
  - property: generated advisories don’t lead to conflicts
  - establish with checker and gatekeeper
extra slides
grouping

A
B

C

stands for

A
B
C

A
B
C

stands for

C

A
B

A
B
templates

\[\text{A} \rightarrow_B \text{B}_i \rightarrow_C \text{C} \quad \text{stands for} \quad \text{A} \rightarrow_B \ldots \rightarrow_B \text{B}_n \rightarrow_C \text{C}\]
related work

dependence models in other fields
› Eppinger’s Design Structure Matrix
› Suh’s Axiomatic Design

configuration models
› Units model, Felleisen et al

code dependences
› similar to my modular slicing (FSE 1994)

construction dependences
› make, etc

architectural dependences
› Richardson et al