Formal methods and continuous change

towards a happy marriage

Carlo Ghezzi
DEIB-Politecnico di Milano
ESEC/FSE 2015
Bergamo, September 2015
The talk

- A personal retrospective view— I've been around since ICSE 2 (San Francisco, 1976)!
- A brief look-ahead
- A tribute to others
Software and change
Facts
Facts

• Software undergoes continuous changes
Facts

- Software undergoes continuous changes
- Unrivalled by any other technology
Facts

• Software undergoes continuous changes

• Unrivalled by any other technology

• Can be a problem
Facts

- Software undergoes continuous changes
- Unrivalled by any other technology
- Can be a problem
- **Can be an opportunity**
Evolution: positive view of change

Embeds the notions of

• improvement

• adaptation
Formal methods can be integrated into software engineering practice to effectively and efficiently turn change into evolution.
Formal methods and continuous change

towards a happy marriage

• How did I get there?

Recovering the common thread

*fil rouge, filo rosso*
The world and the machine

We build (abstract) machines to achieve certain real-world goals, satisfy certain requirements

The world and the machine

We build (abstract) machines to achieve certain real-world goals, satisfy certain requirements.

The world and the machine

We build (abstract) machines to achieve certain real-world goals, satisfy certain requirements.

Software engineer's responsibilities
Software engineer's responsibilities

- Develop a specification $S$ for the machine (and an implementation) such, assuming that the environment behaves according to $E$, we can assure satisfaction of $R$
Software engineer's responsibilities

• Develop a specification $S$ for the machine (and an implementation) such, assuming that the environment behaves according to $E$, we can assure satisfaction of $R$

• Formally,
Software engineer's responsibilities

• Develop a specification $S$ for the machine (and an implementation) such, assuming that the environment behaves according to $E$, we can assure satisfaction of $R$

• Formally,

$$E \& S \models R$$
Software engineer's responsibilities

- Develop a specification $S$ for the machine (and an implementation) such, assuming that the environment behaves according to $E$, we can assure satisfaction of $R$.

- Formally,

\[ E \& S \models R \]

Dependability argument
Problem 1: Getting the machine right

- It is an evolutionary process
- Software design is an exploratory activity
- Software evolves from incomplete to a progressively complete and stable solution
- An then the solution becomes unstable to support further evolution
Fil rouge: 1970's view of the problem
Fil rouge: 1970's view of the problem

• Programming methodologies: stepwise refinement, design for change
Fil rouge: 1970's view of the problem

- Programming methodologies: stepwise refinement, design for change

  N. Wirth, Program development by stepwise refinement. CACM 1971
  D. L. Parnas, On the criteria to be used in decomposing systems into modules. CACM 1972
  D. L. Parnas, Designing software for ease of extension and contraction., ICSE '78
Fil rouge: 1970's view of the problem

• Programming methodologies: stepwise refinement, design for change

  N. Wirth, Program development by stepwise refinement. CACM 1971
  D. L. Parnas, On the criteria to be used in decomposing systems into modules. CACM 1972
  D. L. Parnas, Designing software for ease of extension and contraction., ICSE '78

• First generation programming environments (IDEs) to support incremental development
Fil rouge: 1970's view of the problem

- **Programming methodologies: stepwise refinement, design for change**
  
  N. Wirth, Program development by stepwise refinement. CACM 1971  
  D. L. Parnas, On the criteria to be used in decomposing systems into modules. CACM 1972  
  D. L. Parnas, Designing software for ease of extension and contraction., ICSE '78

- **First generation programming environments (IDEs) to support incremental development**
  

  P. Feiler and R. Medina-Mora, An Incremental Programming Environment, ICSE '81

Formally handling changes through incremental parsing

- Intuition: Re-build the minimum sub-tree “covering” the change \( w \), rooted in \( <N> \), and “plug-it-in” the unmodified portion of tree

C. Ghezzi and D. Mandrioli, Incremental parsing, ACM TOPLAS, 1979
C. Ghezzi and D. Mandrioli, Augmenting Parsers to Support Incrementality. J. ACM, 1980
Fil rouge: late 1980's, 1990's
Fil rouge: late 1980's, 1990's

- Software development processes: humans in the automated process
Fil rouge: late 1980's, 1990's

- Software development processes: humans in the automated process

  L. Osterweil, Software processes are software too. IICSE '87
  R. Balzer, Tolerating inconsistency in software development, ICSE '91
Fil rouge: late 1980's, 1990's

• Software development processes: humans in the automated process
  
  L. Osterweil, Software processes are software too. IICSE '87
  R. Balzer, Tolerating inconsistency in software development, ICSE '91

• Can formal methods help?
Fil rouge: late 1980's, 1990's

- Software development processes: humans in the automated process
  
  L. Osterweil, Software processes are software too. IICSE '87  
  R. Balzer, Tolerating inconsistency in software development, ICSE '91

- Can formal methods help?
  
  
  G. Cugola, E. Di Nitto, C.Ghezzi, M. Mantione, How to deal with deviations during process model enactment.  
  ICSE'95
  
Problem 2: Requirements

- Requirements are highly volatile
  - Hard to get
  - Change rapidly
- Satisfaction of certain requirements generate new requirements

\[ E & S \models R \]
Problem 2: Requirements

- Requirements are highly volatile
  - Hard to get
  - Change rapidly
  - Satisfaction of certain requirements generate new requirements

\[ E \ & S \models R \]
Problem 2: Requirements

• Requirements are highly volatile
  • Hard to get
  • Change rapidly
• Satisfaction of certain requirements generate new requirements

\[ E \& S \models R \]
Problem 3: Understanding the environment

• Getting environment properties and assumptions right is hard
Problem 3: Understanding the environment

• Getting environment properties and assumptions right is hard

• **Properties**

  • Domain laws (e.g., physics)
    
    • **R**: move a body from A to B
    
    • **D**: suitable force A->B causes motion A->B
    
    • **S**: send suitable force command to actuator
Problem 3: Understanding the environment

- Getting environment properties and assumptions right is hard
Problem 3: Understanding the environment

• Getting environment properties and assumptions right is hard

• Assumptions

  • Uncertain/incomplete/changeable knowledge
  
  • \textbf{R}: guarantee given avg response time to users
  
  • \textbf{D}: avg traffic X transactions/msec.
Problem 3: Understanding the environment

- Getting environment properties and assumptions right is hard
Problem 3: Understanding the environment

- Getting environment properties and assumptions right is hard
  - Often wrong properties/assumptions are hypothesized
  - Often assumptions made at design time are uncertain
  - Often assumptions change
Problem 3: Understanding the environment

• Getting environment properties and assumptions right is hard
  • Often wrong properties/assumptions are hypothesized
  • Often assumptions made at design time are uncertain
  • Often assumptions change

\[ E & S \models R \]
Problem 3: Understanding the environment

- Getting environment properties and assumptions right is hard
  - Often wrong properties/assumptions are hypothesized
  - Often assumptions made at design time are uncertain
  - Often assumptions change

\[ E \& S \models R \]
Problem 3: Understanding the environment

- Getting environment properties and assumptions right is hard
  - Often wrong properties/assumptions are hypothesized
  - Often assumptions made at design time are uncertain
  - Often assumptions change

\[ E \& S \models R \]
A general notion of failure

• Any indication of broken dependability argument, manifestation that requirements are not (satisfactorily) satisfied

• Functional or nonfunctional
  
  • not necessarily a "catastrophic event"

• includes violation of **quality of service**, which may lead to financial losses, penalties, or damage to reputation

• *Experienced or (if possible) predicted failure drives evolution*
Modern multi-tenant systems

- Rely on third-party components to provide their own service, which make environment volatile
  - Platform as a Service (cloud)
  - Software as a Service

- Reinvigorating Leslie Lamport's statement
  - A distributed system is a system where I can't get my work done because a computer has failed that I never heard of
Can software self-adapt to environment changes?
Can software self-adapt to environment changes?

- A strong international research wave
Can software self-adapt to environment changes?

• A strong international research wave

• R. de Lemos et al., Software engineering for self-adaptive systems. Dagstuhl Seminar 2009
• E. Di Nitto et al., A journey to highly dynamic, self-adaptive service-based applications. ASE Journal, 2008
• Software Engineering for Adaptive and Self-Managing Systems (SEAMS), starting 2006
  (previous Workshops on Self-Healing (Self-Managed) Systems (WOSS))
Can software self-adapt to environment changes?

• A strong international research wave
    (previous Workshops on Self-Healing (Self-Managed) Systems (WOSS) )

• How can formal methods help?
Can software self-adapt to environment changes?

• A strong international research wave
  
  • R. de Lemos et al., Software engineering for self-adaptive systems. Dagstuhl Seminar 2009
  • E. Di Nitto et al., A journey to highly dynamic, self-adaptive service-based applications. ASE Journal, 2008
  • Software Engineering for Adaptive and Self-Managing Systems (SEAMS), starting 2006
    (previous Workshops on Self-Healing (Self-Managed) Systems (WOSS) )

• How can formal methods help?
  
  • C. Ghezzi, G. Tamburrelli, "Reasoning on Non Functional Requirements for Integrated Services”, RE 2009
  • A. Filieri, C. Ghezzi, G. Tamburrelli, “Run-time efficient probabilistic model checking”, ICSE 2011
  • A. Filieri, C. Ghezzi, G. Tamburrelli, “Supporting Self-adaptation via Quantitative Verification and Sensitivity Analysis at Run Time, to appear on TSE
The autonomic feedback loop
The autonomic feedback loop
Revisiting the runtime structure for self-adaptation

- Detect exogenous changes
- Understand whether they lead to violation of requirements
- React accordingly by changing behavior
Revisiting the runtime structure for self-adaptation

- Detect exogenous changes
- Understand whether they lead to violation of requirements
- React accordingly by changing behavior
Revisiting the runtime structure for self-adaptation

- Detect exogenous changes
- Understand whether they lead to violation of requirements
- React accordingly by changing behavior
Revisiting the runtime structure for self-adaptation

- Detect exogenous changes
- Understand whether they lead to violation of requirements
- React accordingly by changing behavior
Formal methods @ runtime

- A model of S and E must be kept @ runtime
- E's model updated through monitoring and machine learning
- Verification @ runtime to check satisfaction of R
- Self-adaptation triggered by violations
An exemplary framework

- QoS requirements

  - performance (response time), reliability (probability of failure), cost (energy consumption)
An exemplary framework

- QoS requirements
  - performance (response time), reliability (probability of failure), cost (energy consumption)

Sources of uncertainty (and change)
Formal methods @ work

• S, E formalized via probabilistic Markovian models for non functional requirements (reliability, performance, energy consumption)

• R formalized via probabilistic temporal logic, e.g. PCTL
  • probability that workflow leading to purchase succeeds must be greater than .98
    \[ P(\Diamond s=\text{success}) > .98 \]

• Automatic verification via Model Checking
  • e.g. PRISM, M. Kwiatkowska, Univ. Oxford
A simplified case-study

![Diagram showing the sequence of events from Login to Logout with options for normal or express shipping.]
A simplified case-study

Returning customers vs new customers
A simplified case-study

Returning customers vs new customers

Selected requirements:

R1: “Probability of success is > 0.8”
R2: “Probability of a ExpShipping failure for a ReturningCustomer < 0.035”
Assumptions

User profile assumptions

<table>
<thead>
<tr>
<th>$D_{u,n}$</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{u,1}$</td>
<td>$P$(User is a RC)</td>
<td>0.35</td>
</tr>
<tr>
<td>$D_{u,2}$</td>
<td>$P$(RC chooses express shipping)</td>
<td>0.5</td>
</tr>
<tr>
<td>$D_{u,3}$</td>
<td>$P$(NC chooses express shipping)</td>
<td>0.25</td>
</tr>
<tr>
<td>$D_{u,4}$</td>
<td>$P$(RC searches again after a buy operation)</td>
<td>0.2</td>
</tr>
<tr>
<td>$D_{u,5}$</td>
<td>$P$(NC searches again after a buy operation)</td>
<td>0.15</td>
</tr>
</tbody>
</table>

External service assumptions (reliability)

<table>
<thead>
<tr>
<th>$D_{s,n}$</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{s,1}$</td>
<td>$P$(Login)</td>
<td>0.03</td>
</tr>
<tr>
<td>$D_{s,2}$</td>
<td>$P$(Logout)</td>
<td>0.03</td>
</tr>
<tr>
<td>$D_{s,3}$</td>
<td>$P$(NrmShipping)</td>
<td>0.05</td>
</tr>
<tr>
<td>$D_{s,4}$</td>
<td>$P$(ExpShipping)</td>
<td>0.05</td>
</tr>
<tr>
<td>$D_{s,5}$</td>
<td>$P$(CheckOut)</td>
<td>0.1</td>
</tr>
</tbody>
</table>
The trio S, E, R
The trio S, E, R

R1: Probability of success > 0.8
R2: Probability of ExpShipping failure for ReturningCustomer < 0.035
Verification @ runtime as a trigger for adaptation

- The model can have a predictive nature
  - Requirements violation on model predicts future violations
- This may lead to preventive adaptation prior to violations
- Otherwise it leads to self-healing adaptation
Models & Verification @ runtime: challenges

• Paradigm change
  • The development-time / run-time boundary fades
• Real-time constraints prevent applicability of current techniques
Towards efficient verification @ runtime

Make verification incremental

• Instead of checking the model after any change, just try to restrict the check to what has changed

• easier to say than do!
Incrementality by parameterization

- Requires anticipation of changing parameters, represented as symbolic variable
  - The model is partly numeric and partly symbolic
- Evaluation of the verification condition requires partial evaluation (mixed numerical/symbolic processing)
  - Result is a formula (polynomial for reachability on DTMCs)
- Evaluation at run time substitutes actual values to symbolic parameters and is very efficient
Where are we?
Where are we?

• Change is quintessential to software
  • not a nuisance nor something to handle as an afterthought

• Formal methods can set change management on systematic and rigorous grounds that lead to effective and efficient evolution

• They can be brought to runtime to self-manage response to environment changes
Where are we?

- Change is quintessential to software
  - not a nuisance nor something to handle as an afterthought
- Formal methods can set change management on systematic and rigorous grounds that lead to effective and efficient evolution
- They can be brought to runtime to self-manage response to environment changes
- **How can they support a holistic response to changes throughout the software lifetime?**
Looking forward: continuous assurance
Looking forward: continuous assurance

- Change of perspective
  - Development and operation viewed as a continuum
- Focus on assurance that system complies with requirements drives both development and operation
Looking forward: continuous assurance

• Change of perspective
  • Development and operation viewed as a continuum
• Focus on assurance that system complies with requirements drives both development and operation
• Focus on continuous assurance requires revisiting verification methods in the light of continuous change
Looking forward

• Software development has become increasingly incremental, change-oriented, agile

B. Meyer, "Agile! The good, the Hype and the Ugly", Springer, 2015
Looking forward

• Software development has become increasingly incremental, change-oriented, agile

B. Meyer, "Agile! The good, the Hype and the Ugly", Springer, 2015

• The ugly
  
  • deprecation of upfront activities: requirements (replaced by user stories), specification (replaced by tests), modeling…
Looking forward

- Software development has become increasingly incremental, change-oriented, *agile*

  B. Meyer, "Agile! The good, the Hype and the Ugly", Springer, 2015
Looking forward

- Software development has become increasingly incremental, change-oriented, *agile*

  B. Meyer, "Agile! The good, the Hype and the Ugly", Springer, 2015

- The good
  - continuous testing: do not wait for a complete system...
Looking forward

• Software development has become increasingly incremental, change-oriented, *agile*

B. Meyer, "Agile! The good, the Hype and the Ugly", Springer, 2015
Looking forward

- Software development has become increasingly incremental, change-oriented, *agile*

  B. Meyer, "Agile! The good, the Hype and the Ugly", Springer, 2015

- Get rid of the ugly and move the good one step further

  - automate upfront activities and integrate them in agile development

  - *towards verification driven development*
What needs to be done
What needs to be done

• How can we integrate modelling and verification into iterative, agile development?
What needs to be done

• How can we integrate modelling and verification into iterative, agile development?

• Support incomplete, partial specifications

What needs to be done

• How can we integrate modelling and verification into iterative, agile development?

• Support incomplete, partial specifications


• Support reasoning about changes: support incremental verification
Support to incomplete, partial specifications

• Given an incomplete system (model) S, and a set of properties P to be met by S

• Verification can return YES, NO, MAYBE

• In the MAYBE case, it should compute proof obligations for the incomplete parts

• Completion only verifies proof obligations
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- a functionality whose detail model is postponed

\[ E(\neg(\text{Update}) \cup \text{Peach}) \]
Example of an incomplete model

• An FSM where certain states stand for an unspecified FSM

• a functionality whose detail model is postponed

\[ E(\neg(\text{قيقة}) \cup (\text{ملكة})) \]
Example of an incomplete model

• An FSM where certain states stand for an unspecified FSM

• a functionality whose detail model is postponed

\[ E(\neg(\text{\rule{1cm}{0.15em}})) U(\text{\rule{1cm}{0.15em}}) \]
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed

\[ E(\neg(\text{?????}) \cup (\text{?????})) \]
Example of an incomplete model

• An FSM where certain states stand for an unspecified FSM

• a functionality whose detail model is postponed

\[ E(\neg ( \text{??}) ) U ( \text{??} ) \]
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed

\[ E(\neg(\; ) \cup(\; )) \]
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed

\[ E(\neg(\text{ } ) \cup(\text{ } ) ) \]
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed

\[
E(\neg(\bullet))U(\heart) \\
E(\neg(G\neg(\bullet))) \\
E(\neg(\bullet)U(\heart))
\]
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed

\[ E(\neg(\text{}) )U(\text{}) \]

\[ E(\neg(\text{ }) )U(\text{ }) \]

\[ \text{Ep}(G\neg(\text{ })) \]

\[ E(\neg(\text{ }) )U(\text{ }) \]
Example of an incomplete model

- An FSM where certain states stand for an unspecified FSM
- A functionality whose detail model is postponed

\[ E(\neg (\text{��态1}) \cup (\text{状态2})) \]

\[ E_p(G\neg (\text{状态2})) \]

\[ E(\neg (\text{状态1}) \cup (\text{状态2})) \]
Partial models vs. incremental changes

• Initial decomposition affects the kind of incrementally we get

• Can we achieve incremental verification independent of hierarchical decomposition?

• Can a general approach to incremental verification be found independent of model/program and property language to verify?
Supporting incremental verification: the goal
Supporting incremental verification: the goal

- Given a system (model) $S$, and a set of properties $P$ to be met by $S$
  - Change = new pair $S'$, $P'$ where $S' = S + \Delta S$ and $P' = P + \Delta P$
Supporting incremental verification: the goal

- Given a system (model) $S$, and a set of properties $P$ to be met by $S$
  - Change = new pair $S'$, $P'$ where $S' = S + \Delta S$ and $P' = P + \Delta P$
- Let $\prod$ be the proof of $S$ against $P$
  - The proof $\prod'$ of $P'$ against $S'$ can be done by just performing a proof increment $\Delta \prod$ such that $\prod' = \prod + \Delta \prod$
Supporting incremental verification: the goal

• Given a system (model) $S$, and a set of properties $P$ to be met by $S$
  • Change = new pair $S'$, $P'$ where $S' = S + \Delta S$ and $P' = P + \Delta P$

• Let $\prod$ be the proof of $S$ against $P$
  • The proof $\prod'$ of $P'$ against $S'$ can be done by just performing a proof increment $\Delta \prod$ such that $\prod' = \prod + \Delta \prod$

• Expectations:
  • $\Delta \prod$ easy and efficient to perform
  • $\Delta \prod$ explains the change
Towards a general theory of incremental verification

Syntactic-semantic incremental verification

- A general approach
  - independent of the artifact
    - model, program, ....
  - independent of the property language
    - LTL, CTL, PCTL, Hoare logic, Matching Logic, ...
  - unconstrained possible changes in artifact (and in property)
Intuition: back to the 1970's

- Incremental parsing can detect the part of the syntax tree to rebuild and hook it into the unchanged part

\[ 6 \times 5 + 3 \times 2 + 1 \]
\[ 6 \times 5 + 4 + 1 \]
Intuition: back to the 1970's

- Incremental parsing can detect the part of the syntax tree to rebuild and hook it into the unchanged part

\[ 6 \times 5 + 3 \times 2 + 1 \rightarrow 6 \times 5 + 4 + 1 \]
Intuition: back to the 1970's

- Incremental parsing can detect the part of the syntax tree to rebuild and hook it into the unchanged part.

\[ 6^*5+3^*2+1 \rightarrow 6^*5+4+1 \]
Intuition: adding semantics
Intuition: adding semantics

• Semantic functions can be attached to syntactic rules and computed traversing the syntax tree (Knuth's attribute grammars)

• The formalism is Turing complete

• Attributes can be evaluated by bottom-up traversal
Intuition: adding semantics

• Semantic functions can be attached to syntactic rules and computed traversing the syntax tree (Knuth's attribute grammars)

• The formalism is Turing complete

• Attributes can be evaluated by bottom-up traversal

• Any verification algorithm can be expressed via attribute functions
Current stage

- Using the approach in significant case studies
  - Incremental reliability analysis of Web service compositions
  - Complete semantics of Kernel C and verification of Matching Logic properties
  - Building a syntactic-semantic incremental engine
Looking backwards

What did I learn beyond software engineering?
Looking backwards

What did I learn beyond software engineering?

3 wisdom pills
Identify a long-term wide-spectrum problem

- As opposed to "pick and play"
- It provides focus to your work
- It gives you multiple viewpoints
- Helps making your group cohesive
Identify a long-term wide-spectrum problem

- As opposed to "pick and play"
- It provides focus to your work
- It gives you multiple viewpoints
- Helps making your group cohesive
Identify a broad reference community

- Helps you establish long-term interactions and collaborations
- Helps exposing your students/collaborators to an international arena
- Helps you calibrating choice of research targets
Identify a broad reference community

• Helps you establish long-term interactions and collaborations

• Helps exposing your students/collaborators to an international arena

• Helps you calibrating choice of research targets
Nurture your personal environment

No man is an island,
Entire of itself.
Each is a piece of the continent,
A part of the main.
If a clod be washed away by the sea,
Europe is the less.
As well as if a promontory were.
As well as if a manor of thine own
Or of thine friend's were.
Each man's death diminishes me,
For I am involved in mankind.
Therefore, send not to know
For whom the bell tolls,
It tolls for thee.

John Donne
Meditation 17
Devotions upon Emergent Occasions
Nurture your personal environment

No man is an island,
Entire of itself.
Each is a piece of the continent,
A part of the main.
If a clod be washed away by the sea,
Europe is the less.
As well as if a promontory were.
As well as if a manor of thine own
Or of thine friend's were.
Each man's death diminishes me,
For I am involved in mankind.
Therefore, send not to know
For whom the bell tolls,
It tolls for thee.

John Donne
Meditation 17
Devotions upon Emergent Occasions
Thank You
Students & PostDocs
Family
Friends

a grand tour from Japan to Bolivia, Croatia, Zurich, Elba Island, and the Ardennes
IWSSD 1998, Ise Shima, Japan
Onsen expedition
Axel Emeritus & Porcini expedition
Thanks!!!

Questions?