Hot research topics around agile

Agile Development Processes

Eric Knauss

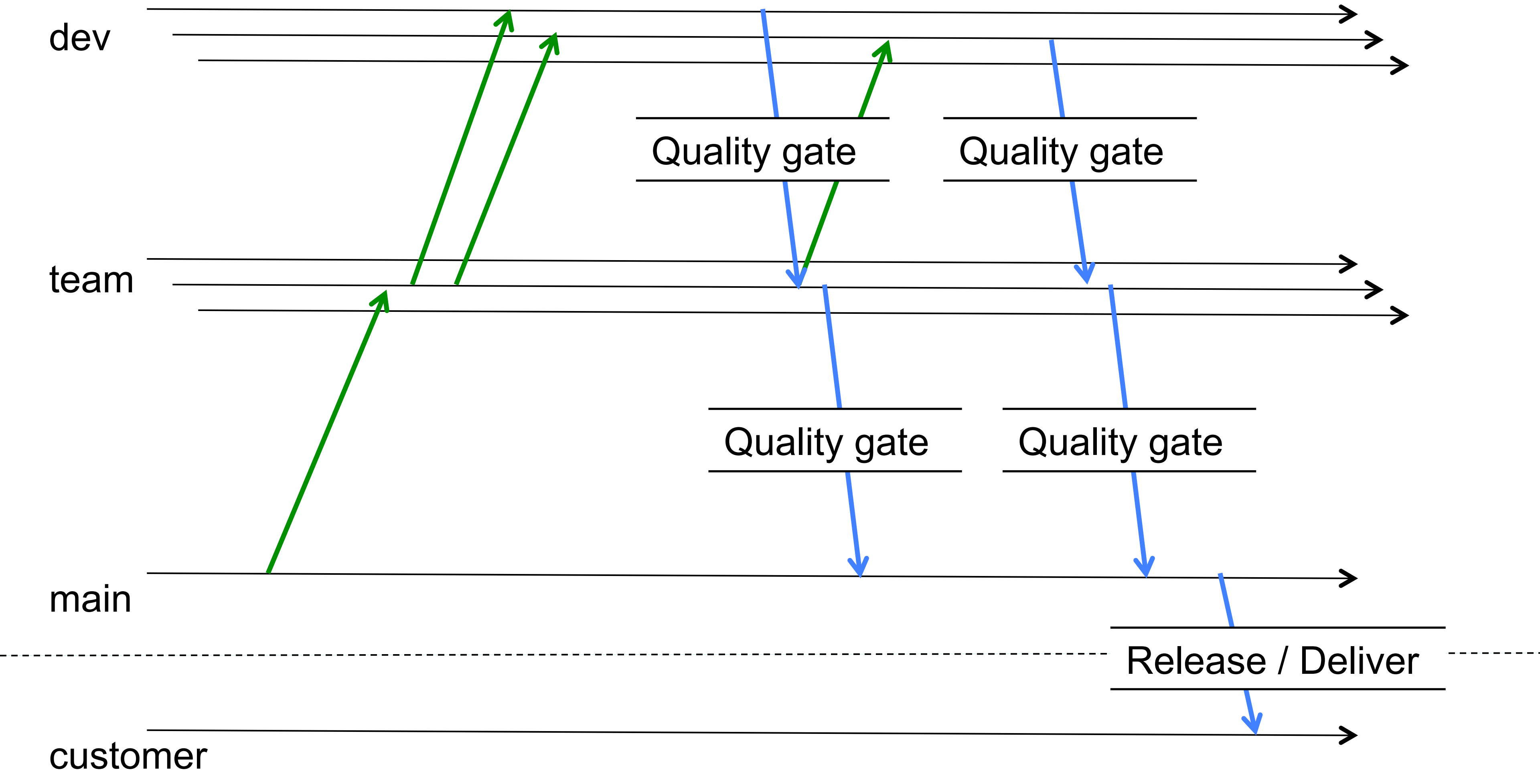
Organizational

- Please send me slides with your project presentation no later than Thursday, 11am
  - Avoid switching time!
  - Try to stay below 10min!
  - Present what is cool about your solution
  - May present challenges / lessons learnt

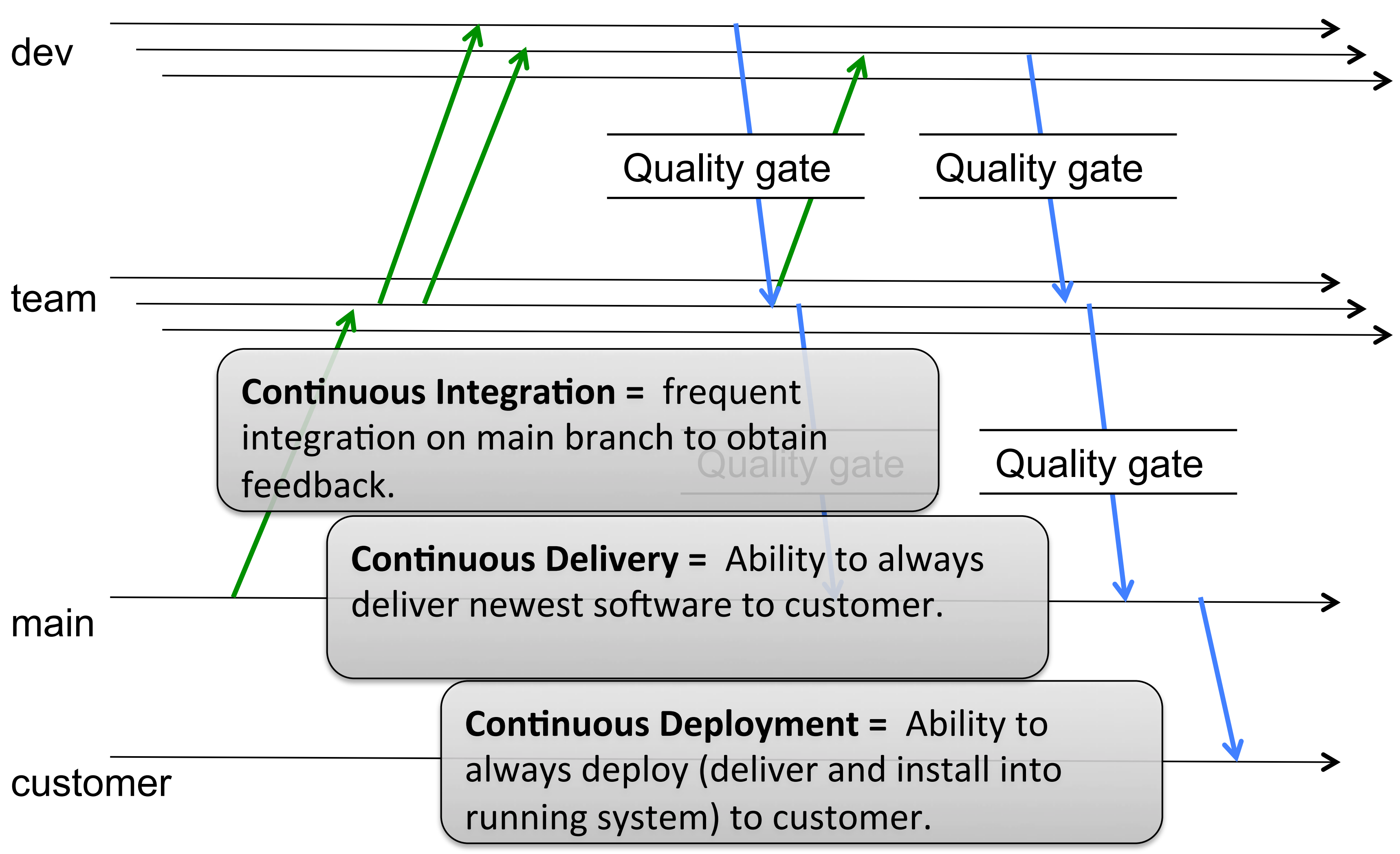
Outline

- Agile maturity and information flows
  - Information and knowledge flows already discussed
  - Lucas Gren: Agile maturity models
- Continuous Integration and Deployment
  - Current work in Software Center #1
- Agile and Architecture
  - ICSE 15 paper
  - Caffea model
- Other topics (discussed, no slides)
  - Agile engineering
  - Measure effectivity / efficiency of agile methods

Continuous Integration (CI) Continuous Deployment (CD)



Defining CI and CD

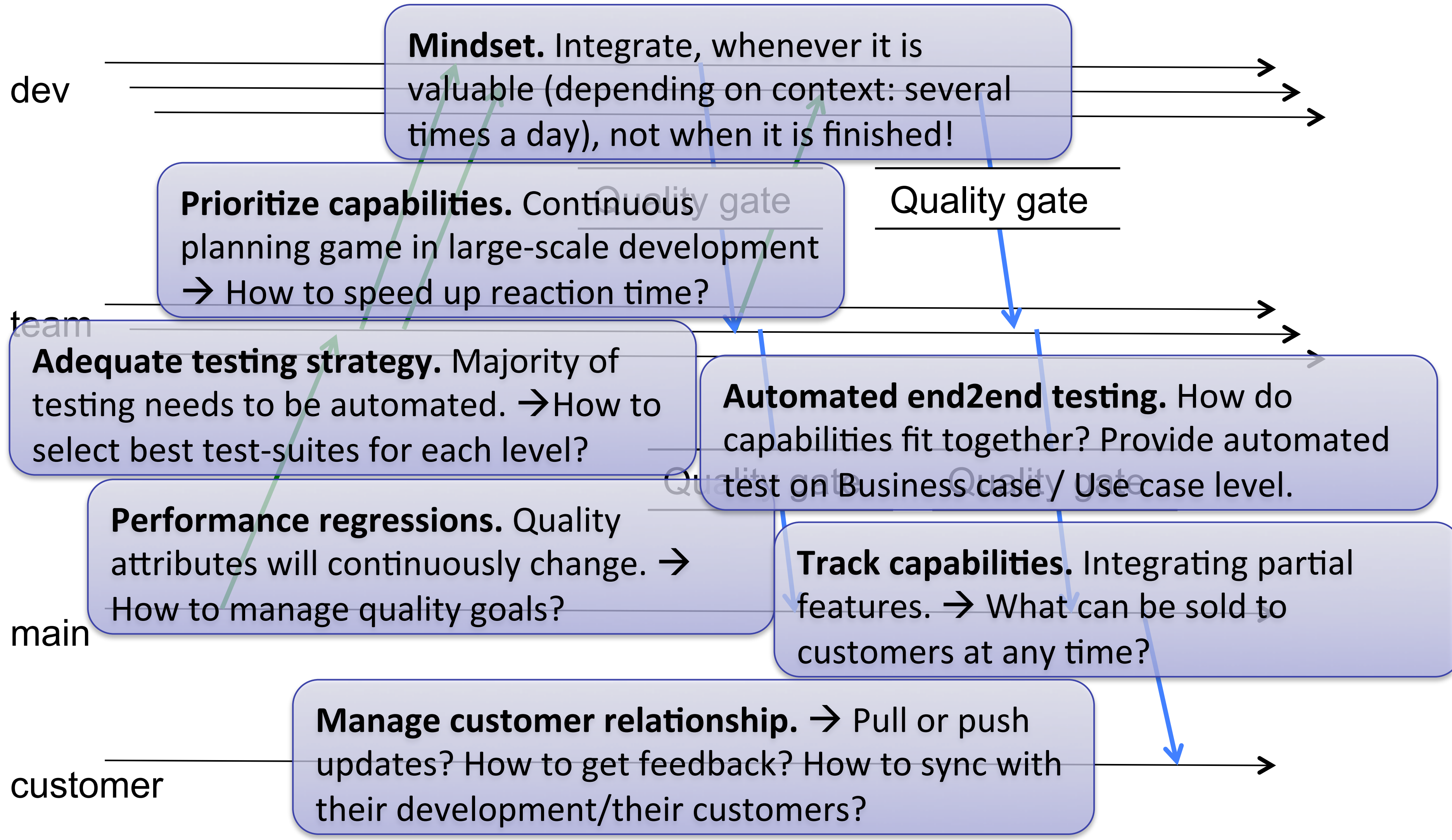


Continuous Integration = frequent integration on main branch to obtain feedback.

Continuous Delivery = Ability to always deliver newest software to customer.

Continuous Deployment = Ability to always deploy (deliver and install into running system) to customer.

CI and CD challenges



Mindset. Integrate, whenever it is valuable (depending on context: several times a day), not when it is finished!

Prioritize capabilities. Continuous planning game in large-scale development → How to speed up reaction time?

Adequate testing strategy. Majority of testing needs to be automated. →How to select best test-suites for each level?

Automated end2end testing. How do capabilities fit together? Provide automated test on Business case / Use case level.

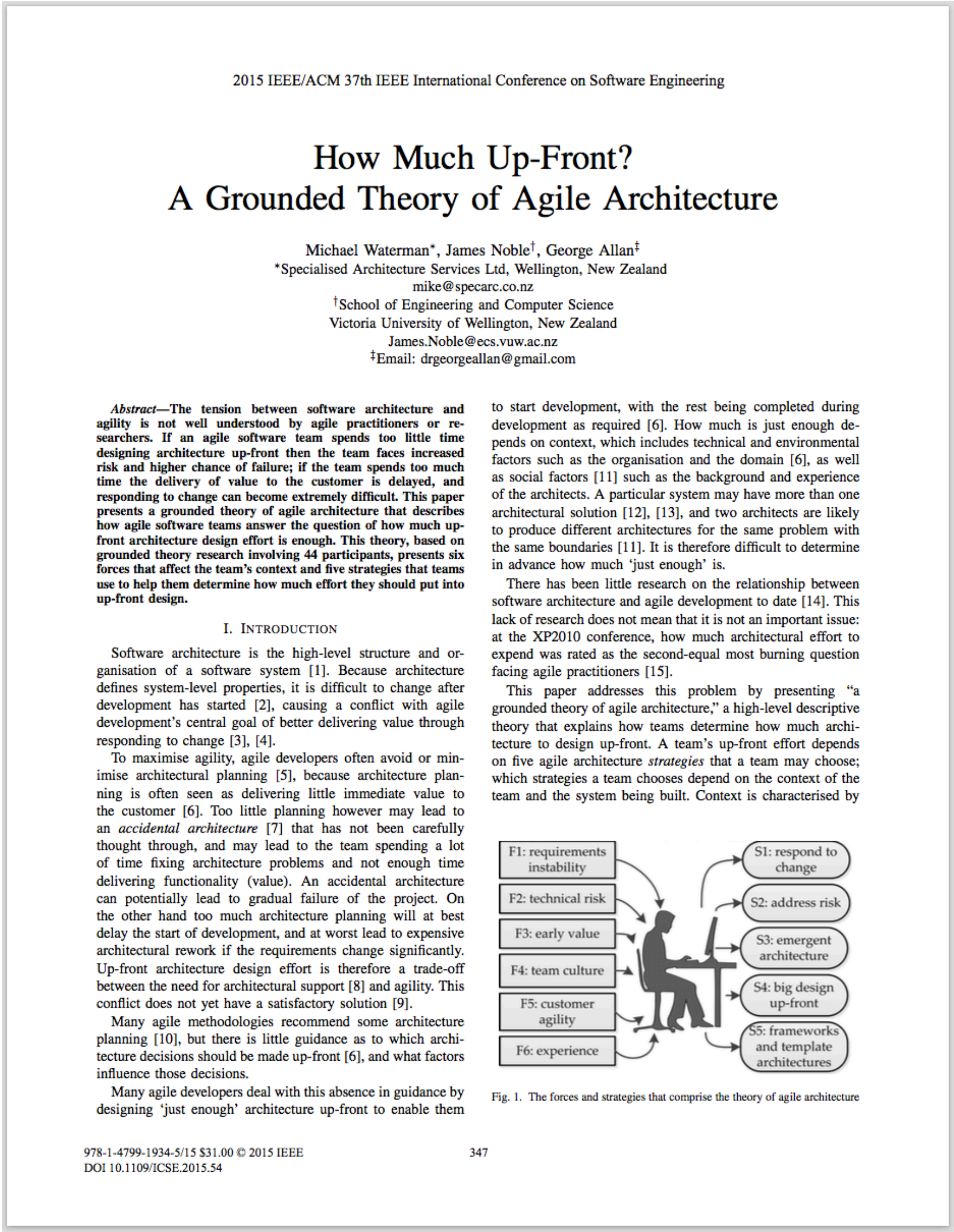
Performance regressions. Quality attributes will continuously change. → How to manage quality goals?

Track capabilities. Integrating partial features. → What can be sold to customers at any time?

Manage customer relationship. → Pull or push updates? How to get feedback? How to sync with their development/their customers?

Agile and Architecture

Michael Waterman, James Noble, George Allan: **How Much Up-Front? A Grounded theory of Agile Architecture**. In: Proc. of Int. Conf. on Software Engineering, pg. 347-358, Florence, Italy, 2015



2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

How Much Up-Front?
A Grounded Theory of Agile Architecture

Michael Waterman*, James Noble†, George Allan‡
*Specialised Architecture Services Ltd, Wellington, New Zealand
mike@specarc.co.nz
†School of Engineering and Computer Science
Victoria University of Wellington, New Zealand
James.Noble@ecs.vuw.ac.nz
‡Email: drgeorgeallan@gmail.com

Abstract—**The tension between software architecture and agility is not well understood by agile practitioners or researchers. If an agile software team spends too little time designing architecture up-front then the team faces increased risk and higher chance of failure; if the team spends too much time the delivery of value to the customer is delayed, and responding to change can become extremely difficult. This paper presents a grounded theory of agile architecture that describes how agile software teams answer the question of how much up-front architecture design effort is enough. This theory, based on grounded theory research involving 44 participants, presents six forces that affect the team's context and five strategies that teams use to help them determine how much effort they should put into up-front design.**

I. INTRODUCTION

Software architecture is the high-level structure and organisation of a software system [1]. Because architecture defines system-level properties, it is difficult to change after development has started [2], causing a conflict with agile development's central goal of better delivering value through responding to change [3], [4].

To maximise agility, agile developers often avoid or minimise architectural planning [5], because architecture planning is often seen as delivering little immediate value to the customer [6]. Too little planning however may lead to an *accidental architecture* [7] that has not been carefully thought through, and may lead to the team spending a lot of time fixing architecture problems and not enough time delivering functionality (value). An accidental architecture can potentially lead to gradual failure of the project. On the other hand too much architecture planning will at best delay the start of development, and at worst lead to expensive architectural rework if the requirements change significantly. Up-front architecture design effort is therefore a trade-off between the need for architectural support [8] and agility. This conflict does not yet have a satisfactory solution [9].

Many agile methodologies recommend some architecture planning [10], but there is little guidance as to which architecture decisions should be made up-front [6], and what factors influence those decisions.

Many agile developers deal with this absence in guidance by designing 'just enough' architecture up-front to enable them to start development, with the rest being completed during development as required [6]. How much is just enough depends on context, which includes technical and environmental factors such as the organisation and the domain [6], as well as social factors [11] such as the background and experience of the architects. A particular system may have more than one architectural solution [12], [13], and two architects are likely to produce different architectures for the same problem with the same boundaries [11]. It is therefore difficult to determine in advance how much 'just enough' is.

There has been little research on the relationship between software architecture and agile development to date [14]. This lack of research does not mean that it is not an important issue: at the XP2010 conference, how much architectural effort to expend was rated as the second-equal most burning question facing agile practitioners [15].

This paper addresses this problem by presenting "a grounded theory of agile architecture," a high-level descriptive theory that explains how teams determine how much architecture to design up-front. A team's up-front effort depends on five agile architecture *strategies* that a team may choose; which strategies a team chooses depend on the context of the team and the system being built. Context is characterised by

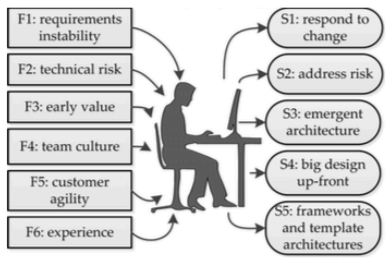
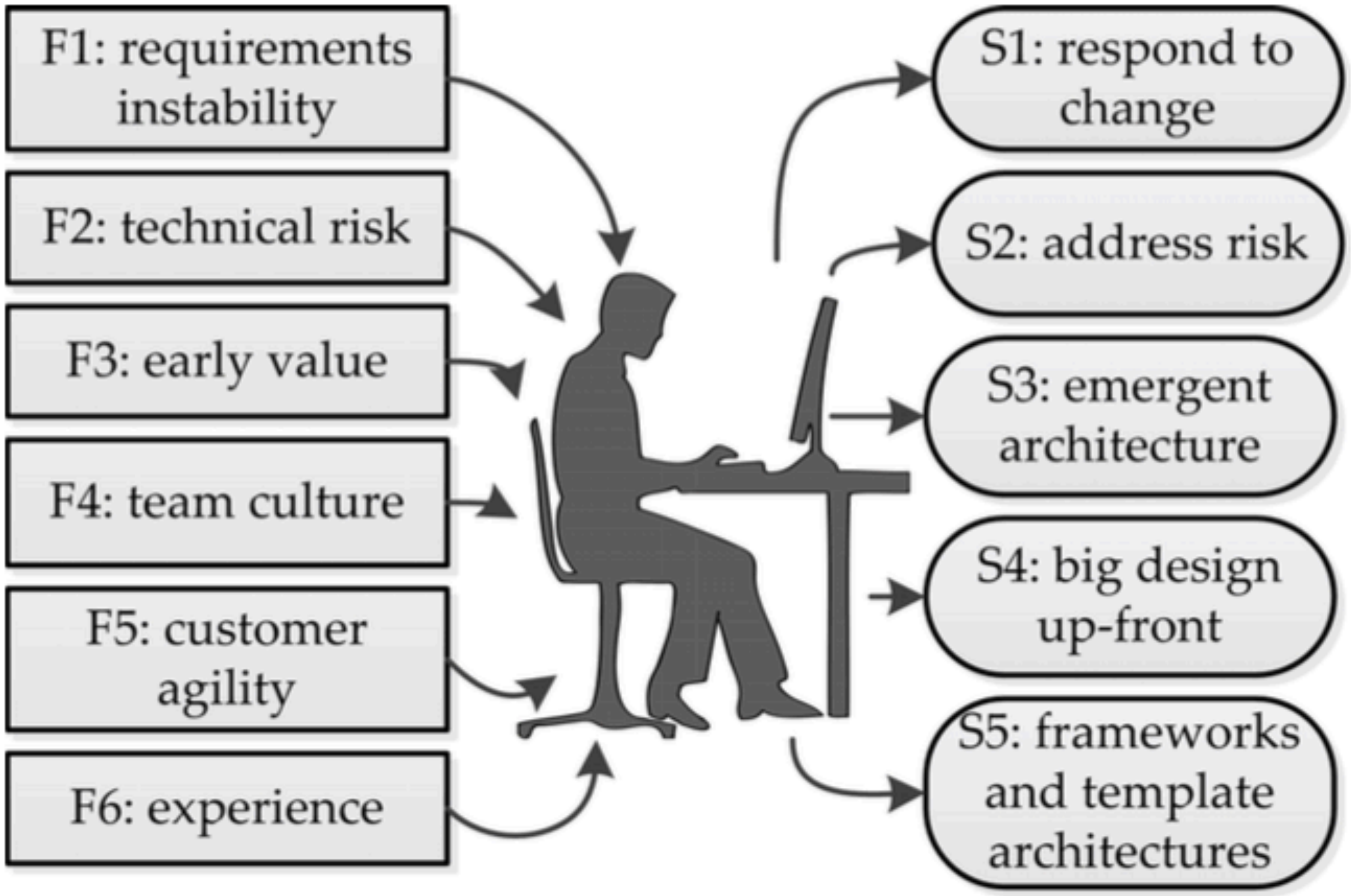


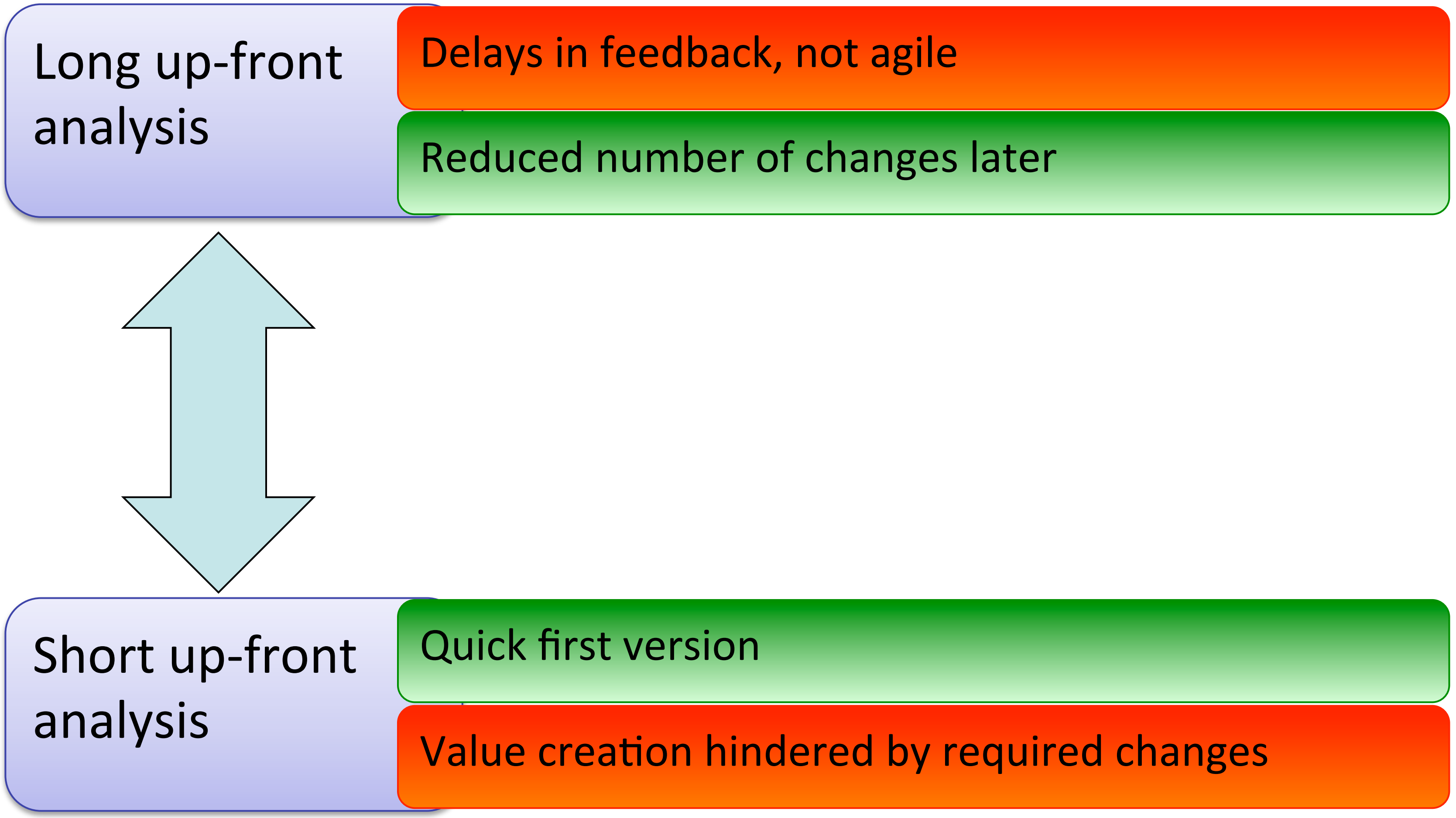
Fig. 1. The forces and strategies that comprise the theory of agile architecture

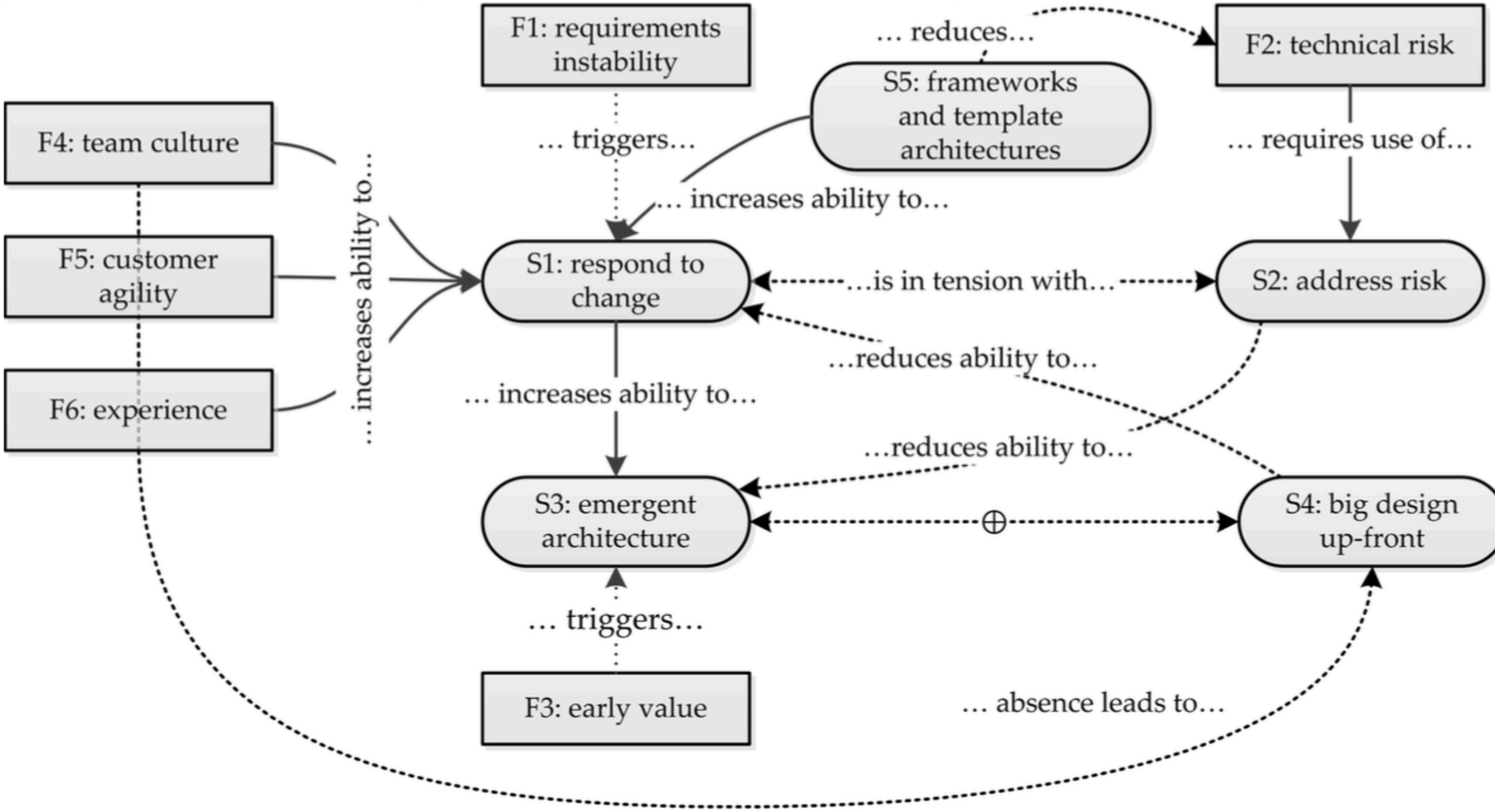
Agile Architecture: Context and Strategies



Michael Waterman, James Noble, George Allan: **How Much Up-Front? A Grounded theory of Agile Architecture**. In: Proc. of Int. Conf. on Software Engineering, pg. 347-358, Florence, Italy, 2015

Tradeoff


Long up-front analysis
Delays in feedback, not agile
Reduced number of changes later
Short up-front analysis
Quick first version
Value creation hindered by required changes



Michael Waterman, James Noble, George Allan: **How Much Up-Front? A Grounded theory of Agile Architecture**. In: Proc. of Int. Conf. on Software Engineering, pg. 347-358, Florence, Italy, 2015