Refactoring Infrastructure: Reducing emissions and energy one step at a time

Chris Parnin, Carsten Görg {vector,goerg}@cc.gatech.edu

PROPOSAL

Infrastructure is strangely mundane and invisible to the casual observer yet fundamental and permeable throughout every aspect of society. Infrastructure includes everything our businesses and homes depend on everyday yet we give little thought in passing: the plumbing and water filtration plants providing fresh water, the electricity delivery networks providing power, farming and transportation networks providing food, and furnaces and insulation providing warmth.

In the coming years, rapid climate change and unsustainable energy resources may require companies, city planners, and government officials to enact large-scale changes to society and the underlying infrastructure. Changes to infrastructure from the top-down can be a notoriously difficult proposition as unexpected interactions and consequences could cause skyrocketing cost and frequent disruptions to service. Further, infrastructure is an entity that not one person controls and requires the coordination of many to make a change – often in a bottom-up manner. In support of this task, we propose *infrastructure refactoring*: a set of tools for applying function-preserving changes to infrastructure without disruption to those services, while changing the internal workings. Such a kind of toolkit would allow engineers and officials to identify the impact of a proposed change to existing infrastructure, outline the steps for making and testing the change, and to support planning *what-if* scenarios for deciding the most suitable technologies for a local region.

Software developers and researchers have several decades of experience in enacting large-scale changes to complex systems. The strategies, lessons and tools we have identified and developed in support of these efforts also have direct applications and analogies to adapting the infrastructure that society depends on. One technique in particular, *refactoring*, has been widely adopted in practice and has received direct tool support in programming environments. Refactoring is the process of changing the structure of a program without changing the way that it behaves. In his book on the subject, Fowler catalogs 72 different refactorings, ranging from localized changes such as Extract Local Variable, to more global changes such as Extract Class. Based on his experience, Fowler claims that refactoring provides significant benefits: it can help programmers add functionality, fix bugs, and understand software.

There are several analogies between the process of refactoring software and refactoring infrastructure. For example, Fowler describes *code smells*, a set of symptoms indicative (but not confirmatory) of the presence of certain design problems, as an indication of when to perform certain types of refactorings. In place of code smells, *energy smells*, can be used to identify problematic energy usage patterns and suggest possible transformations to those problems. As we have several metrics and visualizations for finding code smells, similar ones can be devised for detecting energy smells in building and city plans. Another analogy is as programmers are well-supported with automated refactoring tools in their programming environments, planners could be well-supported with automated refactoring tools for adapting buildings and cities. When viewing plans of buildings and cities, a rich set of refactoring transformations would allow planners and engineers to quickly apply, plan and analyze energy-reducing changes to buildings and cities. These tools would allow planners to detect problems, evaluate several options, and select the best course of action for their regional needs. Yet another analogy the software engineering community can offer: as we have collected a catalog of refactorings for capturing collective knowledge and building a common vocabulary, a catalog of infrastructure refactorings can also facilitate officials in knowing the options they have available and in providing a framework for tools and procedures when transforming infrastructure.

Current trends in energy distribution suggest that making changes to infrastructure will not simply be isolated to a few centralized locations, but rather require the coordination and cooperation of many decentralized locations. For example, Lichtblick (a German eco-supplier of electric energy) and Volkswagen are planing to deploy 100,000 miniature combined heat and power plants in private households within the next year. These miniature plants are powered by natural gas and can produce the same amount of energy as two nuclear power plants. There are numerous benefits for decentralization: use of waste heat for heating, more flexible demand driven energy production, less emissions than coal power plants; however, this wide-spread deployment will need the support of tools and shared know-how to happen effectively. By leveraging a mature infrastructure refactoring toolkit inspired from software engineering, many households could apply an Introduce Mini-Power Plant refactoring.

In summary, contributions software engineering can make toward addressing climate change challenge include:

- a catalog of function-preserving but emission-and-energy-reducing transformations to infrastructure,
- metrics and visualizations techniques for identifying energy smells,
- automated tools for transforming models of infrastructure.

CHALLENGES

To make these strategies and tools usable, we have to overcome several challenges.

An immediate concern is to what extent we have models of existing infrastructure that are suitable for automated analysis. Certainly, at least blueprints and topographical maps exist for much of the infrastructure; however, extra effort may be required to annotate and import this information into a form suitable for analyzing and simulating changes to it.

In addition to modeling infrastructure, city planners and officials may need more extensive test cases to evaluate proposed changes to a system such as handling peak loads, failing gracefully with degradation, and meeting safety regulations. In the same way we analyze control and data dependencies in software, we must devise analysis techniques for identifying risks and impact to other dependencies of infrastructure such as increased water demands a new component may impose.

Although there are many analogies we can derive from software engineering, we have to address important differences in how software changes and infrastructure (much of it physical) changes. Unlike software, a change to infrastructure often cannot be instantaneous – not only because of physical constraints but also caused by constraints arising from coordination with other organizations and stakeholders. In response, tools may need to focus on generating workflows that help officials manage the schedule and procedures for making a change (who do I call to make this happen?).

Numerous contextual and regional information need to be gathered and provided to local officials as part of developing their models. Many decentralized and alternative energy sources require local information such as average sunlight length and orientation or wind strength and direction. An even simpler constraint is shape and variation in roof tops. For instance, a university or company such as Microsoft owns hundreds to thousands of buildings, each having a distinct architecture. If such an organization wanted to apply a refactoring such as Introduce Solar Thermal Water Heating, then it needs models that would easily handle variations between their buildings in order to evaluate the effectiveness of the proposed change.

Discussion points for the workshop:

- What would a description for a Introduce Solar Thermal Water Heater refactoring look like?
- Brainstorm an initial catalog of infrastructure refactorings.
- To what extent models of existing infrastructure can and should be built to support refactoring?